Mobile SDK Specification (Android)

Merchant Integration Guide

Version 2.1.0

Important: Please note that this document is strictly private and confidential and is not intended for public circulation.

Last updated 3 March 17

GHL Systems Berhad (293040-D)
C-G-15, Block C, Jalan Dataran SD1, Dataran SD, PJU 9,
Bandar Sri Damansara 52200 Kuala Lumpur, Malaysia
Tel +603 6286 3388 Fax +603 6280 2999 www.ghl.com

Innovation Centre
L5-E-7B, Enterprise 4, Technology Park Malaysia.
Bukit Jalil, 57000 Kuala Lumpur, Malaysia.
Tel +603 8998 3600 Fax +603 8998 3301

e-pay (M) Sdn. Bhd. (494207-U)
16-18, Jalan PJS 11/28 A, Bandar Sunway,
47500 Subang Jaya, Selangor, Malaysia.
Tel +603 5632 2488 Fax +603 5636 6966 www.e-pay.com.my

Contents

1. Introduction

Mobile Software Development Kit (SDK) allows integration with Merchant mobile application that would like to accept online payment by 3D and non-3D credit card, direct debit and e-Wallet payments.

This Merchant Integration Guide provides merchants with the necessary technical information to integrate their applications (Merchant Systems) with Payment Gateway.

The manual contains message format required between Payment Gateway and Merchant System for various payment transaction types, namely Payment, Query and Reversal. It is intended as a technical guide for merchant developers and system integrators who are responsible for designing or programming the respective applications to integrate with Payment Gateway.

1.1. Prerequisite

All merchants who would like to integrate with Payment Gateway must obtain a valid payment account from eGHL. Upon payment account generated, eGHL will provide merchant a Service ID and Merchant Password.

1.2. System Requirements

- Ready for Android API 17 and up
- Android Studio

2. SDK Integration

This section explains details on eGHL SDK integration. SDK is a library that will be added to the Android Studio project.

The followings are the process of integration:

1) Configure Android Studio project (Adding the library).
2) Configure SDK
3) Merchant app supplies the necessary information to the SDK
4) SDK will handle the payment process.
5) After payment is finished get the result.

2.1. Configure Android Studio project

1. Copy the .aar file to your **libs** folder
2. Declare the **libs** folder as a dependency repository in your application Gradle script.

```
repositories{
    flatDir{
        dirs 'libs'
    }
}
```

3. In the dependency block, compile the .aar library and the Volley library.

```
dependencies {
    compile(name:'eghl-sdk-v2.1.0', ext:'aar')
    compile 'com.android.volley:volley:1.0.0'

}
```

Sample application level Gradle script:

```
apply plugin: 'com.android.application'

android {
...
}
repositories{
    flatDir{
        dirs 'libs'
    }
}

dependencies {
...
    compile(name:'eghl-sdk-v2.1.0', ext:'aar')
    compile 'com.android.volley:volley:1.0.0'
}
```

2.2. Using the SDK

1. Add two meta-data tags inside the Application tag in your manifest with the name of **com.epay.eghl.sdk.PASSWORD** and **com.epay.eghl.sdk.PAYMENT_GATEWAY.**

2. Supply the two meta-data with the right values that you've got from us.

```
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.eghl.demosdk">

<uses-permission android:name="android.permission.INTERNET"/>
    <application
        android:allowBackup="true"
        android:icon="@mipmap/ic_launcher"
        android:label="@string/app_name"
        android:supportsRtl="true"
        android:theme="@style/AppTheme">

        <meta-data android:name="com.epay.eghl.sdk.PASSWORD"
android:value="Replace with merchant password"/>
        <meta-data android:name="com.epay.eghl.sdk.PAYMENT_GATEWAY"
android:value="Replace with EGHL payment gateway"/>

    </application>

</manifest>
```

3. Initialize an instance of the EGHL class.

```
EGHL eghl = EGHL.getInstance();
```

2.2.1. Making a Payment

1. Create the necessary parameters to make a payment by using the **PaymentParams.Builder** class:

```
final String cnasit = eghl.generateId("CNASIT");
final PaymentParams.Builder params = new PaymentParams.Builder()

.setMerchantReturnUrl("https://test2pay.ghl.com/IPGSimulatorJeff/RespFrmGW.aspx
")
        .setServiceId("OM2")
        .setAmount("10.00")
        .setPaymentDesc("eGHL Payment testing")
        .setCustName("Jeff")
        .setCustEmail("jeff.phang@ghl.com")
        .setCustPhone("60123456789")
```

```
        .setMerchantName("GHL ePayment Testing")
        .setPaymentId(cnasit)
        .setOrderNumber(cnasit)
        .setCurrencyCode("MYR")
        .setLanguageCode("EN")
        .setPageTimeout("500")
        .setTransactionType("SALE")
        .setPaymentMethod("ANY");
```

2. Build the parameters by calling the **build()** method on the **PaymentParams.Builder** object. It will generate a **Bundle** object with the payment parameters that you've set.

3. Execute the payment by calling the **executePayment(Bundle params, Activity activity)** on the **EGHL** instance.
   The first parameter is the **Bundle** object from the previous step and the second parameter is the **Activity** that will handle the result of the payment.

```
Bundle paymentParams = params.build();
eghl.executePayment(paymentParams,MainActivity.this);
```

4. Override the **onActivityResult()** inside the **Activity** that you've passed-in as the second parameter of the **executePayment()** from the previous step. After the payment is finished, the result will be returned to this method and the request code will be **EGHL.REQUEST_PAYMENT.** You can handle the different responses from the server here. All the available constants for the result codes are at the **Payment Result Codes** section below.

```
@Override
protected void onActivityResult(int requestCode, int resultCode, Intent data) {
    if(requestCode == EGHL.REQUEST_PAYMENT){
        switch(resultCode){
            case EGHL.TRANSACTION_SUCCESS:
                Log.d(TAG, "onActivityResult: payment successful");
                String status = data.getStringExtra(EGHL.TXN_STATUS);
                String message = data.getStringExtra(EGHL.TXN_MESSAGE);
                String raw = data.getStringExtra(EGHL.TXN_RAW_RESPONSE);
            break;
            case EGHL.TRANSACTION_FAILED:
                Log.d(TAG, "onActivityResult: payment failure");
            break;
            default:
                Log.d(TAG, "onActivityResult: "+resultCode);
            break;
        }
    }

}
```

Payment Result Codes

Status Code	Constant Name	Description
0	EGHL.TRANSACTION_SUCCESS	Transaction successful (for transaction type SALE)
1	EGHL.TRANSACTION_FAILED	Transaction failed
2	EGHL.TRANSACTION_PENDING	Sale pending, retry Query
10	EGHL.TRANSACTION_REFUNDED	Transaction refunded
15	EGHL.TRANSACTION_AUTHORIZED	Transaction authorized (for transaction type AUTH)
16	EGHL.TRANSACTION_CAPTURED	Transaction captured
31	EGHL.TRANSACTION_REVERSAL_PENDING	Reversal pending, merchant system can retry Reversal if merchant system initiated the Reversal request or else merchant system can retry Query.
-1	EGHL.TRANSACTION_NOT_EXISTING	Transaction not exists/not found
9	EGHL.TRANSACTION_REVERSED	Transaction reversed
-2	EGHL.TRANSACTION_INTERNAL_ERROR	Internal system error
-999	EGHL.TRANSACTION_CANCELLED	Transaction cancelled by the user
-888	EGHL.TRANSACTION_NO_STATUS	Can't determine the status, try to query again.

You can also get additional information about the response from the **Intent** object that was passed-in, like in the example above.
The available keys for the extras are the following constants**:**

Constant Name	Description
EGHL.TXN_STATUS	Key constant that can be used in getting the transaction status in a Bundle object returned from the payment.
EGHL.TXN_MESSAGE	Key constant that can be used in getting the transaction message in a Bundle object returned from the payment.
EGHL.RAW_RESPONSE	Key constant that can be used in getting the raw string response from the server.

2.2.2. Making a Masterpass Payment

As described in the eGHL_API document, there are two ways to pay with Masterpass. One needs pairing and one doesn't. Both flows are facilitated by the SDK.
You have to send the first request to the payment gateway. You can do this by using the **MasterpassParams.Builder** class to build the parameters needed. Build it and pass it as the **second** parameter of **executeMasterpassRequest()** method of **EGHL** instance.

```
MasterpassParams.Builder firstRequest = new MasterpassParams.Builder();
firstRequest.setCurrencyCode(currencyCode);
firstRequest.setAmount(amount);
firstRequest.setToken(token);
firstRequest.setPaymentDesc(paymentDesc);
firstRequest.setServiceID(serviceId);
```

TokenType should be "MPE" and the **Token** should be your user's login ID.

As the third parameter of the **executeMasterpassRequest()** you need to supply an object that would implement the **MasterpassCallback** interface.

```
eghl.executeMasterpassRequest(MainActivity.this,firstRequest.build(), new
MasterpassCallback() {
    @Override
    public void onResponse(final String response) {
            // Handle pairing or express
        progress.dismiss();

if(response.contains(Params.MASTERPASS_REQ_TOKEN)&&response.contains(Params.MASTERPASS_PA
IRING_TOKEN)){
            // Needs pairing
            proceedPairing(response);
        }else if (response.contains(Params.MASTERPASS_PRE_CHECKOUT_ID)){
            // Can proceed to checkout via masterpass express
            proceedExpress(response);
        }
    }

    @Override
    public void onError(Exception e) {
        e.printStackTrace();
        Log.e(TAG, "http error", e);
        progress.dismiss();
    }
});
```

You will receive the server response on its **onResponse()** method. You have to handle it. The response could be for **Express Checkout** or **Pairing with Checkout**. Please do note that **Token** and **TokenType** will be a required field after this process. It must be included in the 2nd request to the payment gateway -- that will be explained in the topics below.

Masterpass with Lightbox for Pairing

Once you received a **ReqToken** and **PairingToken** from the first request above, you have to launch the Masterpass Lightbox and passing-in the two values received and the other parameters. The **LightboxParams.Builder** class could help you with this:

```
private void proceedPairing(String response) {
    Uri uri = Uri.parse("?"+response);
    String pairingToken = uri.getQueryParameter(Params.MASTERPASS_PAIRING_TOKEN);
    String reqToken = uri.getQueryParameter(Params.MASTERPASS_REQ_TOKEN);

    LightboxParams.Builder params = new LightboxParams.Builder()
            .setReqToken(reqToken)
            .setPairingToken(pairingToken)
            .setLightBoxCallbackURL("https://radiant-reaches-
88215.herokuapp.com/commands/callback")
            .setMPEMerchantCheckoutID("a32d8440202b408dbdcc3ca8763d4625")

.setLightboxJS("https://sandbox.static.masterpass.com/dyn/js/switch/integration/MasterPas
s.client.js");


    // Result of this pairing will be in the onActivityResult. With request code of
EGHL.REQUEST_PAIRING.
    eghl.executePairing(params.build(),MainActivity.this);
}
```

In the above example notice that it sets the LightBoxJS to:
https://sandbox.static.masterpass.com/dyn/js/switch/integration/MasterPass.client.js

That URL for Masterpass JS script indicates that intends to connect to the Masterpass sandbox environment.

If you want to be in production mode set the value to:
https://static.masterpass.com/dyn/js/switch/integration/MasterPass.client.js

Ref: https://developer.mastercard.com/documentation/masterpass-merchant-integration

The 3rd parameter of the **executePairing()** is an activity that would handle the results of the Lightbox flow. It would return to the **onActivityResult()** method of the activity with the request code of: **EGHL.REQUEST_PAIRING**. The values returned from the LightBox are inside the Intent object of the method (3rd parameter). The expected keys are below. Once it's a success you can proceed to make a normal payment (2nd request to the payment gateway) but you have to include additional parameters that you've got from the Lightbox.

```
@Override
protected void onActivityResult(int requestCode, int resultCode, Intent data) {
if(requestCode == EGHL.REQUEST_PAIRING){
    if(resultCode == EGHL.TRANSACTION_MASTERPASS_FINISHED){

        String reqVerifier = data.getStringExtra("oauth_verifier")!= null ?
data.getStringExtra("oauth_verifier"):"";
        String pairingVerifier = data.getStringExtra("pairing_verifier")!= null ?
data.getStringExtra("pairing_verifier"):"";
```

```
        String checkoutURL = data.getStringExtra("checkout_resource_url")!=null ?
data.getStringExtra("checkout_resource_url"):"";
        String pairingToken = data.getStringExtra("pairing_token")!=null ?
data.getStringExtra("pairing_token"):"";
        String reqToken = data.getStringExtra("oauth_token")!=null ?
data.getStringExtra("oauth_token"):"";
        String status = data.getStringExtra("mpstatus")!=null ?
data.getStringExtra("mpstatus"):"";

        switch (status){
            case "success":
                String cnasit = eghl.generateId("CNASIT");
                params = new PaymentParams.Builder()

.setMerchantReturnUrl("https://test2pay.ghl.com/IPGSimulatorJeff/RespFrmGW.aspx")
                        .setPaymentDesc("payment without previous pairing")
                        .setCustPhone("60123456789")
                        .setLanguageCode("EN")
                        .setPageTimeout("500")
                        .setServiceId(SERVICE_ID)
                        .setAmount(amountEdit.getText().toString())
                        .setCustName(nameEdit.getText().toString())
                        .setCustEmail(emailEdit.getText().toString())
                        .setMerchantName(merchantEdit.getText().toString())
                        .setCurrencyCode(currencyEdit.getText().toString())
                        .setToken(tokenEdit.getText().toString())
                        .setTokenType(tokenTypeEdit.getText().toString())
                        .setTransactionType(transactionTypeEdit.getText().toString())
                        .setPaymentMethod(paymentMethodEdit.getText().toString())
                        .setPairingToken(pairingToken)
                        .setReqToken(reqToken)
                        .setCheckoutResourceURL(checkoutURL)
                        .setPairingVerifier(pairingVerifier)
                        .setReqVerifier(reqVerifier)
                        .setPaymentId(cnasit)
                        .setOrderNumber(cnasit);
                Bundle paymentParams = params.build();
                eghl.executePayment(paymentParams, MainActivity.this);

                break;
            case "cancel":
            // handle cancel
                Toast.makeText(this, "Masterpass cancelled", Toast.LENGTH_SHORT).show();
                break;

            case "failure":
            //handle failure
                Toast.makeText(this, "Masterpass failed", Toast.LENGTH_SHORT).show();

                break;
        }

    }
```

Lightbox Values Returned Keys

Key	Description
mpstatus	String that indicates whether the Masterpass flow resulted in success, failure, or cancel.
checkout_resource_url	The API URL that will be used to retrieve checkout information

Key	Description
oauth_verifier	Checkout verifier token.
oauth_token	Checkout request token. This token has the same value as the checkout request token that is submit in Masterpass UI (Lightbox).
pairing_verifier	Pairing verifier token.
pairing_token	Pairing request token. *Note: Optional return field*

Additional parameters for the Second Request (with lightbox):

Parameter	Description
params.setPairingToken(pairingToken)	Request token (oauth_token) received from Masterpass UI (Lightbox)
params.setReqToken(reqToken)	Request verifier (oauth_verifier) received from Masterpass UI (Lightbox)
params.setCheckoutResourceURL(checkoutURL)	Pairing request token (pairing_token) received from Masterpass UI (Lightbox) Note: This value is needed when value return by Masterpass UI (Lightbox)
params.setPairingVerifier(pairingVerifier)	Pairing request verifier (pairing_verifier) received from Masterpass UI (Lightbox) *Note: This value is needed when value return by Masterpass UI (Lightbox)*
params.setReqVerifier(reqVerifier)	Checkout URL (checkout_resource_url) received from Masterpass UI (Lightbox)

Masterpass Express (without LightBox)

When you received a JSON response with **PreCheckoutId** and an array of cards, it means that the user's account is already paired. You must let the user choose his/her preferred card.

After he/she is done choosing a card you have to then execute another normal payment request – just like in the previous flow (with LightBox pairing), but this time you have to put the **PreCheckoutId** and the **CardId** of the chosen card by the user as the additional parameters:

```
private void proceedExpress(String response) {
    final ExpressResponse expressResponse = new
Gson().fromJson(response,ExpressResponse.class);
    List<Card> cards = expressResponse.getCards();
    final CardsAdapter adapter = new CardsAdapter(MainActivity.this, cards);

    AlertDialog.Builder builder = new AlertDialog.Builder(MainActivity.this);
    builder.setTitle("Cards");
    builder.setSingleChoiceItems(adapter, -1, new DialogInterface.OnClickListener() {
```

```
        public void onClick(DialogInterface dialog, int item) {
        Card card = (Card) adapter.getItem(item);
            String cnasit = eghl.generateId("CNASIT");
            params = new PaymentParams.Builder()

.setMerchantReturnUrl("https://test2pay.ghl.com/IPGSimulatorJeff/RespFrmGW.aspx")
                    .setPaymentDesc("payment without previous pairing")
                    .setCustPhone("60123456789")
                    .setLanguageCode("EN")
                    .setPageTimeout("500")
                    .setServiceId(SERVICE_ID)
                    .setAmount(amountEdit.getText().toString())
                    .setCustName(nameEdit.getText().toString())
                    .setCustEmail(emailEdit.getText().toString())
                    .setMerchantName(merchantEdit.getText().toString())
                    .setCurrencyCode(currencyEdit.getText().toString())
                    .setToken(tokenEdit.getText().toString())
                    .setTokenType(tokenTypeEdit.getText().toString())
                    .setTransactionType(transactionTypeEdit.getText().toString())
                    .setPaymentMethod(paymentMethodEdit.getText().toString())
                    .setCardID(card.getCardId())
                    .setPreCheckoutID(expressResponse.getPreCheckoutId())
                    .setPaymentId(cnasit)
                    .setOrderNumber(cnasit);
            Bundle paymentParams = params.build();
            eghl.executePayment(paymentParams, MainActivity.this);
            dialog.dismiss();

        }
    });
    AlertDialog alert = builder.create();
    alert.show();
}
```

Additional parameters for the Second Request(express):

Parameter	Description
params.setCardID(cardId)	The cardId of the chosen card by the user
params.setPreCheckoutID(preCheckout)	The PreCheckoutId from the first reponse

2.2.3. Making a Query

1. Create and supply the necessary parameters for query by using the **QueryParams.Builder** class.

```
QueryParams.Builder builder = new QueryParams.Builder();
builder.setCurrencyCode("MYR");
builder.setPaymentMethod("ANY");
builder.setAmount("10.00");
builder.setPaymentId("CNASIT77918966729033");
builder.setServiceId("GHL");
```

2. Build the parameters by calling the **build()** method on the **QueryParams.Builder** object. It will generate a **Bundle** object with the query parameters that you've set.

3. Execute the query by calling the **executeQuery(Context context, Bundle params, QueryCallback callback)** on the **EGHL** instance. Pass in the the object that you've created from the previous step as the second parameter and a callback object as the third parameter.

```
eghl.executeQuery(MainActivity.this, queryParams, new QueryCallback() {
    @Override
    public void onResponse(QueryResponse response) {

    }

    @Override
    public void onError(Exception e) {

    }
});
```

The QueryResponse object contains the result of the query.

3. Helpers

- **MasterPassButton** – There's also a **Button** class for **Masterpass**. It extends the **Button** class and sets its backround with the **Masterpass** logo. You can use it inside your **Activity** or **Fragment** layout. The optimal size is 166x38.

```
<com.eghl.sdk.MasterPassButton
    android:layout_width="166dp"
    android:layout_height="38dp"
    android:id="@+id/masterpass"
    android:layout_alignParentBottom="true"
    android:layout_toRightOf="@+id/checkout"
    android:layout_toEndOf="@+id/checkout" />
```

- **ELogger** – If you want to see the debug logs set it by calling **ELogger.setLoggable()** and passing-in a boolean. true if you want to show, false if you want it hidden.

- **JavaDocs** – The SDK is also very well documented in JavaDocs. It's available alongside this document. Just open the folder name **EGHL-JavaDocs-v2.1.0** and open the **index.html** file in your browser.

- **Demo app** – There's also a demo app alongside this document that you can reference to.

4. Migrating from V1.x to 2.x

The following ways have been changed from v1.x to 2.x

1. Password and PaymentGateway should now be defined inside your **AndroidManifest** as a **meta-data.**

2. Building Payment and Query parameters should now be done using a **Builder** pattern. **PaymentParameters.Builder** and **QueryParameters.Builder** etc. class could now help you build the parameters needed.

3. You don't have to query after the payment is finished anymore. The SDK will do it for you. You just need to handle the results of the payment via **onActivityResult()**

4. Masterpass checkout is introduced.

5. From the previous version of the SDK we are using a JAR file as a container for the SDK. In v2.0, the SDK is now inside an AAR file. You have to import it via gradle script.

5. Appendix

5.1. Response format for payment request

No.	Field	Data Type	Max Length	Req?	Description
1.	TransactionType	A	7	**Y**	Follows request
2.	PymtMethod	A	3	**Y**	Payment Method **CC –** Credit Card (Online 3D/Non3D) **MO –** Credit Card **(**MOTO – Mail Order Telephone Order) **DD –** Direct Debit **WA –** e-Wallet **CC –** Credit Card
3.	ServiceID	AN	3	**Y**	Follows request
4.	PaymentID	AN	20	**Y**	Follows request
5.	OrderNumber	AN	20	**Y**	Follows request
6.	Amount	N	12(2)	**Y**	Follows request for transaction not entitled for promotion For transaction entitled for promotion, Amount will not be the same as the Amount provided in Payment Request. Amount will be the actual payment amount which is the final amount after deducting promotion discount. The original Amount will be available in PromoOriAmt
7.	CurrencyCode	N	3	**Y**	Follows request
8.	HashValue	AN	100	**Y**	Message digest value calculated by Payment Gateway in **hexadecimal** string using **SHA256** hash algorithm Re: Section 2.13.1.2

9.	HashValue2	AN	100	**Y**	**HIGHLY RECOMMENDED** to verify this message digest value calculated by Payment Gateway in **hexadecimal** string using **SHA256** hash algorithm Re: Section 2.13.1.2
10.	TxnID	AN	30	**Y**	Unique Transaction ID or Reference Code assigned by Payment Gateway for this transaction
11.	IssuingBank	AN	30	**Y**	Follows request if this field is provided in request If not provided in request, For PymtMethod “CC”, this field is the Bank Name keyed in by customer on eGHL Payment Info Collection Page For PymtMethod “DD”, this field is the Bank Name chosen by customer to perform Direct Debit transaction
12.	TxnStatus	N	4	**Y**	Re: Section 3.2
13.	AuthCode	AN	12	N	Authorization Code returned by bank
14.	TxnMessage	AN	255	N	Message that briefly explains the response
15.	TokenType	A	3	N	Token Type If merchant is subscribed to eGHL One-Click Payment feature, upon payment approved, TokenType will be “**OCP**”, together with token value in **Token** field
16.	Token	ANS	50	C	Token Value If Token Type is “**OCP**”, Token field will hold the Token Value for One-Click Payment purposes
17.	Param6	ANS	50	C	Follows request
18.	Param7	ANS	50	C	Follows request

19.	CardHolder	AN	30	C	Cardholder's Name Only available as per request
20.	CardNoMask	AN	19	C	Credit Card Number used for payment authorization, first 6 and last 4 digits are in clear, the rests are masked with "X". e.g. 444433XXXXXX1111 Only available as per request
21.	CardExp	N	6	C	Expiry date of credit card. Date format is **YYYYMM**, e.g. 201312 for year 2013 December Only available as per request
22.	CardType	A	10	C	Credit Card Type e.g. VISA/MASTERCARD/AMEX/ JCB/DINERS Only available as per request
23.	SettleTAID	N	10	C	Terminal Account ID identifying the respective TID used to process the transaction and to be used for TxnType SETTLE. Only available for MOTO transaction (PymtMethod MO)
24.	TID	N	8	C	Actual terminal ID assigned by the bank to perform the transaction Only available for MOTO transaction (PymtMethod MO)
25.	EPPMonth	N	2	C	Follows request
26.	EPP_YN	N	1	C	Identifier for installment entitlement **0** – Not entitled for installment **1** – Entitled for installment

27.	PromoCode	AN	10	C	Promotion Code Follows request if PromoCode is provided in Payment Request If PromoCode is not provided in Payment Request but the transaction is entitled for promotion, PromoCode will be available in Payment Response
28.	PromoOriAmt	N	12(2)	C	Follows request's Amount for transaction entitled for promotion Original amount before deducting promotion discount. Only available for transaction entitled for promotion

5.2. Response format for query request

No.	Field	Data Type	Max Length	Req?	Description
1.	TxnExists	N	1	**Y**	An identifier to indicate whether the transaction being queried exists in Payment Gateway. **0 –** Transaction being queried **exists** in Payment Gateway. Merchant System can proceed to refer to the rest of other fields for details, e.g. TxnStatus **1 –** Transaction being queried **does not exist** in Payment Gateway. In other words, Payment Gateway is not able to find any transaction that is matched with all Query request fields submitted by merchant system **TxnStatus** will be **-1** **2 –** There was some kind of internal **error** occurred during query processing. Merchant System can retry sending query request **TxnStatus** will be **-2**
2.	QueryDesc	AN	255	N	Description of query result

5.3. Payment/Capture Transaction Status

TxnStatus	Description
0	Transaction successful
1	Transaction failed
2	Transaction pending

5.4. Query Transaction Status

TxnStatus	Description
0	Transaction successful (for transaction type SALE)
1	Transaction failed
2	Sale pending, retry Query
10	Transaction refunded
15	Transaction authorized (for transaction type AUTH)
16	Transaction captured
31	Reversal pending, merchant system can retry Reversal if merchant system initiated the Reversal request or else merchant system can retry Query
9	Transaction reversed
-1	Transaction not exists/not found
-2	Internal system error

5.5. Currency Code

Currency Code	Currency
MYR	Malaysia Ringgit
SGD	Singapore Dollar
THB	Thai Baht
CNY	China Yuan (Ren Min Bi)
PHP	Philippine Peso

5.6. Language Code

Language Code	Language
EN	English (default language)
BM	Malaysia
TH	Thai
CN	Chinese

5.7. Country Code

Country Code	Country
MY	Malaysia
TH	Thailand
SG	Singapore
PH	Philippines
CN	China
ID	Indonesia