



SDK DOCUMENT

Java

VERSION 1.2.2

Serino Inc.

Revision History

DATE	REVISION	AUTHOR	DESCRIPTION
2022-10-14	1.2.2	Chester Canilao	<ul style="list-style-type: none"> • Include version number in SDK library filename/properties • Aligned SDK Document version in the current application release version
2022-03-23	2.3.1	Winston Carmelo	Updated the flow for the timeout under section 3.2. And updated the diagram of figure 4.
2022-02-14	2.3.0	Winston Carmelo	Added AutoTicket support for following transactions: <ul style="list-style-type: none"> • Sale • Refund • Hotel PreAuthorization • Hotel Completion • Account Verification • Duplicate • Reverse • Logon
2021-07-26	2.2.0	Raniel Antonio	Add Support for Multiple Ingenico Terminal Device for Pay@Table Mode under section 2.3 Multiple Client.
2021-04-23	2.1.2	Raniel Antonio	Added External Referral Management command
2020-11-25	2.1.1	Ariel Lagonsin Steven Lester Tan Xedrick Camba	Added Timeout Handling
2020-10-16	2.1.0	Ariel Lagonsin Steven Lester Tan Xedrick Camba	Added Pay@Table section Added Pay@Table methods and enums to Appendix
2020-10-02	2.0.0	Ariel Lagonsin Steven Lester Tan Xedrick Camba	More detailed definition of commands, responses, classes and enums
2020-04-30	1.0.1	Steven Lester Tan Rommel Vallejo	Addition of Java Samples Codes Addition of Sample Code for Serial Terminal Configuration

2020-02-28	1.0.0	Peter Yanong Steven Lester Tan	Starting document
------------	-------	-----------------------------------	-------------------

Contents

Revision History	2
Introduction	6
Initializing a connection from the EPoS to the terminal	7
Setting up a Connection Configuration	7
Creating a Device	7
Multiple Client	8
Request Messages from EPoS to the Terminal	9
Message Construction	9
Timeout Handling	10
Timeout Error Messages	11
Response data from the Terminal to the EPoS	11
IngenicoTerminalResponse	11
REP Fields	12
IngenicoTerminalReportResponse	12
IngenicoTerminalReceiptResponse	12
TerminalStateResponse	13
POSIentifierResponse	13
Handling of Broadcast Messages	13
Commands	14
Payment Management	14
Sale	14
Refund	15
Hotel Pre-Authorization	16
Hotel Completion	18
Account Verification	19
Tax Free Cash Refund	20
Tax Free Credit Card Refund	21
External Referral Management	22
5.2 Transaction Management	23
Cancel	23
Duplicate	24
Reverse	24
Reverse with Transaction ID	25

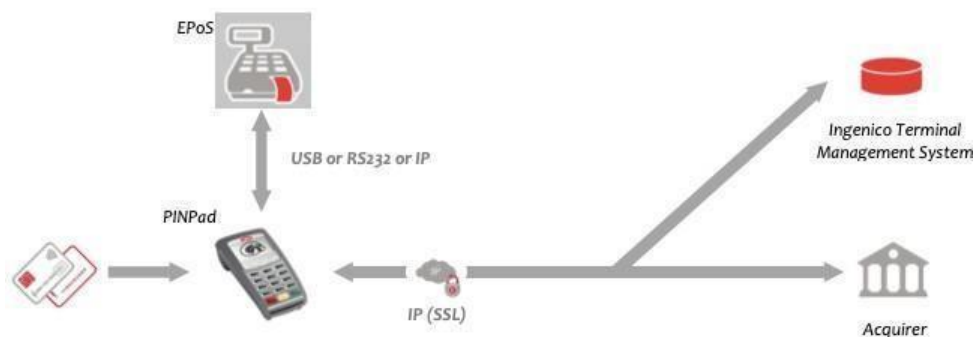
Report Management	26
EOD	26
Banking	27
XBAL	27
ZBAL	28
XML Management	29
Report	29
Ticket	29
SplitR	30
Tax Free	30
Terminal Management	31
Call TMS	31
Logon	31
Reset	32
State	32
PID	33
Pay@Table Mode	34
Communication Flow / Initiating a Transaction	34
Handling Terminal Requests	35
PATRequest	35
Responding to Terminal Requests	36
Confirmation Response	37
XML Response	37
Request Types	39
Table List	39
Table Lock	40
Receipt Request	41
Transaction Outcome	41
Additional Message	42
Split Sale Report Data	42
Final Report Data	43
Ticket	43
Transfer Data	44
EOD Report	45
Table Unlock	45
Appendix	47

ConnectionConfig	47
TerminalAuthBuilder	50
TerminalManageBuilder	51
TerminalReportBuilder	51
IngenicoTerminalResponse	53
IngenicoTerminalReceiptResponse	54
Enums	54
O'Donnell, H. (2019) EPoS interface for TMS helium terminals (semi-integration).	58

1) 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. Java Android minSDKVersion is 21 and for Java Agnostic the SDK's JDK version is 1.8 both variant uses Java 8 version.

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



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

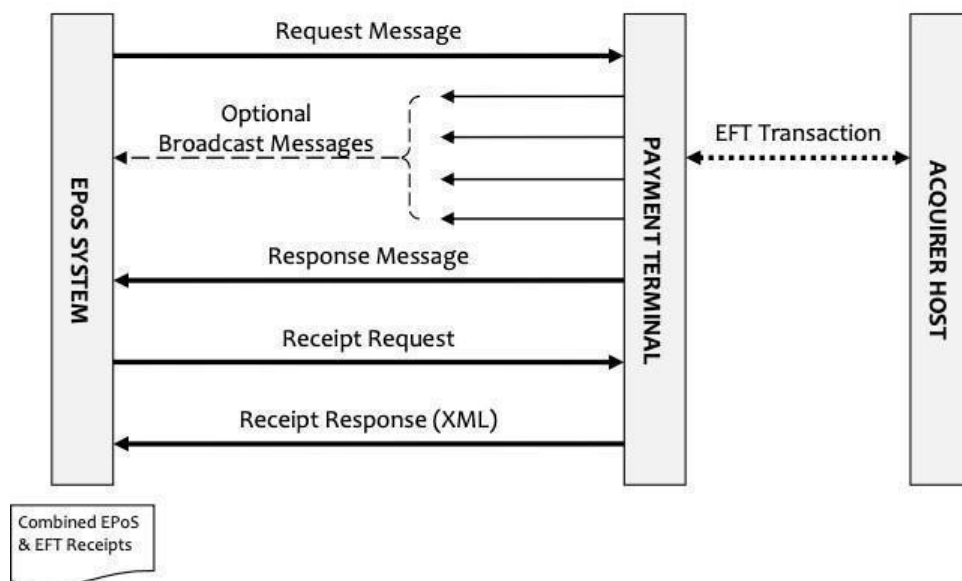


Figure 1: Referenced from Ingenico Documentation (Section 2, Page 9)

- See References section for more details on the reference document hereinafer 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.

2) 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 the following package **com.global.api.terminals**, **com.global.api.services**, **com.global.api.entities** in your class then setup your connection by referring to the sample code below.

2.1 Setting up a Connection Configuration

See appendix 7.1 for more details on ConnectionConfig class.

```
TCP / IP Connection
ConnectionConfig connectionConfig = new ConnectionConfig();
config.setDeviceType(DeviceType.INGENICO);
config.setConnectionMode(ConnectionModes.TCP_IP_SERVER);
config.setPort(18101);
config.setTimeout(65000);
```

```
Serial Connection
ConnectionConfig connectionConfig = new
ConnectionConfig();
config.setDeviceType(DeviceType.INGENICO);
config.setConnectionMode(ConnectionModes.SERIAL);
config.setPort(9);
config.setBaudRate("9600");
config.setDataBits(DataBits.Eight);
config.setParity(Parity.Even);
config.setStopBits(StopBits.One);
config.setTimeout(65000);
```

```
Pay@Table
ConnectionConfig connectionConfig = new ConnectionConfig();
config.setDeviceType(DeviceType.INGENICO);
config.setConnectionMode(ConnectionModes.PAY_AT_TABLE);
config.setPort("18101");
```

Note: For Java (Agnostic) serial communication support, library was integrated to the SDK for this purpose. Java Android currently does not have a support for serial communication

2.2 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.

```
try {
    IDeviceInterface device = DeviceService.create(connectionConfig);
} catch (ApiException e) {
    e.printStackTrace();
}
```

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.

2.3 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 `Socket` 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.

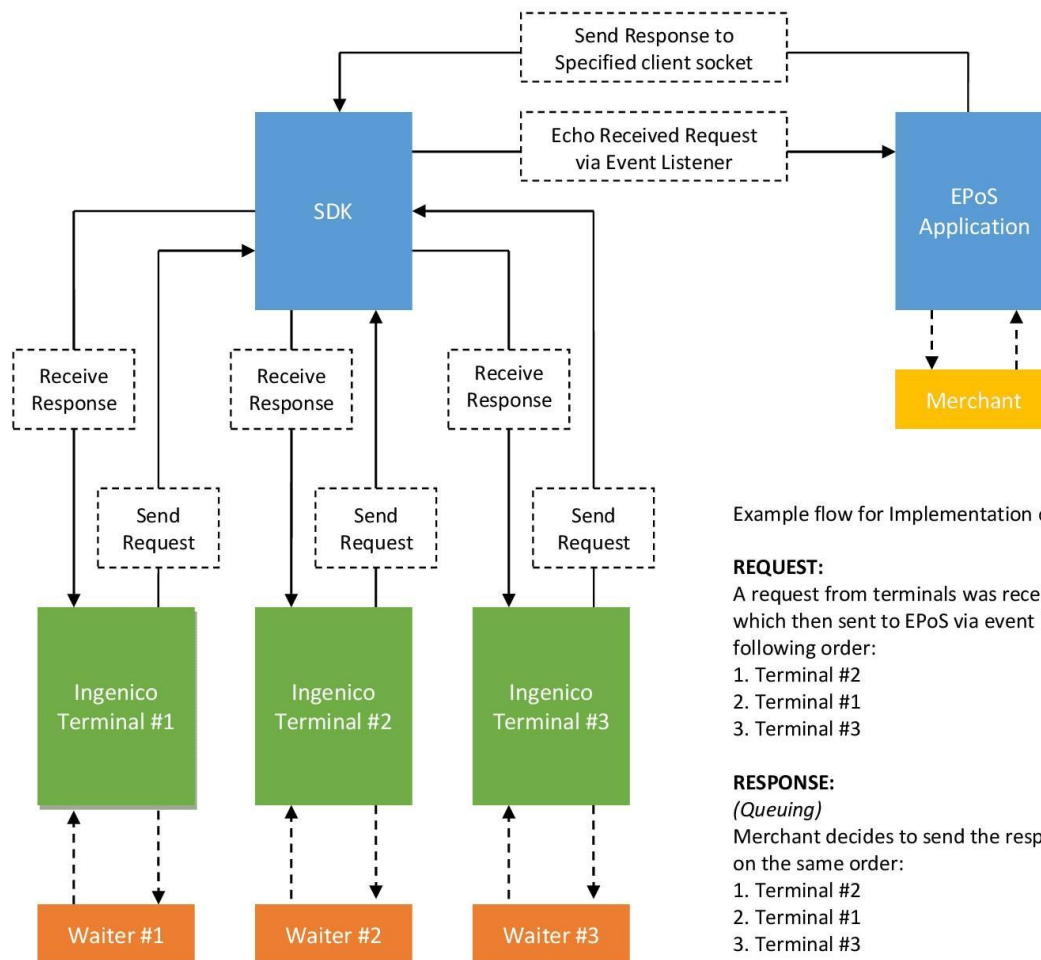


Figure 3: Multiple Device Communication Flow

3) Request Messages from EPOS to the Terminal

3.1 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 = (_device.sale(new BigDecimal("12.5"))
                .withReferenceNumber(1)
                .withPaymentMode(PaymentMode.APPLICATION)
                .withCurrencyCode("826")
                .execute());
} catch (ApiException e){
    e.printStackTrace();
}
```

Figure 3: 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 3. 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. Error classes such as *ApiException*, *BuilderException*, *ConfigurationException* etc. are included in the message which give an idea of the type of error that is thrown along with a more descriptive message.

3.2 Timeout Handling

A timeout timer is always initiated upon successful connection. An error will be thrown in the catch block (Refer to Figure 3) 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 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 4 (right), so the SDK developers opted for the current approach as illustrated in Figure 4 (left).

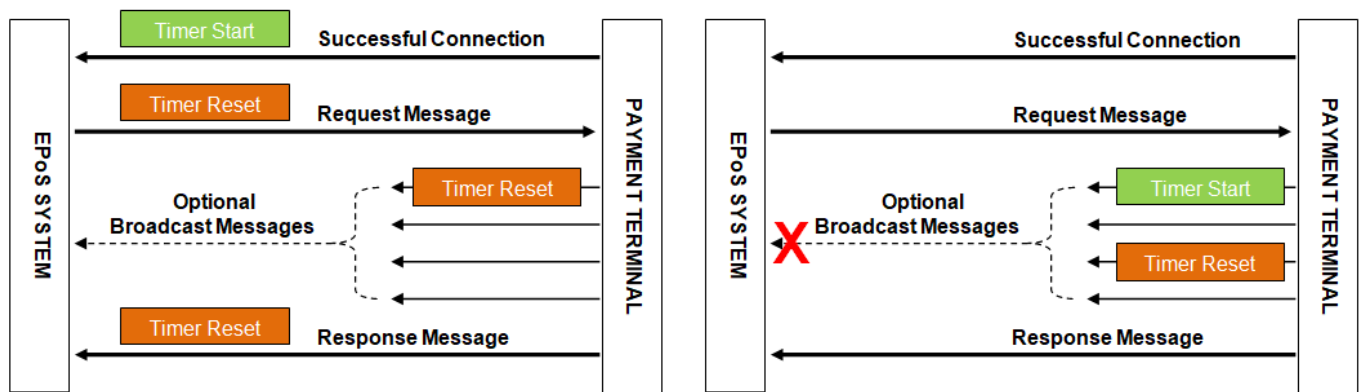


Figure 4: Timeout is initiated upon successful connection (left). Timeout is initiated only when the SDK receives a response from the terminal (right).

This approach (Figure 4, 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 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's 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.

3.2.1 Timeout Error Messages

Timeout is returned as an error message which can be printed using the exception object in Figure 3. Error messages for the following connection modes are listed below:

TCP - "Socket Error: Read timeout"

Serial - "Terminal did not respond within timeout"

4) 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.

4.1 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.getReferenceNumber();
String status = response.getStatus();
BigDecimal amount = response.getTransactionAmount();
PaymentMode paymentMode = response.getPaymentMode();
String currencyCode = response.getCurrencyCode();
String privateData = response.getPrivateData();
```

4.1.1 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 transactional amount is broken down). (Refer to Ingenico Documentation section 7.2)

```
String authCode = response.getAuthorizationCode();
BigDecimal cashBackAmount = response.getCashBackAmount();
BigDecimal gratuityAmount = response.getTipAmount();
BigDecimal finalAmount = response.getFinalTransactionAmount();
BigDecimal availableAmount = response.getBalanceAmount();
PaymentMethod paymentMethod = response.getPaymentMethod();
TransactionSubTypes txnSubType = response.getTransactionSubType();
BigDecimal splitSaleAmount = response.getSplitSaleAmount();
DynamicCurrencyStatus dccStatus = response.getDynamicCurrencyCodeStatus();
BigDecimal dccAmount = response.getDynamicCurrencyCodeAmount();
String dccCurrency = response.getDynamicCurrencyCode();
```

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*, *PaymentMethod*, *TransactionSubTypes* and *DynamicCurrencyStatus*.

4.2 IngenicoTerminalReportResponse

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

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

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

4.3 IngenicoTerminalReceiptResponse

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

- 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();
```

4.4 TerminalStateResponse

This response type is used specifically for the command State under the Terminal Management module. Data can be accessed from the following properties/methods:

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

Additional methods:

```
String terminalStatus = response.getTerminalStatus();
String salesMode = response.getSalesMode();
BigDecimal terminalCapabilities = response.getTerminalCapabilities();
String addlTerminalCapabilities = response.getAdditionalTerminalCapabilities();
String appVersion = response.getAppVersion();
String handsetNumber = response.getHandsetNumber();
String terminalID = response.getTerminalId();
```

4.5 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.getSerialNumber();
```

4.6 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.setOnBroadcastMessageReceived(new IBroadcastMessageInterface() {
    public void broadcastReceived(String code, String message) {
        Display.getDefault().asyncExec(new Runnable() {
            @Override
            public void run() {
                print(String.format("%s | %s \n", code, message));
            }
        });
    }
});
```

The above can be declared as soon as you have initialized the device. The event `setOnBroadcastMessageReceived` 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.

5) 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.

5.1 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 an *IngenicoTerminalResponse* (see appendix 7.6).

5.1.1 Sale

5.1.1.1 Request

```
IngenicoTerminalResponse response = new IngenicoTerminalResponse()  
response = (IngenicoTerminalResponse)_device.sale(new BigDecimal("100"))  
                .withReferenceNumber(1)  
                .withPaymentMode(PaymentMode.APPLICATION)  
                .withCashback(new BigDecimal("12.5"))  
                .withCurrencyCode("826")  
                .withTicket(true)  
                .execute();
```

Builder Type: **TerminalAuthBuilder**

Additional Methods:

Required
withReferenceNumber
withCurrencyCode

Optional
withCashBack
withPaymentMode (Default = PaymentMode.APPLICATION)
withTableNumber
withTicket

5.1.1.2 Response

Response Type: **IngenicoTerminalResponse**

Properties/methods:

With values	
getReferenceNumber	getCurrencyCode
getStatus	getPrivateData
getTransactionAmount	getPaymentMethod
getPaymentMode	getBalanceAmount
getAuthorizationCode	getFinalTransactionAmount

Conditional values	Condition
getTipAmount	Will have a value if gratuity amount is entered on the terminal
getCashbackAmount	Will have a value if request has withCashBack method
getReceipt	Will have a value if AutoTicket is enabled. Note: If TICKET is unsuccessful, Receipt will be empty.

5.1.2 Refund

5.1.2.1 Request

```
IngenicoTerminalResponse response = new IngenicoTerminalResponse()  
response = (IngenicoTerminalResponse)_device.refund(new  
BigDecimal("100"))  
  
                .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

5.1.2.2 Response

Response Type: **IngenicoTerminalResponse**

Properties/methods:

With values	
getReferenceNumber	getCurrencyCode
getStatus	getPrivateData
getTransactionAmount	getPaymentMethod
getPaymentMode	getBalanceAmount
getAuthorizationCode	getFinalTransactionAmount

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

5.1.3 Hotel Pre-Authorization

5.1.3.1 Request

```
IngenicoTerminalResponse response = new IngenicoTerminalResponse ()
response = (IngenicoTerminalResponse)_device.authorize(new BigDecimal ("100"))
    .withReferenceNumber (1)
    .withTableNumber ("1WE3464Y")
    .withCurrencyCode ("826")
    .withTicket (true)
    .execute ();
```

Builder Type: **TerminalAuthBuilder**

Additional Methods:

Required
withReferenceNumber
withCurrencyCode

Optional
withTableNumber
withTicket

5.1.3.2 Response

Response Type: **IngenicoTerminalResponse**

Properties/methods:

With values	
getReferenceNumber	getCurrencyCode
getStatus	getPrivateData
getTransactionAmount	getPaymentMethod
getPaymentMode	getBalanceAmount
getAuthorizationCode	getFinalTransactionAmount

Conditional values	Condition
getTipAmount	Will have a value if gratuity amount is entered on the terminal
getCashbackAmount	Will have a value if fallback is triggered. Cashback is entered on the terminal
getReceipt	Will have a value if AutoTicket is enabled. Note: If TICKET is unsuccessful, Receipt will be empty.

5.1.4 Hotel Completion

5.1.4.1 Request

```
IngenicoTerminalResponse response = new IngenicoTerminalResponse ()
response = (IngenicoTerminalResponse)_device.capture(new BigDecimal("100"))
```

```

.withReferenceNumber(1)
.withAuthCode("123456")
.withCurrencyCode("826")
.withTicket(true)
.execute();

```

Builder Type: **TerminalManageBuilder**

Additional Methods:

Required
withReferenceNumber
withCurrencyCode

Optional
withAuthCode
withTicket

5.1.4.2 [Response](#)

Response Type: **IngenicoTerminalResponse**

Properties/methods:

With values	
getReferenceNumber	getCurrencyCode
getStatus	getPrivateData
getTransactionAmount	getPaymentMethod
getPaymentMode	getBalanceAmount
getAuthorizationCode	getFinalTransactionAmount

Conditional values	Condition
getTipAmount	Will have a value if gratuity amount is entered on the terminal

getCashbackAmount	Will have a value if fallback is triggered. Cashback is entered on the terminal
getReceipt	Will have a value if AutoTicket is enabled. Note: If TICKET is unsuccessful, Receipt will be empty.

5.1.5 Account Verification

5.1.5.1 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

5.1.5.2 Response

Response Type: **IngenicoTerminalResponse**

Properties/methods:

With values	
getReferenceNumber	getCurrencyCode
getStatus	getPrivateData
getTransactionAmount	getPaymentMethod
getPaymentMode	

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

5.1.6 Tax Free Cash Refund

5.1.6.1 Request

```
IngenicoTerminalResponse response = new IngenicoTerminalResponse()
response = (IngenicoTerminalResponse)_device.refund(new
BigDecimal("100"))
    .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)

5.1.6.2 Response

Response Type: **IngenicoTerminalResponse**

Properties/methods:

With values	
getReferenceNumber	getCurrencyCode
getStatus	getPrivateData
getTransactionAmount	getPaymentMethod
getPaymentMode	

5.1.7 Tax Free Credit Card Refund

5.1.7.1 Request

```
IngenicoTerminalResponse response = new IngenicoTerminalResponse()
response = (IngenicoTerminalResponse)_device.refund(new
BigDecimal("100"))
    .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)

5.1.7.2 Response

Response Type: **IngenicoTerminalResponse**

Properties/methods:

With values

getReferenceNumber	getCurrencyCode
getStatus	getPrivateData
getTransactionAmount	getPaymentMethod
getPaymentMode	

5.1.8 External Referral Management

5.1.8.1 Request

```
IngenicoTerminalResponse response = new IngenicoTerminalResponse()
response = (IngenicoTerminalResponse)_device.referralConfirmation
    .withReferenceNumber(1)
    .withAuthCode("12345")
    .withCurrencyCode("826")
    .execute();
```

Builder Type: **TerminalManageBuilder**

Additional Methods:

Required
withReferenceNumber
withCurrencyCode

Optional
withAuthCode

5.1.8.2 Response

Response Type: IngenicoTerminalResponse

Properties/methods:

With values

getReferenceNumber	getCurrencyCode
getStatus	getPrivateData
getTransactionAmount	getPaymentMethod
getPaymentMode	getFinalTransactionAmount
getAuthorizationCode	getBalanceAmount

5.2 Transaction Management

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

5.2.1 Cancel

5.2.1.1 Request

```
IngenicoTerminalResponse response = new IngenicoTerminalResponse ()
response = (IngenicoTerminalResponse)_device.cancel();
```

Does not utilize a builder. Executed upon method call.

5.2.1.2 Response

Response Type: **IngenicoTerminalResponse**

Properties/methods:

With values	
getReferenceNumber	getCurrencyCode
getStatus	getPrivateData
getTransactionAmount	getPaymentMode

5.2.2 Duplicate

5.2.2.1 Request

```
IngenicoTerminalResponse response = new IngenicoTerminalResponse ()
response = (IngenicoTerminalResponse)_device.duplicate(true);
```

Does not utilize a builder. Executed upon method call.

Additional Methods: Input a boolean parameter to enable/disable automatic Ticket

5.2.2.2 Response

Response Type: **IngenicoTerminalResponse**

Properties/methods:

With values	
getReferenceNumber	getCurrencyCode
getStatus	getPrivateData
getTransactionAmount	getPaymentMode

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

5.2.3 Reverse

5.2.3.1 Request

```
IngenicoTerminalResponse response = new IngenicoTerminalResponse()  
response = (IngenicoTerminalResponse)_device.reverse(new BigDecimal("100"))  
                .withReferenceNumber(1)  
                .withTicket(true)  
                .execute();
```

Builder Type: **TerminalManageBuilder**

Additional Methods:

Required
withReferenceNumber

Optional
withTicket

5.2.3.2 Response

Response Type: **IngenicoTerminalResponse**

Properties/methods:

With values	
getReferenceNumber	getCurrencyCode
getStatus	getPrivateData
getTransactionAmount	getPaymentMode

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

5.2.4 Reverse with Transaction ID

5.2.4.1 Request

```
IngenicoTerminalResponse response = new IngenicoTerminalResponse()  
response = (IngenicoTerminalResponse)_device.reverse(new BigDecimal("100"))  
                .withReferenceNumber(1)  
                .withTransactionID("8153")  
                .withTicket(true)  
                .execute();
```

Builder Type: **TerminalManageBuilder**

Additional Methods:

Required
withReferenceNumber
withTransactionID

Optional
withTicket

5.2.4.2 Response

Response Type: **IngenicoTerminalResponse**

Properties/methods:

With values	
getReferenceNumber	getCurrencyCode
getStatus	getPrivateData
getTransactionAmount	getPaymentMode

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

5.3 Report Management

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

5.3.1 EOD

5.3.1.1 Request

```
IngenicoTerminalReportResponse response = new IngenicoTerminalReportResponse ()  
response = (IngenicoTerminalReportResponse)_device.getReport (ReportTypes.EOD)  
                .execute ();
```

Builder Type: **TerminalReportBuilder**

Additional Methods: None

5.3.1.2 Response

Response Type: **IngenicoTerminalReportResponse**

Properties/methods:

With values	
getReferenceNumber	getCurrencyCode
getStatus	getPrivateData
getTransactionAmount	getPaymentMode

5.3.2 Banking

5.3.2.1 Request

```
IngenicoTerminalReportResponse response = new IngenicoTerminalReportResponse()  
response = (IngenicoTerminalReportResponse)_device.getReport(ReportTypes.BANKING)  
                .execute();
```

Builder Type: **TerminalReportBuilder**

Additional Methods: None

5.3.2.2 Response

Response Type: **IngenicoTerminalReportResponse**

Properties/methods:

With values	
getReferenceNumber	getCurrencyCode
getStatus	getPrivateData
getTransactionAmount	getPaymentMode

5.3.3 XBAL

5.3.3.1 Request

```
IngenicoTerminalReportResponse response = new IngenicoTerminalReportResponse()  
response = (IngenicoTerminalReportResponse)_device.getReport(ReportTypes.XBAL)  
                .execute();
```

Builder Type: **TerminalReportBuilder**

Additional Methods: None

5.3.3.2 Response

Response Type: **IngenicoTerminalReportResponse**

Properties/methods:

With values	
getReferenceNumber	getCurrencyCode
getStatus	getPrivateData
getTransactionAmount	getPaymentMode

5.3.4 ZBAL

5.3.4.1 Request

```
IngenicoTerminalReportResponse response = new IngenicoTerminalReportResponse()  
response = (IngenicoTerminalReportResponse)_device.getReport(ReportTypes.ZBAL)  
                .execute();
```

Builder Type: **TerminalReportBuilder**

Additional Methods: None

5.3.4.2 Response

Response Type: **IngenicoTerminalReportResponse**

Properties/methods:

With values	
getReferenceNumber	getCurrencyCode
getStatus	getPrivateData
getTransactionAmount	getPaymentMode

5.4 XML Management

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

5.4.1 Report

5.4.1.1 Request

```
IngenicoTerminalReceiptResponse response = new IngenicoTerminalReceiptResponse()  
response = (IngenicoTerminalReceiptResponse)_device.getLastReceipt(ReceiptType.RE
```

```
PORT).execute();
```

Builder Type: **TerminalReportBuilder**

Additional Methods: None

5.4.1.2 Response

Response Type: **IngenicoTerminalReceiptResponse**

Properties/methods:

With values
toString

5.4.2 Ticket

5.4.2.1 Request

```
IngenicoTerminalReceiptResponse response = new IngenicoTerminalReceiptResponse()  
response = (IngenicoTerminalReceiptResponse)_device.getLastReceipt(ReceiptType.TI  
CKET).execute();
```

Builder Type: **TerminalReportBuilder**

Additional Methods: None

5.4.2.2 Response

Response Type: **IngenicoTerminalReceiptResponse**

Properties/methods:

With values
toString

5.4.3 SplitR

5.4.3.1 Request

```
IngenicoTerminalReceiptResponse response = new IngenicoTerminalReceiptResponse()  
response = (IngenicoTerminalReceiptResponse)_device.getLastReceipt(ReceiptType.SP  
LITR).execute();
```

Builder Type: **TerminalReportBuilder**

Additional Methods: None

5.4.3.2 Response

Response Type: **IngenicoTerminalReceiptResponse**

Properties/methods:

With values
toString

5.4.4 Tax Free

5.4.4.1 Request

```
IngenicoTerminalReceiptResponse response = new IngenicoTerminalReceiptResponse()  
response = (IngenicoTerminalReceiptResponse)_device.getLastReceipt(ReceiptType.TAX  
FREE).execute();
```

Builder Type: **TerminalReportBuilder**

Additional Methods: None

5.4.4.2 Response

Response Type: **IngenicoTerminalReceiptResponse**

Properties/methods:

With values
toString

5.5 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, *TerminalStateResponse* and *POSIdentifierResponse* (see sections 4.4 and 4.5). Refer to Ingenico Documentation section 5.4 for more details.

5.5.1 Call TMS

5.5.1.1 Request

```
IngenicoTerminalResponse response = new IngenicoTerminalResponse()  
response = (IngenicoTerminalResponse)_device.getTerminalConfiguration();
```

5.5.1.2 Response

Response Type: **IngenicoTerminalResponse**

Properties/methods:

With values	
getReferenceNumber	getCurrencyCode
getStatus	getPrivateData
getTransactionAmount	getPaymentMode

5.5.2 Logon

5.5.2.1 Request

```
IngenicoTerminalResponse response = new IngenicoTerminalResponse()  
response = (IngenicoTerminalResponse)_device.testConnection(true);
```

Additional Methods: Input a boolean parameter to enable/disable automatic Ticket

5.5.2.2 Response

Response Type: **IngenicoTerminalResponse**

Properties/methods:

With values	
getReferenceNumber	getCurrencyCode
getStatus	getPrivateData
getTransactionAmount	getPaymentMode

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

5.5.3 Reset

5.5.3.1 Request

```
IngenicoTerminalResponse response = new IngenicoTerminalResponse()  
response = (IngenicoTerminalResponse)_device.reboot();
```

5.5.3.2 Response

Response Type: **IngenicoTerminalResponse**

Properties/methods:

With values	
getReferenceNumber	getCurrencyCode
getStatus	getPrivateData
getTransactionAmount	getPaymentMode

5.5.4 State

5.5.4.1 Request

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

5.2.1.1 Response

Response Type: **IngenicoStateResponse**

Properties/methods:

With values	
getReferenceNumber	getPrivateData
getStatus	getPaymentMode
getTransactionAmount	getTerminalStatus
getSalesMode	getTerminalCapabilities
getAdditionalTerminalCapabilities	getAppVersion
getHandsetNumber	getTerminalId

5.5.5 PID

5.5.5.1 Request

```
IngenicoTerminalResponse response = new IngenicoTerminalResponse()  
response = (IngenicoTerminalResponse)_device.initialize();
```

5.5.5.2 Response

Response Type: **POSIentifierResponse**

Properties/methods:

With values	
getReferenceNumber	getCurrencyCode
getStatus	getPrivateData
getTransactionAmount	getPaymentMode
getSerialNumber	

6) 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.

6.1 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 5.

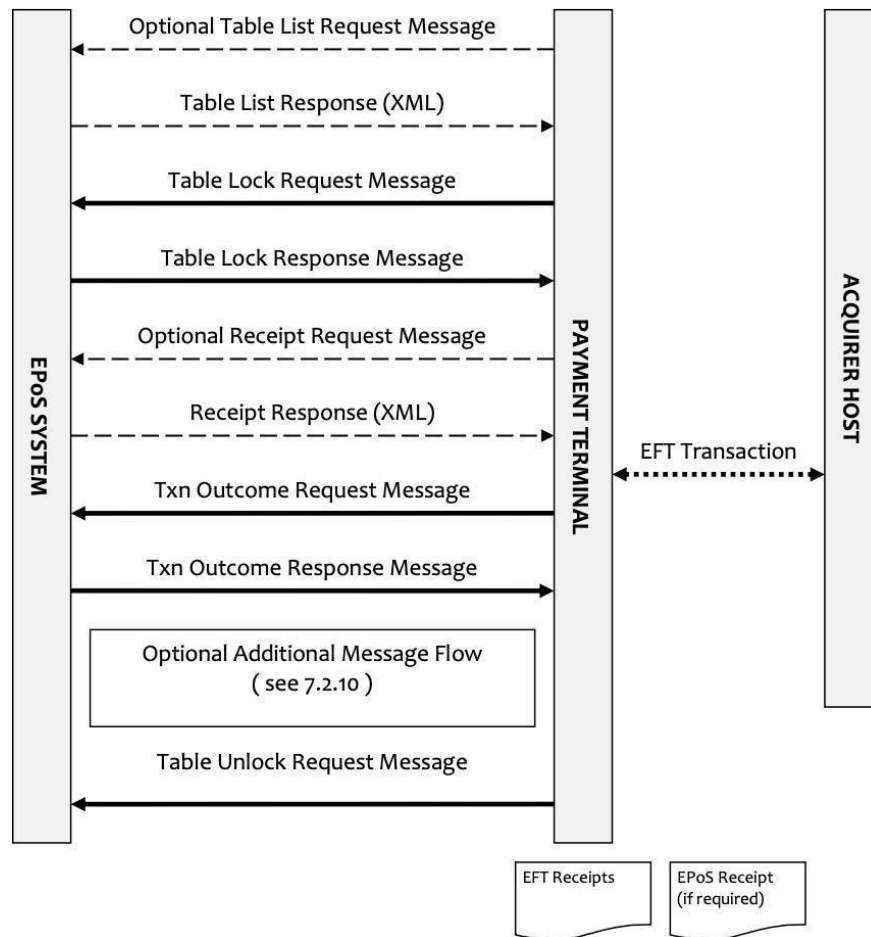


Figure 5: Referenced from Ingenico Documentation (Section 7.2.3, Page 68)

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.

6.2 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.setOnPayAtTableRequest(new IPayAtTableRequestInterface() {  
    public void onPayAtTableRequest(final PATRequest 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.

6.2.1 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:

```
String waiterID = request.getWaiterId();  
String tableID = request.getTableId();  
String terminalID =  
request.getTerminalId();  
String terminalCurrency = request.getTerminalCurrency();  
String xmlData = request.getXMLData();  
PATRequestType requestType = request.getRequestType();  
TransactionOutcome txnOutcome = request.getTransactionOutcome();
```

6.2.1.1 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.getRequestType();
if (requestType == PATRequestType.TABLE_LOCK) {
    //Send Table Lock response
}
```

6.2.1.2 TransactionOutcome

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

```
String status = txnOutcome.getTransactionStatus().name();
String amount = txnOutcome.getAmount();
String currencyCode = txnOutcome.getCurrencyCode();
String privateData = txnOutcome.getPrivateData();
DataResponse repFields = txnOutcome.getRepFields();
```

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

```
String authCode = response.getAuthorizationCode();
BigDecimal finalAmount = repFields.getFinalTransactionAmount();
PaymentMethod paymentMethod = response.getPaymentMethod();
(...)
```

6.2.1.3 XMLData

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

```
String xmlData = request.getXMLData();
```

6.3 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()`. This is further built upon by *TerminalAuthBuilder* methods similar to how request messages are constructed in standard/normal mode.

6.3.1 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.TABLE_LOCK) {
    try{
        _device.payAtTableResponse()
            .withPATResponseType(PATResponseType.CONF_OK)
            .withAmount(new BigDecimal("100"))
            .withPATPaymentMode(PATPaymentMode.NO_ADDITIONAL_MSG)
            .withCurrencyCode("826")
            .execute();
    } catch (ApiException e) {
        e.printStackTrace();
    }
}
```

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_MSG*). Refer to section 7.3 for more info.

6.3.2 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.TABLE_LIST) {
    try{
        _device.payAtTableResponse()
            .withXML("C:\\Users\\john.doe\\Desktop\\sample.xml")
            .execute();
    } catch (ApiException e) {
        e.printStackTrace();
    }
}
```

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).

6.4 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.

6.4.1 Table List

6.4.1.1 Request

Request Type: **TABLE_LIST**

Properties/methods:

With values
getRequestType

Conditional values	Condition
getWaiterId	Will have a value if Waiter ID is entered on the terminal

6.4.1.2 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:

- "if the Waiter ID is "00" (or empty) then it is expected that a list of all open tables 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).
- We recommend using *less than 10 characters* for tableID 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 tableID names with 10 characters or more due to issues with length formatting on the current API version.

6.4.2 Table Lock

6.4.2.1 Request

Request Type: **TABLE_LOCK**

Properties/methods:

With values
getRequestType
getTableId

Conditional values	Condition
getWaiterId	Will have a value if Waiter ID was previously entered on the terminal and sent along with the Table List request.
getTerminalId	Will have a value if enabled in terminal configuration. Refer to Ingenico Documentation section 7.2.1.
getTerminalCurrency	Will have a value if enabled in terminal configuration. Refer to Ingenico Documentation section 7.2.1.

6.4.2.2 Response

Response Type: **Confirmation Response**

Additional Methods:

Required
withPATResponseType
withAmount
withPATPaymentMode
withCurrencyCode

6.4.3 Receipt Request

6.4.3.1 Request

Request Type: **RECEIPT_MESSAGE**

Properties/Methods:

With values
getRequestType

6.4.3.2 Response

Response Type: **XML Response**

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

6.4.4 Transaction Outcome

6.4.4.1 Request

Request Type: **TRANSACTION_OUTCOME**

Properties/methods:

With values
getRequestType
getTransactionOutcome (Refer to section 6.2.1.2)

6.4.4.2 Response

Response Type: **Confirmation Response**

Additional Methods:

Required
withPATResponseType
withAmount
withPATPaymentMode
withCurrencyCode

If an additional message option is enabled on the terminal, you can set

USE_ADDITIONAL_MESSAGE on withPATPaymentMode to signal the terminal to send additional data to the EPOS.

6.4.5 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.

6.4.5.1 Request

Request Type: **ADDITIONAL_MESSAGE**

Properties/methods:

With values
getRequestType
getXmlData

6.4.5.2 Response

Response Type: **XML Response**

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

6.4.6 Split Sale Report Data

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

6.4.6.1 Request

Request Type: **SPLITSALE_REPORT**

Properties/methods:

With values
getRequestType
getXmlData

6.4.6.2 Response

Response Type: **Confirmation Response**

Additional Methods:

Required
withPATResponseType
withAmount
withPATPaymentMode
withCurrencyCode

6.4.7 Final Report Data

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

6.4.8 Ticket

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

6.4.8.1 Request

Request Type: **TICKET**

Properties/methods:

With values
getRequestType
getXmlData

6.4.8.2 Response

Response Type: **Confirmation Response**

Additional Methods:

Required
withPATResponseType
withAmount
withPATPaymentMode
withCurrencyCode

6.4.9 Transfer Data

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

6.4.9.1 Request

Request Type: **TRANSFER_DATA**

Properties/methods:

With values
getRequestType
getXmlData

6.4.9.2 Response

Response Type: **Confirmation Response**

Additional Methods:

Required
withPATResponseType
withAmount
withPATPaymentMode
withCurrencyCode

6.4.10 EOD Report

Sent after a Transfer Data request is received.

6.4.10.1 Request

Request Type: **EOD_REPORT**

Properties/methods:

With values
getRequestType
getXmlData

6.4.10.2 Response

Response Type: **Confirmation Response**

Additional Methods:

Required
withPATResponseType
withAmount
withPATPaymentMode
withCurrencyCode

6.4.11 Table Unlock

6.4.10.1 Request

Request Type: **TABLE_UNLOCK**

Properties/methods:

With values
getRequestType
getTableId

6.4.10.2 Response

Response Type: **No response**

7) Appendix

7.1 ConnectionConfig

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

Method Name	Description
setBaudRate	Connection Baud Rate
setConnectionMode	Connection Mode (See 7.8 for enum values)
setDataBits	Connection Data Bits
setDeviceType	Type of Device to be used. (See 7.8 for enum values)
setIpAddress	IP Address of device. (TCP/IP Connection)
setParity	Connection Parity
setPort	Port of device.
setRequestIdProvider	Request Id Provider
setStopBits	Connection Stop Bits
setTimeout	Set a timeout (in milliseconds) for the connection.
ConnectionConfig	Method used to connect the current connection.
Validate	Used to validate connection.

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 (BigDecimal amount)	Payment Management	Perform Sale (Payment Type) transaction into terminal
refund (BigDecimal amount)		Perform Refund (Payment Type) transaction into terminal
capture(BigDecimal amount)		Perform Hotel Mode Completion (Payment Type) transaction into terminal
authorize (BigDecimal amount)		Perform Hotel Mode Pre-authorisation (Payment Type) transaction into terminal
verify()		Perform Account Verification (Payment Type) transaction into terminal
cancel ()	Transaction Management	The terminal will immediately cancel the current transaction.
duplicate ()		The terminal 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(ReportTypes 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
getTerminalConfiguration()	Terminal Management	The terminal immediately initiates a TMS Call to pick up the latest configuration for the terminal.
testConnection()		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(Socket socket)	Pay@Table	Command used for responding to terminal requests in Pay@Table mode. Socket is a mandatory parameter which is a reference to specify which device to send the response.

Events

Method Name	Description
setOnBroadcastMessageReceived(IBroadcastMessageInterface onBroadcastReceived)	Event listener for broadcast data sent from the terminal.
setOnPayAtTableRequest(IPayAtTableRequestInterface onPayAtTableRequest)	Event listener for terminal requests in Pay@Table mode

Event Parameters

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

7.2 TerminalAuthBuilder

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

Property Name	Description
withReferenceNumber(Integer value)	This command allows addition of Epos Number in request message.
withCashback(BigDecimal 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 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(BigDecimal amount)	Allows the addition of an amount to a confirmation response. (Pay@Table mode)
withPATResponseType(PATResponseType type)	Used to specify a positive (CONF_OK) or negative (CONF_NOK) confirmation. (Pay@Table mode)

withPATPaymentMode(PATPaymentMode mode)	Tells the terminal whether additional message is being requested (USE_ADDITIONAL_MSG) or not (NO_ADDITIONAL_MSG). (Pay@Table mode)
withTicket(Boolean 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)

7.3 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(Boolean 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

7.4 TerminalReportBuilder

Property Name	Description
getLastReceipt(ReceiptType type)	Automatically executed upon method call in IDeviceInterface (section 7.2).
getReport(ReportTypes 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
-----------	---

7.5 IngenicoTerminalResponse

Base response data

Method Name	Data Type	Description
getReferenceNumber	String	Terminal / EPOS reference number
getStatus	String	Status / Response Code of Transaction. Refer to section 7.8 for return values.
getTransactionAmount	Big Decimal	Amount of Transaction
getCurrencyCode	String	The currency used for the transaction.
getPaymentMode	String	The mode of Payment. Refer to section 7.8 for return values.
getPrivateData	String	Returns various private data sent by the EPOS/ Terminal.
getReceipt	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
getReceiptErrorMessage	String	Contains error message if Exception occur in AutoTicket

REP Field data

Method Name	Data Type	Description
getAuthorizationCode	String	Authorization Code came from the terminal. (Can be used for Completion)
getCashBackAmount	Big Decimal	Cashback Amount from request message.
getTipAmount	Big Decimal	Amount of Gratuity
getFinalTransactionAmount	Big Decimal	Total amount including Cashback and Gratuity
getBalanceAmount	Big Decimal	The available amount of transaction.
getDynamicCurrencyCode	String	The currency of Dynamic currency conversion (DCC).
getDynamicCurrencyCodeStatus	enum	The status of dynamic currency conversion (DCC). Refer to section 7.8 for enum values.

getDynamicCurrencyCodeAmount	Big Decimal	The amount of dynamic currency conversion (DCC).
getPaymentMethod	String	The method of Payment used. Refer to section 7.8 for return values.
getTransactionSubType	enum	The sub-type of transaction. The terminal response the following: Split Sale Transaction, DCC Transaction, or Referral Result. Refer to section 7.8 for enum values.
getSplitSaleAmount	Big Decimal	The amount of split sale.

7.6 IngenicoTerminalReceiptResponse

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

7.7 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)
MAILORDER (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
REPORT

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_MSG
NO_ADDITIONAL_MSG

REFERENCES

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