Dragonpay Online Payment

Merchant Payment Switch API

Version 2.22 – May 22, 2025

Table of Contents

1. About this Document

This document describes the Application Programming Interface (API) between Payment Switch (PS) and the Merchant's e-commerce website. The PS is responsible for communicating with the financial partner's (eg. Bank) payment gateway for payment requests using a separate API. Upon validating the request, it redirects the end-user to his funding source of choice. The information needed by the PS to process a merchant payment for a transaction is transmitted using the API described in this document.

This document provides an overall introduction to the system, including its general architecture and structure. It then goes into detail on how to actually implement the system.

If you have any questions please do not hesitate to contact **sales@dragonpay.ph**.

2. Intended Audience

The intended audience for this document is technical personnel or programmers with background knowledge of programming and e-commerce. The examples in this document are written in Microsoft C# .NET. However, the programmer is free to implement the interfaces using other programming languages as long as they conform to Web standards such as HTTP GET, Name-Value Pair, and JSON/RESTcalls.

3. Change Log

Version	Date	Changes
2.00	Sept 9, 2019	Base Version for REST v1 implementation
2.01	Jan 16, 2020	Additional fields for Collect
2.02	Mar 20, 2020	Transaction History retrieval endpoints
2.03	Jun 17, 2020	Added missing Section 5.2.3
2.04	Jul 10, 2020	Added lifetime id activation support
2.05	Jul 14, 2020	Modified transaction history inquiry to use GET param
2.06	May 15, 2021	Added note to use clear text passwords in frontend application
2.07	Jul 9, 2022	Added *PreferredId* parameter for 5.3.3.1
2.08	Nov 21, 2022	Added INPY to sample procId in 5.4.1.2
2.09	Jan 11, 2023	Added amount/ccy/procId to 5.2.3; added note on txnid format for lifetime id's
2.10	Mar 2, 2023	Added documentation for MUVA vs LID
2.11		Added Bin optional field for MUVA creation
2.12	Jan 27, 2024	Updated email information
2.13	Feb 26, 2024	Added Updating lifetime user for LID and Muva
2.14	Mar 03, 2024	Added RefNo, MobileNo, ProcMsg and Fee on 5.3.1.2 Response Parameters
2.15	March 14, 2024	Added ABQR sample request, note and sample response.
2.16	May 9, 2024	Added information for generating static QR for MUVA/LID
2.17	July 15, 2024	Updated Appendices Terminology updated- Collection API key (old password)
2.18	Sept 30, 2024	Added information related to HMAC-SHA256 signature
2.19	Nov 27, 2024	Added settledate for the callback
2.20	March 6, 2025	Added new GCash Webflow implementations
2.21	April 7, 2025	Update LID/MUVA parameter description for preferred id
2.22	May 22, 2025	Required details for E-Wallet Merchants: additional notes added in section 5.2.1.

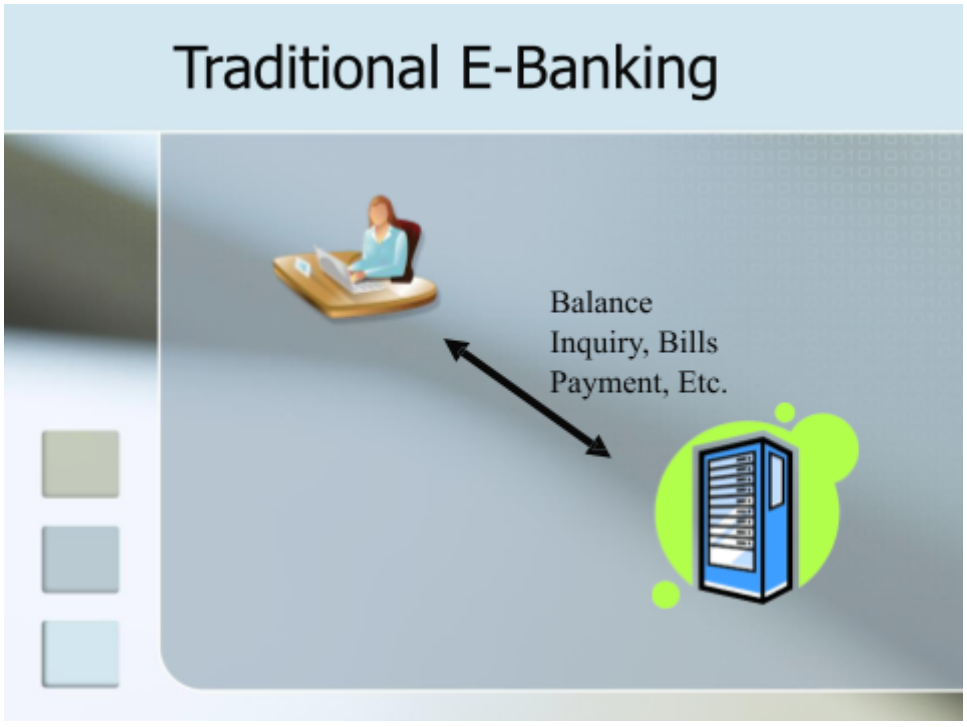
4. Introduction

E-commerce is gaining more and more acceptance by the general public each day. Its full potential, however, is hampered by the lack of available online payment options. While credit cards remain to be the most popular online payment option, most consumers shy away from it for fear of getting their card information compromised. Online merchants are also very wary of credit cards because of the high fraud rate. And for those selling high-ticket items, the percentage-based fee structure of credit cards is not appealing. Furthermore, only a small percentage of the population has access to credit cards because of credit history requirements.

Online bank debit payment presents a very effective alternative to this dilemma. Opening a bank account is certainly simpler than opening a credit card account. This presents a larger potential customer base to online merchants. The online banking interface is also inherently more secure than the usual credit card interface. This gives assurance to the customer that the transaction is safe. And because there is no concept of chargebacks with debit payments, merchants are also assured of payments for their products or services.

4.1 What is online bank debit payment?

In a typical online banking session, bank customers can perform basic functions such as balance inquiry, bills payment, checkbook reorder, and funds transfer remotely from their homes or offices. The bank's online interface is simply accessed using a web browser over a secure channel (https).

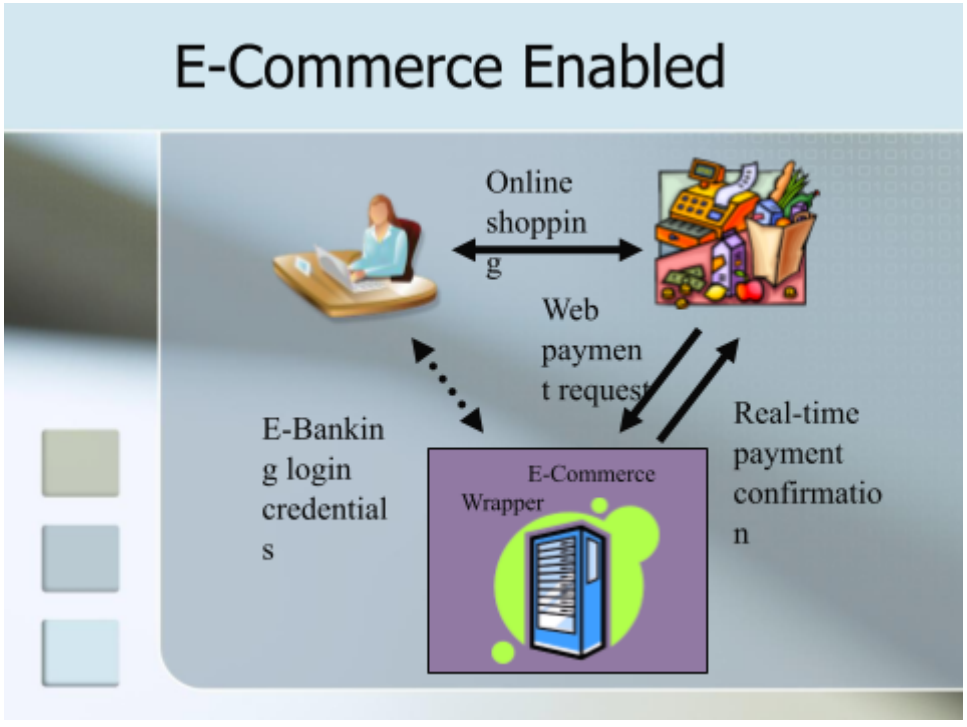


Under this scenario, the bank's system assumes that it is transacting with a live person. It responds to the requests sent by the bank customer over the browser. These requests are made by navigating through the web interface's menu system and by filling up on-screen forms.

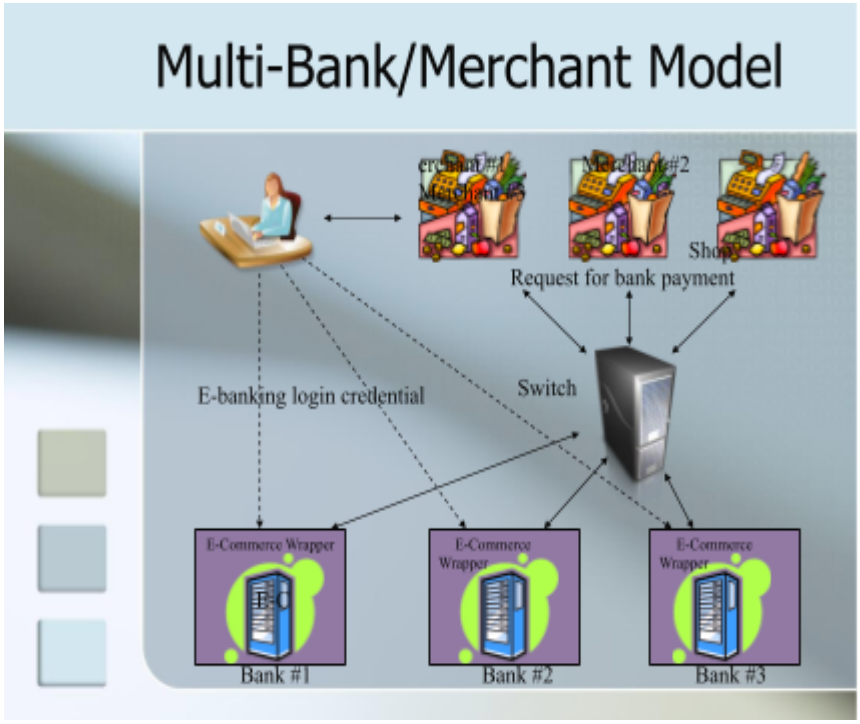
Online banking systems are normally not designed to work with e-commerce merchants or online stores which require machine-to-machine communication. They do not have the capability to accept requests programmatically from 3rd party websites or applications (ex. Shopping cart systems) for debiting the bank account of a particular customer. Subsequently, online banking systems also do not have the capability to communicate with a 3rd party system to inform it if a payment was done successfully or not.

Because of these limitations, it is currently impossible for online merchants to bill customers using their bank accounts in an automated, single-flow process. Merchants normally resort to off-line means such as asking the customer to deposit to their bank account over-the-counter and fax them the deposit slip as proof of payment. This makes it impossible to do e-commerce which require real-time responses (ex. airline ticketing, digital downloads). For merchants with high-volume transactions, the manual validation of deposit slips is also not a scalable solution.

PS seeks to address the problem by providing a "wrapper" interface to the online banking system. This will provide 3rd party online store applications with a programmatic interface to request for payments from the customer's bank, and for the bank to provide real-time feedback or confirmation if the payment was successful or not. In doing so, PS can enable any existing online banking platform to provide e-commerce functionality without or with very little changes, if any.



PS will also perform the role of a traffic cop. It will route the payment request to the appropriate bank chosen by the customer. It will accept payments from the customer on behalf of the merchant, and it will settle with the merchants on a scheduled basis.



M

4.2 How does online bank debit payment work?

All online transactions generally follow the same pattern.

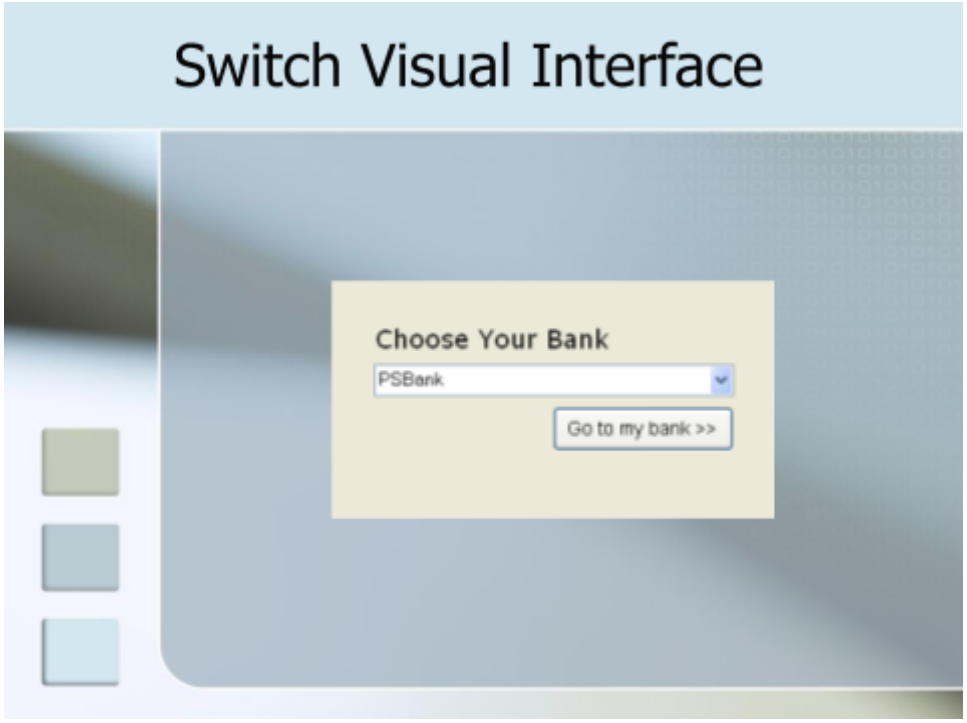
1. Customer surfs an online store
2. Customer clicks on items that he wants
3. Item is placed in an online shopping cart
4. Customer goes to *Checkout*
5. Customer is presented with several payment options
6. Customer clicks on the payment option he prefers
7. Payment processing is performed
8. Online shopping is completed

Where the shopping experience generally vary is in step #7. Different payment options have different process flows. Credit card payments are usually more straightforward – you enter your card details; click a button to confirm; and it's done. Most of the time, the customer does not have to leave the store's *Checkout* page.

With most other payment options (ex. PayPal, BancNet), however, the customer's browser is first redirected to the secure website of the payment processor. From there, he is asked to enter his credentials (ex. PayPal account id and password, BancNet ATM card number and PIN). When all information is entered correctly and

the transaction is confirmed, the customer's browser is redirected back to the online store (step #8) where the shopping is completed.

The PS process flow follows the general convention of the other payment options. From the *Checkout* page, the customer is redirected to PS and is presented with a list of banks to choose from.



Customer picks his bank from the list and clicks the button to proceed. PS will then transfer the request to the bank using the API described in this document. At this stage, the bank will generally perform the following operations:

1. Prompt for the necessary credentials (online banking id and password)
2. Let the customer choose from a list of available bank accounts (ex. checking account, savings account)
3. Confirm with the customer if he wants to charge the transaction against his chosen account. At this stage, some banks may perform additional authentication (ex. prompting for a transaction password, retrieving confirmation via SMS or email, random number generator)

When payment processing is completed, the customer is sent back to the PS using the return API described in this document.

PS keeps track of all payment transaction requests and their statuses. It talks to the bank systems in real-time, as well as, with the merchant shopping systems. It performs the role of the traffic cop and ensures all messages are routed to the appropriate party.

5. Payment Switch API

This section of the document describes the Merchant Payment Switch (PS) API in detail, covering the various functions used, as well as, codes that can be used to integrate them.

5.1 System Requirements

In order to integrate with the PS, Merchant must fulfill the following prerequisites:

1. Merchant website/app must be capable of getting the required data from customers (ex. amount, item description, email)
2. Merchant website/app can send HTTP request data to the PS system when a customer wishes to pay the Merchant.
3. Merchant website/app must have a Postback URL to accept real-time confirmation from PS.

Each Merchant is assigned the following:

- merchant id – unique code identifying the Merchant
- collection API key – a unique key assigned to the Merchant for checksum validation.

To authenticate REST/JSON payment requests, PS uses the industry-standard HTTP Basic Authentication method wherein the assigned merchant id and collection API key is Base64-encoded and passed as part of the HTTP request header.

For .NET developers, you can use a code like this to include the basic authentication in the request header:

```
C/C++
HttpWebRequest request = (HttpWebRequest)WebRequest.Create(url);
request.Credentials = new System.Net.NetworkCredential(merchantId, collection
API key);
```

If you prefer to manually add basic authentication to the headers, the same can be achieved by doing something like this:

```
C/C++
HttpWebRequest request = (HttpWebRequest)WebRequest.Create(url);
var credentials = System.Text.Encoding.UTF8.GetBytes(merchantId + ":" +
collection API key);
string token = System.Convert.ToBase64String(credentials);
request.Headers.Add("Content-Type", "application/json");
request.Headers.Add("Authorization", "Basic " + token);
```

Note: collection API key should never be hardcoded in the front-end application where it is viewable in clear text. All API calls should only be made from the merchant's backend servers for security.

5.2 Requesting a Payment

The general flow for making a payment request is as follows:

1. Merchant system requests for a payment to be collected from the PS via JSON/REST, passing all the necessary parameters
2. PS replies with a message containing a status, refno, and url
3. Merchant system performs a browser redirect to the passed url
4. Upon payment completion, PS will call the Merchant system's postback url and redirect the browser back to the Merchant system's return url.

You may use the following URLs as the service entry point for Version 1 (v1) implementation.

Service Production base URL:

```
Unset
https://gw.dragonpay.ph/api/collect/<version-no>
```

Service Test base URL:

```
Unset
https://test.dragonpay.ph/api/collect/<version-no>
```

For merchants migrating from the first generation API, using <*version-no*> of "v1" will provide backward compatibility. The *url* returned will be similar in flow to the original API. For new integrations, we recommend using <*version-no*> of "v2".
Thus, the url to be used would normally be:

Service Production base URL:

```
Unset
https://gw.dragonpay.ph/api/collect/v1
```

Service Test base URL:

```
Unset
https://test.dragonpay.ph/api/collect/v1
```

NOTE: For API version 2 "**procId**" is a required property when creating transactions.

5.2.1 Request Parameters

These are the parameters passed by the Merchant via JSON/REST to request for a payment to be collected. Make sure to set the header *Content-Type* to *application/json*.

NOTE: Merchants that provide E-Wallet Services are required to pass the Account Number and Name in the description field e.g. "0000-000-0000 - Juan Dela Cruz"

Endpoint: <baseurl>/<txnid>/post
Method: POST

NOTE: The *txnid* must be alphanumeric (dashes are allowed) and must not exceed more than 40 characters in length.

Parameter	Data Type	Description
Amount	Numeric(12,2)	The amount to collect (XXXX.XX)
Currency	Char(3)	The currency (see Appendix 1)
Description	Varchar(128)	A brief description of the request
Email	Varchar(40)	Email address of customer
MobileNo	Varchar(20)	[OPTIONAL] mobile no of customer
ProcId	Varchar(4)	[OPTIONAL] preferred payment channel
Param1	Varchar(80)	[OPTIONAL] value that will be posted back to merchant postback/return url when completed
Param2	Varchar(80)	[OPTIONAL] value that will be posted back to merchant postback/return url when completed [REQUIRED for AGGREGATOR setup]
Expiry	DateTime	[OPTIONAL] payment expiry period (best effort)
BillingDetails	BillingInfo	[OPTIONAL] billing details of the customer needed for credit card transactions
SenderShippingDetails	ShippingInfo	[OPTIONAL] shipping details of the sender needed for COD transactions
RecipientShippingDetails	ShippingInfo	[OPTIONAL] shipping details of the recipient needed for COD transactions
IpAddress	Varchar(16)	[OPTIONAL] IP address of end-user
UserAgent	Varchar(256)	[OPTIONAL] Browser user agent of end-user

You may keep the *ProcId* blank if you want the user to perform the selection on Dragonpay's side (recommended). If you wish to pre-select the channel, please refer to Section 5.4 of this document on advanced controls.

For credit card transactions, the merchant system must pass the *BillingDetails* field. For Cash on Delivery (COD) transactions, the merchant system must pass the sender and recipient shipping addresses.

The BillingInfo structure is as follows:

Parameter	Data Type	Description
FirstName	Varchar(60)	First name of customer
LastName	Varchar(60)	Last name of customer
Address1	Varchar(120)	Street address
Address2	Varchar(120)	Village, subdivision, etc.
City	Varchar(40)	City or municipality
State	Varchar(40)	State or province
Country	Varchar(2)	2-char ISO country code (ex. PH, US, CA)
ZipCode	Varchar(12)	[OPTIONAL] zip code
TelNo	Varchar(40)	Telephone number of customer
Email	Varchar(40)	Email address of customer

The ShippingInfo structure is as follows:

Parameter	Data Type	Description
FirstName	Varchar(60)	First name of customer
MiddleName	Varchar(60)	Middle name of customer
LastName	Varchar(60)	Last name of customer
Address1	Varchar(120)	Street address
Address2	Varchar(120)	Village, subdivision, etc.
Barangay	Varchar(60)	Barangay
City	Varchar(40)	City or municipality
Province	Varchar(40)	State or province
Country	Varchar(2)	2-char ISO country code (ex. PH, US, CA)
ZipCode	Varchar(12)	[OPTIONAL] zip code
Landmark	Varchar()	[OPTIONAL] landmarks or directions to help courier locate the address
TelNo	Varchar(40)	Telephone number of customer
Email	Varchar(40)	Email address of customer

At a bare minimum, a sample JSON payment request payload for Metrobankdirect online can look like:

https://test.dragonpay.ph/api/collect/v2/test20200118001/post

```
Unset
{
        "Amount": "100.00",
        "Currency": "PHP",
        "Description": "Sample transaction",
        "Email": "juan.dela.cruz@sampledomain.com",
        "ProcId": "MBTC"
}
```

Below is a sample JSON payload to request collection with GCash:

https://test.dragonpay.ph/api/collect/v2/test20200118002/post

```
Unset
{
	"Amount": "100.00",
	"Currency": "PHP",
	"Description": "Sample transaction",
	"Email": "juan.dela.cruz@sampledomain.com",
	"ProcId": "GCSH"
}
```

Below is a sample JSON payload to request collection with QRPH (ABQR):
https://test.dragonpay.ph/api/collect/v2/test20200118002/post

```
Unset
{
	"Amount": "100.00",
	"Currency": "PHP",
	"Description": "Sample transaction",
	"Email": "juan.dela.cruz@sampledomain.com",
	"ProcId": "ABQR"
}
```

Below is a full sample JSON payload to request collection with credit cards:

https://test.dragonpay.ph/api/collect/v2/test20200118004/post

```
Unset
{
	"Amount": "100.00",
	"Currency": "PHP",
	"Description":"Sample Transaction",
	"Email": "juan.dela.cruz@sampledomain.com",
	"ProcId": "CC",
	"Param1": "Test parameter 1",
	"Param2": "Test parameter 2",
	"BillingDetails": {
		"FirstName": "Juan",
		"MiddleName": "Dela",
		"LastName": "Cruz",
		"Address1": "123 Sesame Street",
		"Address2": "Childrens Television",
		"Workshop", "City": "Marikina",
		"Province": "Metro Manila",
		"Country": "PH",
		"ZipCode": "1800",
		"TelNo": "86556820",
		"Email": "juan.dela.cruz@sampledomain.com"
	},
	"IpAddress": "123.12.4.67",
	"UserAgent": "Mozilla/5.0 (Windows NT 10.0; WOW64) AppleWebKit/537.36
(KHTML, like Gecko) Chrome/48.0.2564.103 Safari/537.36"
}
```

Below is a full sample JSON payload to request collection with COD + Delivery:

https://test.dragonpay.ph/api/collect/v2/test20200118005/post

```
Unset
{
	"Amount": "100.00",
	"Currency": "PHP",
	"Description": "Sample transaction",
	"Email": "juan.dela.cruz@sampledomain.com",
	"ProcId": "BAE",
	"Param1": "Test parameter 1",
	"Param2": "Test parameter 2",
	"SenderShippingDetails": {
		"FirstName": "Juan",
		"MiddleName": "Dela",
		"LastName": "Cruz",
		"Address1": "170 Salcedo Street",
		"Address2": "Legaspi Village",
		"Barangay": "San Lorenzo",
		"City": "Makati",
		"Province": "Metro Manila",
		"Country": "PH",
		"ZipCode": "1229",
		"Landmark": "Near Greenbelt Park",
		"TelNo": "79751111",
		"Email": "operations@companyabc.com"
	},
	"RecipientShippingDetails": {
		"FirstName": "Juan",
		"MiddleName": "Dela",
		"LastName": "Cruz",
		"Address1": "123 Sesame Street",
		"Address2": "Childrens Television Workshop",
		"Barangay": "San Roque",
		"City": "Marikina",
		"Province": "Metro Manila",
		"Country": "PH",
		"ZipCode": "1800",
		"Landmark": "Near Sta Lucia Mall",
		"TelNo": "86556820",
		"Email": "juan.dela.cruz@sampledomain.com"
	},
	"IpAddress": "123.12.4.67",
	"UserAgent": "Mozilla/5.0 (Windows NT 10.0; WOW64) AppleWebKit/537.36
	(KHTML, like Gecko) Chrome/48.0.2564.103 Safari/537.36"
}
```

5.2.2 Response Parameters

After posting the transaction details via REST, PS replies back with the following JSON message:

Parameter	Data Type	Description
RefNo	Varchar(10)	A unique sequence assigned by PS to this request
Status	Char(1)	(S)uccessful or (F)ailed payment initiation request.
Message	Varchar(128)	If status is (F)ailed, this may contain additional reason as to why it failed.
Url	Varchar(256)	The url that merchant system should redirect the browser to.

If *Status* returned is (S)uccess, Merchant system should then redirect the customer's browser to the provided *Url* parameter. Take note that *Status* here is not referring to the collection status of the transaction itself, but just of the payment initiation request. So a (S)uccess only means that a payment request has been initiated. It does not mean that the payment has already been collected successfully.

For Version 1 (v1) implementation, the *RefNo* parameter will return a token and not the actual Dragonpay refno. You may just send the browser to *Url* and PS will handle the rest.

For Version 2 (v2) implementation, the *RefNo* parameter is the actual unique reference no of the transaction on the Dragonpay side. If merchant is passing a pre- selected *ProcId* in the collect request in conjunction with the ***GetAvailableProcessors*** service discussed further in Section 5.4, they must check if that *ProcId* has its *MustRedirect* flag set or not. If it is set, then browser must be redirected to the *Url* next. Otherwise, the *Url* must be just pointing to a regular instruction page that the merchant's UI can hide or display in its desired format.

NOTE: For ABQR, there is an additional parameter in the response, which is "**QRPH**". You can utilize this parameter by appending the value of the QRPH parameter to *https://gw.dragonpay.ph/Bank/GenerateQRPHCode.aspx?code*= to retrieve the QR code. This QR code can be used in case you want to customize your UI to display only the QR code.

Below is the sample JSON response of ProcId=ABQR.

```
Unset
{
       "RefNo": "LMRW4THXB4",
       "Status": "S",
       "Message": "Successfully created refno",
       "Url":"https://test.dragonpay.ph/Bank/ProcessAllBank.aspx?procid=A
BQR&refno=LMRW4THXB4&amount=100.00&ccy=PHP&description=Deposit&billerId=
&email=abc%40gmail.com&digest=e982109e45cebb18d76914149ba49fea86b3ffc7&e
xpiry=3%2f14%2f24+01%3a17&merchantid=TEST&txnid=fpOZoc9A26t0m5wiCQ811U12
M1a",
       "QRPH":"00020101021128760011ph.ppmi.p2m0111OPDVPHM1XXX031577714800
000002204165294813724664091050300052046016530360858O2PH5909Dragonpay6015
City Of
Mandalu62280010ph.allbank0510LMRW4THXB488310012ph.ppmi.qrph0111OPDVPHM1X
XX6304F40B"
}
```

5.2.3 Payment Completion Response Parameters

When payment processing has completed, the PS should redirect back the customer's browser to the Merchant's registered callback URL and pass along the parameters below.

IMPORTANT Note: Merchants should upgrade to use HMAC-SHA256 signature

Parameter	Description
txnid	A unique id identifying this specific transaction from the merchant side
refno	A common reference number identifying this specific transaction from the PS side
status	The result of the payment. Refer to Appendix 3 for codes.
message	If *status* is SUCCESS, this should be the PG transaction reference number. If *status* is FAILURE, return one of the error codes described in Appendix 2. If *status* is PENDING, the message would be a reference number to complete the funding.
amount	The amount processed
ccy	The currency of the amount processed
procid	The Payment Channel used to process the payment
settledate	Datetime of when the end user has paid
digest	A sha1 checksum digest of the parameters along with the secret key.
signature	An HMAC-SHA256 digest of the parameters along with a different secret key. The digest is encoded in hex and in UPPERCASE. This is a more secure way to verify our callbacks.

PS recognizes two kinds of Callback URLs – one is the HTTP POST *Postback URL* and the HTTP GET *Return URL*. The *postback URL* is invoked directly by the PS and does not expect any return value. Because the invocation is directly done by the PS, it is very difficult to fake. The merchant can perform additional source IP address validation to ensure it is the PS making the call. The *postback URL* handler should return with a simple `content- type:text/plain` containing only the single line: `result=OK`.

The *return URL* is passed to the customer's browser via an HTTP redirect. The merchant normally responds with a visual web page reply informing the customer the status of the transaction.

It is not necessary for the merchant to implement both callback URLs, although it is recommended. PS will always invoke the *postback URL* first before the browser redirects to the *return URL*. Thus the ideal process flow is: upon receiving the *postback URL* call, the merchant's system performs the necessary database updates and initiates whatever back-end process is required. Then when it receives the *return URL* call, it counter-checks the status in the database and provides the visual response. If a merchant does not provide both callback URLs, PS will only invoke the one provided.

An HTTP GET from PS to either callback URLs may look something like this:

```
Unset

http://www.abcstore.com/Postback.aspx?txnid=1234&refno=5678&status=S&
message=72843747212&digest=a4b3d08462......&signature=ADSADE123Q2....
```

The digest is computed using the SHA1 algorithm. Below is a sample code showing how to generate the SHA1 digest using C# .NET:

```
C/C++
String message = String.Format("{0}:{1}:{2}:{3}:{4}",
       Request["txnid"].ToString(),
       Request["refno"].ToString(),
       Request["status"].ToString(),
       Request["message"].ToString(),
       Application["secretkeysha1"].ToString());
String digest = GetSHA1Digest(message);
```

Then compare against the passed digest:

```
C/C++
if (GetSHA1Digest(message) != Request["digest"].ToString()) {
       // Display some error message and abort processing
} else {
       // If status = "SUCCESS", process customer order for shipment
}
```

The code sample below shows how to compute the HMAC-SHA256 signature. Take note that amount is included here!

NOTE: Appendix 4 provides the complete implementation of the GetHMACSHA256Digest() function.

```
C/C++
String message = String.Format("{0}:{1}:{2}:{3}:{4}",
       Request["txnid"].ToString(),
       Request["refno"].ToString(),
       Request["status"].ToString(),
       Request["message"].ToString(),
       Request["amount"].ToString());
String secretKey = Application["secretkeysha256"].ToString();
String signature = GetHMACSHA256Digest(message, secretKey);
```

Then compare against the passed signature:

```
C/C++
String secretKey = Application["secretkey"].ToString();
String expectedSignature = GetHMACSHA256Digest(message, secretKey);

if (expectedSignature != Request["signature"].ToString()) {
       // Display some error message and abort processing
} else {
       // If status = "SUCCESS", process customer order for shipment
}
```

In cases wherein the transaction *status* returned is PENDING, the merchant may receive an asynchronous call to the *postback URL* in the future once the funding is completed. The format will just be similar to the HTTP GET callback described above. If a *postback URL* is not defined for the merchant, PS will invoke the *return URL* instead. The merchant should take care in checking the *status* and should only ship goods or render service when *status* value has become SUCCESS.

```
Unset
Special note for Lifetime Id's or Virtual Accounts (VA's), the dynamically
generated txnid follows the format "<lifetimeid>-yyMMddHHmm" for LID and
"<MUVA>-yyMMddHHmm" for MUVA.
```

Security Enhancement: Upgrading to HMAC-SHA256

To further ensure that our Payment Completion Response / webhook / callback isn't falsifiable, we are upgrading from using SHA1 to HMAC-SHA256. We're doing a phased rollout so merchants can adhere to the more secure hashing algorithm. During this slow rollout phase, we're sending out both the SHA1 *digest*, and HMAC-SHA256 *signature* so nothing will break. **We expect merchants to notify us once they've successfully migrated to HMAC-SHA256 so we can disable SHA1 for them, thus removing that attack vector.**

5.3 Additional Support Functions

The PS provides some supplementary functions allowing merchants to more tightly integrate and automate their systems.

5.3.1 Transaction Status Inquiry

The merchant can programmatically inquire the status of a transaction by using this function.

5.3.1.1 Request Parameters

These are the parameters passed by the Merchant to the PS via JSON/REST request for a transaction status. The merchant id and collection API key must be passed to the endpoint using standard HTTP Basic Auth username and password.

Endpoint: <baseurl>/refno/<refno>
Method: GET

Parameter	Data Type	Description
refno	Varchar(20)	A unique Dragonpay refno assigned to the specific transaction from the merchant side

Alternatively, if you do not have the Dragonpay refno, you may use the merchant-assigned transaction id using this endpoint.

Endpoint: <baseurl>/txnid/<txnid>
Method: GET

Parameter	Data Type	Description
txnid	Varchar(40)	A unique id identifying this specific transaction from the merchant side

5.3.1.2 Response Parameters

The endpoint above returns a JSON-formatted record with the following fields:

Parameter	Data Type	Description
RefNo	Varchar(20)	A unique sequence assigned by PS to this request
MerchantId	Varchar(20)	A unique code assigned to Merchant
TxnId	Varchar(40)	A unique id identifying this specific transaction from the merchant side
RefDate	DateTime	Timestamp when transaction was requested
Amount	Numeric(12,2)	The amount to get from the end-user (XXXX.XX)
Currency	Char(3)	The currency of the amount (see Appendix 1)
Description	Varchar(128)	A brief description of what the payment is for
Status	Char(1)	Transaction status (see Appendix 3)
Email	Varchar(40)	Email address of customer
MobileNo	Varchar(15)	Mobile number of customer
ProcId	Varchar(4)	Payment channel selected
ProcMsg	Varchar(180)	Messages for transaction specific to procId
SettleDate	DateTime	Timestamp when transaction was completed
Param1	Varchar(80)	[OPTIONAL] value that will be posted back to merchant url when completed
Param2	Varchar(80)	[OPTIONAL] value that will be posted back to merchant url when completed
Fee	Float	Fee charged by Dragonpay to Merchant

NOTE: If the transaction cannot be located, an HTTP Status 404 (not found) will be returned.

5.3.2 Cancellation of Transaction

The merchant can programmatically cancel a pending transaction by using this function.

5.3.2.1 Request Parameters

These are the parameters passed by the Merchant to the PS via REST request for voiding of a pending transaction.

Endpoint: <baseurl>/void/<txnid>
Method: GET

Parameter	Data Type	Description
txnid	Varchar(40)	A unique id identifying this specific transaction from the merchant side

5.3.2.2 Response Parameters

The service will respond with a single *status* string:

Parameter	Description
Status	Returns zero (0) if successful, else a negative number
Message	Extra description regarding the cancellation request

If the transaction cannot be located, an HTTP Error Status 404 (not found) will be returned.

5.3.3 Multiple-Use Virtual Accounts and Lifetime Id Functions

The merchant can programmatically manage Lifetime Id's (LID) or Multiple-Use Virtual Accounts (MUVA) by using these functions. LID's are alphanumeric and are generally used with non-bank OTC channels. MUVA are 16-digit numbers used with Instapay and PESONet. For single-use VA payments, use the standard flow (5.2).

NOTE: Before you can use this feature you must coordinate with your Account Manager so Dragonpay can provide the required BIN number that is required to create MUVA.

5.3.3.1 Request Parameters for Creation of MUVA or LID

These are the parameters passed by the Merchant to the PS via REST request for a lifetime id.

IMPORTANT NOTE: Dragonpay MUVA is always 16 digits. For 6-digit BINs, a 10-digit PreferredId is required; for 8-digit BINs, an 8-digit PreferredId is needed. The BIN and PreferredId together always form a 16-digit MUVA. For Merchant Supplied LID, the PreferredId must be between 6 and 10 digits.

Endpoint: <baseurl>/lifetimeid/create
Method: POST

Parameter	Data Type	Description
Prefix	Char(2)	The LID prefix assigned to the merchant
Bin	Char(6)	[**OPTIONAL**] provide the assigned 6-digit BIN instead of the 2-char LID prefix for MUVA use
Name	Varchar(60)	Name of the customer assigned to this id
Email	Varchar(60)	Email of the customer assigned to this id
Remarks	Varchar(80)	Additional remarks regarding this id
PreferredId	Varchar(6) or Varchar(8) or Varchar(10)	[**REQUIRED for MUVA and Merchants with Supplied LIDs**] Merchant provided unique id assigned to the customer.
GenerateQR	Boolean	[**OPTIONAL**] Returns a QR string relative to the created MUVA/LID. **Note**: This parameter is only available on API v2 for API v1 please see section **5.3.3.5 Generating Static QR for Existing MUVA or LID**

Sample request for regular LID using merchant-assigned prefix with merchant- supplied 6-alphanumeric id. This will return a lifetime id of "TT123456".

```
Unset
{
        "Prefix": "TT",
        "Name": "Juan dela Cruz",
        "Email": "juan.dela.cruz@sampledomain.com",
        "Remarks": "Customer 123456",
        "PreferredId": "123456"
}
```

Sample request of a MUVA using Dragonpay-assigned BIN 705299 and merchant-supplied 10-digit id. This will return a MUVA value of "70529912345890".

```
Unset
{
        "Bin": "705299",
        "Name": "Juan dela Cruz",
        "Email":"juan.dela.cruz@sampledomain.com",
        "Remarks": "Customer 123456",
        "PreferredId": "1234567890"
}
```

By default Static QR is only available in API v2. However, merchants using API v1 can use the endpoint mentioned in section **5.3.3.5 Generating Static QR for Existing MUVA or LID** to generate a Static QR tied to existing LID/MUVA.

```
Unset
{
        "Prefix": "TT",
        "Name": "Juan dela Cruz",
        "Email": "juan.dela.cruz@sampledomain.com",
        "Remarks": "Customer 123456",
        "GenerateQR": true
}
```

Sample MUVA request payload that will generate a QR string.

```
Unset
{
        "Bin": "705299",
        "Name": "Juan dela Cruz",
        "Email":"juan.dela.cruz@sampledomain.com",
        "Remarks": "Customer 123456",
        "PreferredId": "1234567890",
        "GenerateQR": true
}
```

5.3.3.2 Response Parameters for Creation of MUVA or LID

For API Version 1, The service will respond with a single *lifetime id/VA* string:

Parameter	Description
LifetimeId/MUVA	Returns generated LID or MUVA, or empty string if failed.

For API Version 2 the service will return the following parameters.

Parameter	Description
LifetimeId	The generated LID, returns null if failed to generate or using MUVA.
MUVA	The generated MUVA, returns null if failed to generate or using LID.
QRPH	String representation of the generated QR. Returns empty if failed or the GeneratedQR parameter is set to false.

5.3.3.3 Request Parameters for Activation/Deactivation of MUVA or LID

These are the parameters passed by the Merchant to the PS via REST request for a lifetime id. Replace <verb> with either *activate* or *deactivate*.

Endpoint:<baseurl>/lifetimeid/<verb>/<id>
Method: GET

Parameter	Data Type	Description
Id	Varchar(16)	The LID or MUVA to activate or deactivate

5.3.3.4 Response Parameters for Activation/Deactivation of MUVA or LID

The service will respond with an http status 200 if successful else status 404.

5.3.3.5 Generating Static QR for Existing MUVA or LID

Generate a static QRPH tied to the provided existing LID.

NOTE: For merchants who wanted to utilize static QR for existing LIDs. A way for v1 merchants to use static QR.

Endpoint:<baseurl>/lifetimeid/<id>/generate-static-qr
Method: POST

Parameter	Data Type	Description
Id	Varchar(16)	The LID or MUVA to generate a static QR for

5.3.3.6 Request Parameters for Retrieving MUVA or LID

These are the parameters passed by the Merchant to the PS via REST request for a lifetime id.

Endpoint:<baseurl>/lifetimeid/<id>
Method: GET

Parameter	Data Type	Description
Id	Varchar(16)	The LID or MUVA to retrieve

5.3.3.7 Response Parameters for Retrieving MUVA or LID

The service will respond with the following record:

Parameter	Data Type	Description
userId	Varchar(12)	The assigned LID/MUVA
merchantId	Varchar(20)	The merchant id who owns the lifetime id
custName	Varchar(60)	The registered name of the customer using the id
custEmail	Varchar(60)	The registered email address of the customer using the id
remarks	Varchar(80)	Additional remarks regarding the customer using the id
status	Char(1)	(A)ctive or (I)nactive
created	Timestamp	Date and time when the id was created
staticQr	Varchar(512)	The generated static QR string tied to the LID/MUVA provided. Returns null if QR is not yet generated.

5.3.3.7 Updating Lifetime Users for MUVA and LID

These are the parameters passed by the Merchant to the PS via REST request for a lifetime id and Va number.

Endpoint: <baseurl>/lifetimeid/<LID or VA>
Method: PATCH

Parameter	Data Type	Description
Lifetime id / VA	Varchar(12)	The assigned Lifetime id / VA
Name	Varchar(60)	The registered name of the customer using the Lifetime id / VA
Email	Varchar(60)	The registered email address of the customer using the Lifetime id / VA
remarks	Varchar(80)	Additional remarks regarding the customer using the Lifetime id / VA

Sample request for regular LID using merchant-assigned prefix with merchant-supplied 6-alphanumeric id. This will return a lifetime id of “TT123456”.

```
Unset
{
	"name": "Juan dela Cruz",
	"email": "juan.dela.cruz@sampledomain.com,
	"remarks": "Customer 123456"
}
```

Sample request of a MUVA using Dragonpay-assigned BIN 705299 and merchant-supplied 10-digit id. This will return a MUVA value of “70529912345890”.

```
Unset
{
	"Name": "Juan Dela Cruz",
	"Email": "juan.dela.cruz@sampledomain.com",
	"Remarks": "Customer 123456"
}
```

5.3.3.8 Response parameters for Updating Lifetime Users for MUVA and LID

The server will respond with an HTTP Code 200 if successful, and no response body will be thrown, else, HTTP Error (e.g 404, etc.) will be thrown, and a brief message regarding the error.

Parameter	Description
Message	Extra description for updating request

5.3.4 Transaction History Functions

The merchant can programmatically retrieve a list of transactions

5.3.4.1 Request for Transaction History

To get the list of all transactions between a date/time range via REST, you may call this endpoint.

Endpoint: <baseurl>/transactions?startdate={start}&enddate={end}
Method: GET

Both *{start}* and *{end}* may use the format *yyyy-MM-dd* to indicate a date range. If you need to indicate the time as well, use the format *yyyy-MM-ddThh:mm:ss*. Since this is an HTTP GET parameter, make sure to url-encode the colons (":") to %3A. If no specific hour is indicated with the *{end}* parameter, it is assumed to be up to 23:59:59.999 of that date.

5.3.4.2 Response for Transaction History

The system will respond with an array of records containing the details of all transactions within the period.

5.3.4.3 Request for History of Successfully Completed Transactions

To get the list of all transactions between a date/time range via REST, you may call this endpoint.

Endpoint:
<baseurl>/transactions/settled?startdate={start}&enddate={end}
Method: GET

See note on 5.3.4.1 for date/time format.

5.3.4.4 Response for History of Successfully Completed Transactions

The system will respond with an array of records containing the details of all transactions within the period.

5.4 Customization of Payment Selection

There may be instances wherein the merchant would want to filter the payment channels that they want to appear in Dragonpay's payment selection page, or they may want to skip the Dragonpay page altogether and go straight to the payment details for a specific channel. There is support for these features and this section discusses them in detail.

There are two general forms of customization:

1. Simple control of what payment options appear in Dragonpay's dropdown list
2. Moving the payment selection process to the merchant side and calling Dragonpay in the background

5.4.1 Simple Control

With the simple method, the process flow is still essentially the same – merchants redirect the page to Dragonpay for the buyer to make the payment selection. However, merchants can control to a certain degree which options appear in the payment selection list, or merchants can make a pre-selection to a specific channel.

5.4.1.1 Filtering Payment Channels

Dragonpay payment channels are grouped together by type – ex. Online banking, Over-the-Counter/ATM, etc. Merchants can programmatically instruct Dragonpay which grouping to show when the user is redirected to the payment gateway by using the "mode" parameter.

Mode Value	Grouping Type
1	Online Banking
2	Over-the-Counter Banking and ATM
4	Over-the-Counter non-Bank
8	E-Wallets (inc. Bitcoins)
16	(reserved internally)
32	PayPal
64	Credit Cards
128	Mobile (Gcash)
256	International OTC
512	Bancnet
1024	Auto Debit Arrangement (ADA)
2048	Cash on Delivery (COD)
4096	Installments

"Mode" is a bitmask which can be OR-ed to achieve the result intended. The following example will only show the online banking options:

```
Unset
https://gw.dragonpay.ph/Pay.aspx?merchantid=ABC&txnid=1234&…&mode=1
```

Merchants who avail of PayPal or GCash from Dragonpay but do not want them to appear in the dropdown list, may specify a "mode=7" to display only the basic alternative payments in the dropdown list.

5.4.1.2 Pre-selecting Payment Channels

Dragonpay has basic support to allow merchants to go directly to a payment channel without having to select it from the dropdown list. The following are sample processor id's which can be used to go straight to the selection:

Proc Id	Name
BDO	BDO Online Banking
CC	Credit Cards
CEBL	Cebuana Lhuillier
DPAY	Dragonpay Prepaid Credits
ECPY	ECPay
GCSH	Gcash
PYPL	PayPal
MLH	M. Lhuillier
RDS	Robinsons Dept Store
SMR	SM Payment Counters
711	7-Eleven (if applicable)
INPY	Instapay (if applicable)

Merchants who want to receive Gcash or PayPal payments may put separate radio buttons at their checkout page to give users the capability to go straight to that channel without stopping by the Dragonpay payment selection page by passing a "procid" parameter.

The following example will direct the buyer to our Gcash payment page from the merchant's checkout page:

```
Unset
https://gw.dragonpay.ph/Pay.aspx?merchantid=ABC&txnid=1234&…&procid=GCSH
```

For PayPal and credit card acceptance, Merchant is required to apply for a separate acquirer's id with the respective payment gateways. Contact our Sales for assistance.

NOTE: With the recent GCash (GCSH) updates they implemented a new flow for specific platforms. Please see Appendix 5 - GCash Payment Flow for more details.

5.4.2 Advanced Control

If the merchant wishes to keep the payment user experience as close to their checkout page as possible, Dragonpay provides support to perform this to a certain extent.

The general process flow is as follows:

1. Call the *GetAvailableProcessors* web service to retrieve a list of all supported payment channels.
2. Merchant dynamically render its checkout page depending on the result set of *GetAvailableProcessors*.
3. Merchant redirects the browser to the Payment Url passing the selected processor id (procid) as parameter.
4. When payment is completed, Dragonpay invokes the Merchant's Postback / Return URLs.

5.4.2.1 Determining Available Payment Channels

Depending on various factors, some payment channels may not be available at all times. Merchants who implement the techniques mentioned in the previous section to perform payment channel selection at their checkout page, and use the *procid* parameter to send the user directly to the payment channel instruction, have to be careful not to send the user to an inactive channel.

Merchants can query Dragonpay for the available payment channels at a particular point in time by using the REST Web Service *GetAvailableProcessors*.

Merchants are strongly discouraged from statically listing Dragonpay payment channels as there are various rules that determine their availability. These include:

1. Some processors are only available on certain days of the week (ex. Not available on weekends or non-banking days).
2. Some processors are only available between certain times of the day (ex. Goes down nightly for maintenance).
3. Some processors have limits on the minimum or maximum amount that can be processed through them.
4. Scheduled or unscheduled system maintenance.

For these reasons, Merchants who want to customize the user experience by moving the payment selection onto their checkout page have to be aware of all these rules. Otherwise, customers may encounter problems.

5.4.2.1.1 GetAvailableProcessors Request Parameters

To retrieve the list of all processor channels, you can use this:

Endpoint: <baseurl>/processors

To retrieve the list of available processor channels for a particular amount, you can use this:

Endpoint: <baseurl>/processors/available/<amount>

Parameter	Data Type	Description
amount	Numeric(12,2)	The amount of the transaction

If an **amount** value greater than zero is passed, *GetAvailableProcessors* will return a list of channels available for that amount. But if you want to retrieve the full list regardless of the amount so you can cache it locally and avoid having to calling the web method for each transaction, you can set **amount** to a special value of –1000.

5.4.2.1.2 GetAvailableProcessors Response Parameters

The web service will return an array of records in a REST/JSON or XML/SOAP envelope format. Each record contains the following fields:

Parameter	Data Type	Description
procId	Varchar(4)	A unique code assigned to this processor
shortName	Varchar(15)	A brief name for this processor. Can be used if UI space is limited.
longName	Varchar(40)	A longer, more descriptive name of the processor. Can be used if UI space allows.
logo	Varchar(160)	A url pointing to the logo of this procid that Dragonpay uses
currencies	Varchar(80)	A comma-delimited list of currencies that this procid can support.
type	Integer	Bitmask. Refer to Section 5.4.1.1 for various meanings
status	Char(1)	Can be (A)ctive or (I)nactive. As of this writing, GetAvailableProcessors only return (A)ctive procid's.
remarks	Varchar(320)	This string may be displayed by merchant in its checkout page to give user more details or descriptions about what this procid is about.
dayOfWeek	Char(7)	A string mask corresponding to the 7 days of the week starting from Sunday and ending Saturday. If an "X" is in the mask position, that means the procid is available on that day; else, it is unavailable and should not be displayed.
startTime	Char(5)	Starting time when this procid is available to process (in 24-hr "HH:MM" format)
endTime	Char(5)	Ending time after which this procId is no longer

		available to process.

minAmount	Numeric(12,2)	The smallest amount this procid can process.
maxAmount	Numeric(12,2)	The amount over-and-above which procid is not allowed to process.
mustRedirect	Bool	This flag tells the Merchant whether a browser redirect is mandatory.
surcharge	Numeric(12,2)	The amount added for payments using this channel
hasAltRefNo	Bool	Has a 10-digit alternate refno used when paying

Additional Notes:

1. If **dayOfWeek** is "0XXXXX0", for example, that means it is not available on Sundays and Saturdays, but is available from Monday to Friday.
2. It is strongly recommended that Merchant uses the **remarks** field to display tips or additional description when the channel is selected. This field will also inform the user if there are any surcharges that may be applied for using this channel.
3. If **startTime**=**endTime**, then this procId is available 24-hrs a day. If **endTime** is "00:00", but **startTime** is not "00:00", then **endTime** should be interpreted as the stroke of midnight.
4. The **minAmount** and **maxAmount** fields should be implemented as follows – `if (amount >= minAmount && amount < maxAmount) then` proceed with this channel, else do not show this channel. That is **not** `amount <= maxAmount`.
5. It is recommended that the *GetAvailableProcessors* web service be invoked by a scheduled cron job every 30 mins to every hour with **amount** = -1000. While the field values generally will not change, the status can change during the day for various reasons. For example, a bank partner may have an unscheduled downtime. If Merchant does not refresh its internal copy of this list, it may think the channel is still active whereas it has already been deactivated temporarily (or permanently) on Dragonpay's side.

Appendix 1 – Currency Codes

Code	Description
PHP	Philippine Peso
USD	US Dollar

Appendix 2 – Error Codes

Code	Description
000	Success
101	Invalid payment gateway id
102	Incorrect secret key
103	Invalid reference number
104	Unauthorized access
105	Invalid token
106	Currency not supported
107	Transaction canceled
108	Insufficient funds
109	Transaction limit exceeded
110	Error in operation
111	Security Error
112	Invalid parameters
201	Invalid Merchant Id
202	Invalid Merchant Password

Appendix 3 – Status Codes

Code	Description
S	Success
F	Failure
P	Pending
U	Unknown
V	Void

Appendix 4 – Upgrading to HMAC-SHA256

Overview

We want our move towards using HMAC-SHA256 *signature* and away from SHA1 *digest*, to be as seamless as possible. We encourage merchants to generate a new SHA256 secret key so it's different from the current SHA1 secret key. However, to reduce friction, we're defaulting the SHA256 secret key to your current secret key and use **UTF-8** encoding. Moving forward, we plan to generate SHA256 secret keys encoded in Hex so to reduce errors due to character encoding.

Once you have finished the build to verify our *Payment Completion Response* or callback/webhook using the HMAC-SHA256 *signature*, you can also start removing the check for SHA1 digest. Notify us also so we can also stop sending the SHA1 *digest* as part of our *Payment Completion Response* and just send the HMAC-SHA256 *signature*.

Here's a few examples of messages, SHA256 secret keys, and signatures. Note that 53517666774C716C397A6A4F32363936 is just SQvfwLql9zjO2696 in hex encoding.

- Using SHA256 secret key encoded in UTF-8 [Online Tool]
    - Message: 37f193f8-0494-4356-a2df-868eaef2ed08:P4WNRADT71:S:[000] BOG Reference No: 20240905104752 #P4WNRADT71:13.56
    - Secret Key (UTF-8): SQvfwLql9zjO2696
    - Signature: BCD945971A4002384046D1D75EBC316FE6C294073B3B737577313A6907B97ACA
- Using SHA256 secret key encoded in Hex [Online Tool]
    - Message: 37f193f8-0494-4356-a2df-868eaef2ed08:P4WNRADT71:S:[000] BOG Reference No: 20240905104752 #P4WNRADT71:13.56
    - Secret Key (Hex): 53517666774C716C397A6A4F32363936
    - Signature: BCD945971A4002384046D1D75EBC316FE6C294073B3B737577313A6907B97ACA

Migration Flow

1. Merchant builds the feature to verify our callback using HMAC-SHA256 signature
2. Merchant test that the signature they're receiving matches what they've computed
3. Merchant updates their software to stop using the SHA1 digest as a verification
4. Merchant notifies devops@dragonpay.ph so we stop sending SHA1 digest to them
5. Merchant tests if they're successfully migrated to HMAC-SHA256 signature: meaning they're able to verify our callbacks AND they're not receiving the SHA1 digest anymore

Sample Function written in C#

```C/C++
public static string GetHMACSHA256Digest(string message, string keyInHex)
{
	// Hex Decode the Secure Secret for use in using the HMACSHA256 hasher
	// Hex decoding eliminates this source of error as it is independent
of the character encoding
	// Hex decoding is precise in converting to a byte array and is the
preferred form for representing binary values as hex strings.
	byte[] keyHexDecoded = new byte[key.Length / 2];
	for (int i = 0; i < key.Length / 2; i++)
	{
		keyHexDecoded[i] = (byte)Int32.Parse(key.Substring(i * 2, 2),
System.Globalization.NumberStyles.HexNumber);
	}

	System.Security.Cryptography.HMACSHA256 hmacsha256 = new
System.Security.Cryptography.HMACSHA256(keyHexDecoded);

	byte[] data = System.Text.Encoding.UTF8.GetBytes(message);
	byte[] result = hmacsha256.ComputeHash(data);

	System.Text.StringBuilder sb = new System.Text.StringBuilder();
	for (int i = 0; i < result.Length; i++)
		sb.Append(result[i].ToString("X2"));
	return sb.ToString();
}
```

FAQS

1. How do we generate a SHA256 secret key? You can email devops@dragonpay.ph for the meantime while we're working on adding this to our Web Portal
2. Why doesn't our signature match with the expected signature?
    a. One point of confusion is the encoding of the secret key. The example code above, `GetHMACSHA256Digest()` assumes that the key is in hex so you might need to hex encode before you put into your env variables.
    b. There's also a difference between the SHA1 message and SHA256 message: SHA256 also includes the amount.
    c. We pass the signature in UPPER CASE, it's best that your checks are case insensitive.

Appendix 5 - GCash Payment Flow

Overview

To enhance user security and provide a seamless online payment experience, GCash has introduced updates to its payment flow. These changes require users to authenticate their transactions through the GCash mobile application and implement advanced risk assessment protocols. The updated payment process involves two new payment methods that ensure secure and efficient handling of transactions across web-based and app-based platforms.

Key Changes

1. **User Authentication Requirement:**
    - All transactions now require users to log into their GCash mobile application to complete payments.
    - Transactions will be subject to enhanced risk assessment, leveraging multi-factor authentication methods such as OTP (One-Time Password), MPIN, and Face Scan.
2. **New Payment Options:**
    - **Redirect Button:** Users can click on a Redirect button to open the GCash app on the same device and proceed with payment.
    - **Redirect QR:** A dynamic QR code will be generated for each transaction, which users can scan using the GCash in-app scanner. Upon scanning, users will be redirected to select from their available payment options.

Action Needed

To ensure a smooth transition and uninterrupted payment functionality, action may be required based on your platform implementation:

1. **Web-Based Platform**
    - **No action is required.** The new payment flow will be automatically supported.
2. **App-Based Platform**
    - **Changes required.** Implement the necessary code changes in your mobile application to support the new GCash payment flows.

Code Implementation Guidelines

If you operate an app-based store, follow these steps to update your platform:

Android Implementation (WebView - Java)

1. Apply the provided function to your WebView class.
2. Test the integration in the UAT (User Acceptance Testing) environment to verify compatibility with both old and new payment flows.

```java
// Code Changes for ANDROID:

public boolean shouldOverrideUrlLoading(WebView view, WebResourceRequest
request) {

    String url = request.getUrl().toString();

    //Use gcash: on production and sit environment

    if (url.startsWith("gcash://")) {

        val i = Intent(Intent.ACTION_VIEW);

        i.data = Uri.parse(url);

        startActivity (i);

        return true;

        // Let the WebView handle the URL normally

    } else {

        // Use existing logic

        return false;

        //Allow the WebView in your application to do its thing

    }

}
```

iOS Implementation

- **Safari WebView:** No code changes are required. The behavior will update automatically once GCash rolls out the changes.
- **WebKit WebView (Objective-C & Swift):** Implement the provided function in your WebView navigation handler.

C/C++

```
// Code Changes for IOS:
- (void)viewDidLoad {
    self.webView.navigationDelegate = self;
}
- (void)webView:(WKWebView *)webView
decidePolicyForNavigationAction:(WKNavigationAction *)navigationAction
decisionHandler:(void (^)(WKNavigationActionPolicy))decisionHandler {
    NSString *scheme = [webView.URL scheme];
    NSString *query = [webView.URL query];
    NSURL *directUrl;
    if (scheme) {
        if ([scheme isEqualToString:@"gcash://"]){
            // Scheme URL
            directUrl = webView.URL;
        }
        if (directUrl) {
            UIApplication *application = [UIApplication sharedApplication];
            // scheme
            if (@available(iOS 10.0, *)) {
                [application openURL:[NSURL URLWithString:directUrl]
options:@{}
                completionHandler:^(BOOL success) {
                    if (success) {
                         // do something.
                    } }];
            } else {
                [application openURL:[NSURL URLWithString:directUrl]];
            }
        }
    }
    decisionHandler(WKNavigationActionPolicyAllow);
}
```