Dragonpay Online Payment

Merchant Payment Switch API

Version 1.01 – Nov 9, 2017

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 SOAP/XML Web Services calls.

3. Change Log

Version	Date	Changes
0.10	May 25, 2010	Alpha Version
0.11	June 28, 2010	Added email as required parameter
0.12	Aug 9, 2010	Changed merchant txnid to varchar(40)
		Updated URL's to api.dragonpay.ph
0.13	Nov 21, 2012	Added support for optional 'param1' and 'param2'
		Documented MerchantRequest.aspx
		Added SendBillingInfo web method
0.14	Dec 9, 2012	Changed server name from 'api' to 'secure'
0.15	Jan 25, 2013	Corrected 2nd parameter of SendBillingInfo
0.16	May 28, 2014	Changed live server from 'secure' to 'gw'
		Added Section 5.3.4
0.17	Sept 2, 2014	Added procid list in 5.3.4.2
0.18	Mar 23, 2015	Included CAD in Currency list
		Changed live web service from 'secure' to 'gw'
0.19	Sept 2, 2015	Updated proc list in 5.3.4.2
0.20	Nov 8, 2015	Added section 5.3.4.3
1.00	Jul 1, 2016	Added section 5.4
1.01	Nov 9, 2017	Updated 5.4.2.1.2

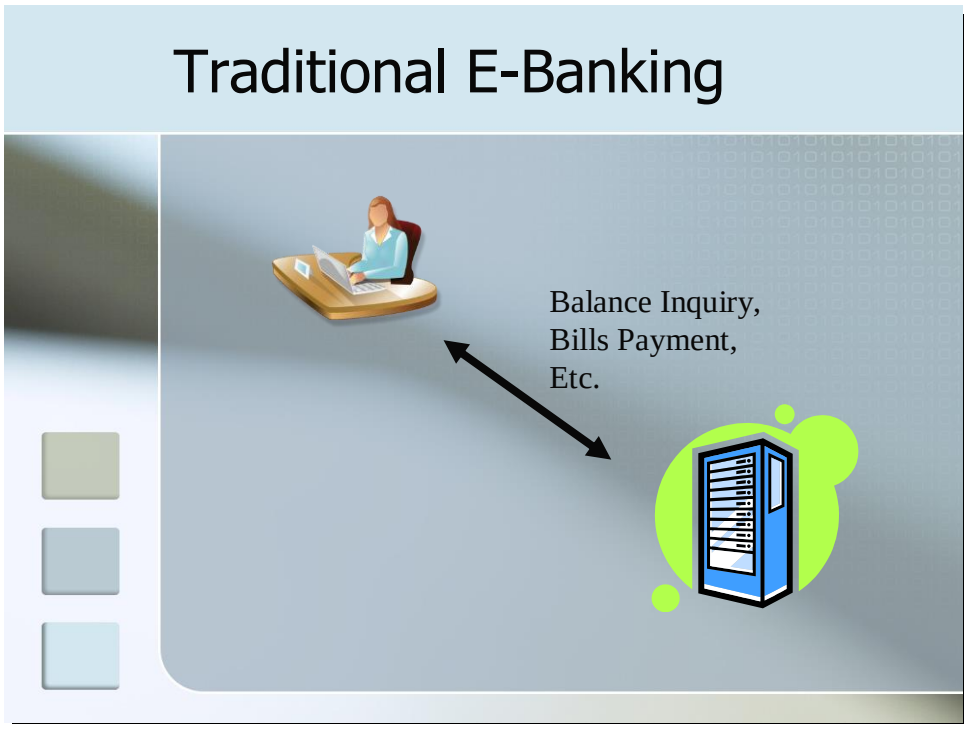
4. Introduction

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 card remains 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 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).

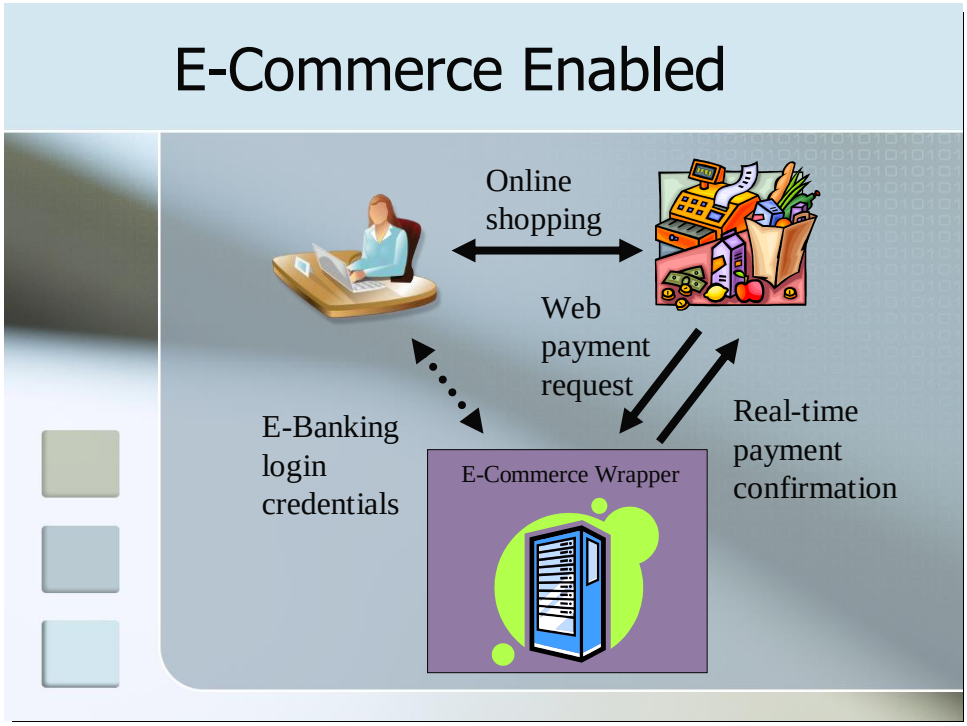


Under this scenario, the bank's system assumes that it is transacting with a live person. It responds to the requests sent by the bank customer over the browser. These requests are made by navigating through the web interface's menu system and by filling up on-screen forms.

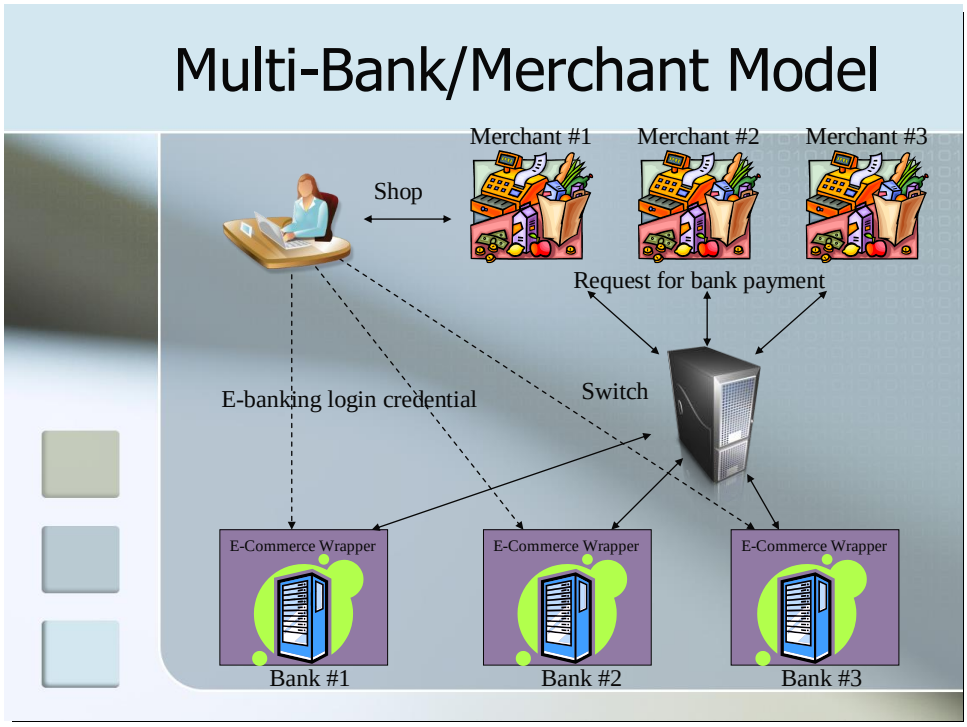
Online banking systems are normally not designed to work with e-commerce merchants or online stores which require machine-to-machine communication. They do not have the capability to accept requests programmatically from 3rd party websites or applications (ex. Shopping cart systems) for debiting the 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 in behalf of the merchant, and it will settle with the merchants on a scheduled basis.



4.2 How does online bank debit payment work?

All online transactions generally follow the same pattern.

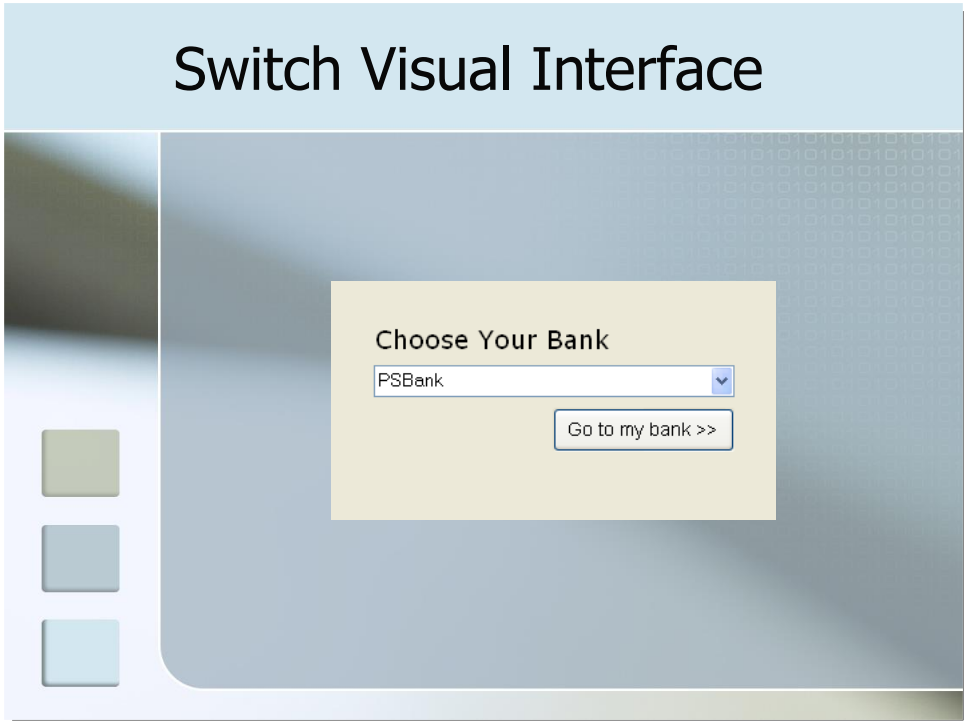
1. Customer surfs an online store
2. Customer clicks on items that he wants
3. Item is placed in an online shopping cart
4. Customer goes to *Checkout*
5. Customer is presented with several payment options
6. Customer clicks on the payment option he prefers
7. Payment processing is performed
8. Online 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 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 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, 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 site must be capable of getting the required data from customer (ex. amount, item description, email)
2. Merchant site can send http request data to PS system when a customer wishes to pay the Merchant with his bank account.
3. Merchant site 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
- secret key – a unique password assigned to Merchant for checksum validation

Production Payment URL:

```
https://gw.dragonpay.ph/Pay.aspx
```

Test Payment URL:

```
http://test.dragonpay.ph/Pay.aspx
```

Although this document uses Microsoft .NET conventions, it should be implementable under other operating environments (ex. Linux, PHP, Perl, Java). (Note that since this is an alpha documentation, the Postback URL's may change in the future.)

5.2 Message Passing (Merchant ->PS and PS->Merchant)

This section describes how the merchant will pass a request to the PS system for payment processing and vice versa. There are currently two integration models available – the Name-Value Pair Model and the Web Services Model.

5.2.1 Name-Value Pair Model

Under the Name-Value Pair Model, Merchant sends the request parameters using HTTP GET with a browser redirect. The PS system only needs to read and parse the GET Query String to extract all the necessary information.

The PS system can check the authenticity of the request by two means:

1. It can check the URL or IP address of the HTTP Referer and make sure it belongs to the Merchant.
2. It can use its secret key to compute for the message digest based on the parameters passed and compare it against the passed digest. If the computed digest does not match, then it should reject the transaction as the parameters have most likely been compromised.

5.2.1.1 Request Parameters

These are the parameters passed by the Merchant to the PS via name-value pairs to request for a payment.

Parameter	Data Type	Description
merchantid	Varchar(20)	A unique code assigned to Merchant
txnid	Varchar(40)	A unique id identifying this specific transaction from the merchant side
amount	Numeric(12,2)	The amount to get from the end-user (XXXX.XX)
ccy	Char(3)	The currency of the amount (see Appendix 1)
description	Varchar(128)	A brief description of what the payment is for
email	Varchar(40)	email address of customer
digest	Char(40)	A sha1 checksum digest of all the parameters along with the secret key.
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

An HTTP GET to PS may look something like this:

```
https://gw.dragonpay.ph/Pay.aspx?merchantid=ABC&txnid=12345678&amount=1000.00&
ccy=PHP&description=Box+of+Chocolates&digest=a4b3d08462......
```

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

```
public static string GetSHA1Digest(string message)
{
	byte[] data = System.Text.Encoding.ASCII.GetBytes(message);

	System.Security.Cryptography.SHA1 sha1 = new
		System.Security.Cryptography.SHA1CryptoServiceProvider();
	byte[] result = sha1.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().ToLower();
}
```

The message string is built by just concatenating all the required parameters with the assigned secret key and using the colon symbol for delimiter.

```
string message = String.Format("{0}:{1}:{2}:{3}:{4}:{5}:{6}",
       merchantId,
       txnId,
       amount.ToString("#0.00"),
       currency,
       description,
       email,
       Application["secretkey"].ToString());

String digest = GetSHA1Digest(message);

String redirectString =
       String.Format("{0}?merchantid={1}&txnid={2}&amount={3}&ccy={4}" +
              "description={5}&email={6}&digest={7}",
              paymentSwitchUrl,
              merchantId,
              txnId,
              amount.ToString("#0.00"),
              currency,
              Server.UrlEncode(description),
              Server.UrlEncode(email),
              digest);

// send browser to Payment Switch
Response.Redirect(redirectString, true);
```

5.2.1.2 Response Parameters

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

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.
digest	A sha1 checksum digest of the parameters along with the secret key.

PS recognizes two kinds of callback URL's – the *postback URL* and the *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 URL's, although it is recommended. PS will always invoke the *postback URL* first before the browser redirect 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 initiate 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 merchant does not provide both callback URL's, PS will only invoke the one provided.

An HTTP GET from PS to either callback URL's may look something like this:

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

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

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

Then compare against the passed digest:

```
if (GetSHA1Digest(message) != Request["digest"].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.

5.2.2 SOAP/XML Web Service Model

For greater security, the Merchant may choose to implement the API using the XML Web Services model. Under this model, the parameters are not passed through browser redirects which are visible to end-users. Instead, parameters are exchanged directly between the Merchant site and PS servers through SOAP calls.

The general flow of this method is:

1. Merchant system requests for a token from the PS via SOAP
2. PS replies with a token
3. Merchant system uploads the payment information via SOAP using the token as reference
4. Merchant system performs a browser redirect with the token as the parameter

The advantages of using this model are:

1. Parameters are not visible on the browser
2. Merchant server is sending the parameters directly to PS thus reducing the likelihood of 3rd party manipulation

You may use the following URL's as the Web Service entry point.

Web Service Production URL:

`https://gw.dragonpay.ph/DragonPayWebService/MerchantService.asmx`

Web Service Test URL:

`http://test.dragonpay.ph/DragonPayWebService/MerchantService.asmx`

5.2.2.1 Request Parameters

These are the parameters passed by the Merchant via SOAP to request for a token.

Web Method: GetTxnToken

Parameter	Data Type	Description
merchantId	Varchar(20)	A unique code assigned to Merchant
password	Varchar(20)	The password assigned to the Merchant
merchantTxnId	Varchar(20)	A unique id identifying this specific transaction from the merchant side
amount	Numeric(12,2)	The amount to get from the end-user (XXXX.XX)
ccy	Char(3)	The currency of the amount (see Appendix 1)
description	Varchar(128)	A brief description of what the payment is for
email	Varchar(40)	[OPTIONAL] email address of customer
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

The *GetTxnToken()* method will return a *tokenid* string which will be used to refer to this transaction in future Web Method calls. Note that validity of this *tokenid* is limited only to at most one (1) hour. If the value of *tokenid* is 3-characters or less, it must be an error code. Refer to Appendix 2 for the list of error codes. Possible errors are incorrect *merchantId* or *secretKey*.

After posting the transaction details via SOAP, the Merchant system performs an HTTP GET redirect with the following parameters:

Parameter	Data Type	Description
tokenid	Varchar(40)	The id returned by *GetTxnToken*

The code may look like this:

```
String redirectString =
       String.Format("{0}?tokenid={1}",
              paymentSwitchUrl,
              tokenid);

// send browser back to PS
Response.Redirect(redirectString, true);
```

5.2.2.2 Response Parameters

The response of PS to a payment request from the Merchant using the Web Service model is just similar to the one for Name-Value Pair. Refer to 5.2.1.2 for details.

5.3 Additional Support Functions

The PS provides some supplementary functions allowing merchants to more tightly integrate and automate their systems. These functions are available in Name-Value Pair HTTP GET or POST, and XML Web Service models

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 using Name-Value Pair

These are the parameters passed by the Merchant to the PS via name-value pairs to request for a transaction status. Name-value pairs may be sent using either HTTP GET or HTTP POST to the *MerchantRequest.aspx* function.

Parameter	Data Type	Description
op	Varchar(20)	The operation to perform (value = GETSTATUS)
merchantid	Varchar(20)	A unique code assigned to Merchant
merchantpwd	Varchar(20)	The merchant's API password
txnid	Varchar(40)	A unique id identifying this specific transaction from the merchant side

```
string message = String.Format("{0}:{1}:{2}",
       "GETSTATUS",
       merchantId,
       Application["secretkey"].ToString(),
       txnId);
```

An HTTP GET to PS may look something like this:

```
https://gw.dragonpay.ph/MerchantRequest.aspx?op=GETSTATUS&merchantid=ABC&
merchantpwd=MySecret&txnid=12345678
```

5.3.1.2 Response Parameters using Name-Value Pair

MerchantRequest.aspx will respond to the inquiry with a plain-text http reply:

Parameter	Description
status	The result of the payment. Refer to Appendix 3 for codes.

5.3.1.3 Request Parameters using XML Web Service

These are the parameters passed by the Merchant to the PS via SOAP request for a transaction status.

Web Method: GetTxnStatus

Parameter	Data Type	Description
merchantId	Varchar(20)	A unique code assigned to Merchant
merchantPwd	Varchar(20)	The API password assigned to Merchant
txnId	Varchar(40)	A unique id identifying this specific transaction from the merchant side

You may use the following URL's as the Web Service entry point. (Note that since this is an alpha documentation, the actual URL's may change in the future.)

5.3.1.4 Response Parameters using XML Web Service

The *GetTxnStatus()* method will respond with a single *status* string:

Parameter	Description
status	The result of the payment. Refer to Appendix 3 for codes.

For more details on error codes due to FAILURE, or reference numbers for SUCCESS or PENDING, please access the web-based administrator page.

5.3.2 Cancellation of Transaction

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

5.3.2.1 Request Parameters using Name-Value Pair

These are the parameters passed by the Merchant to the PS via name-value pairs to request for a transaction cancellation. Name-value pairs may be sent using either HTTP GET or HTTP POST to the *MerchantRequest.aspx* function.

Parameter	Data Type	Description
op	Varchar(20)	The operation to perform (value = VOID)
merchantid	Varchar(20)	A unique code assigned to Merchant
merchantpwd	Varchar(20)	The merchant's API password
txnid	Varchar(40)	A unique id identifying this specific transaction from the merchant side

```
string message = String.Format("{0}:{1}:{2}",
       "VOID",
       merchantId,
       Application["secretkey"].ToString(),
       txnId);
```

An HTTP GET to PS may look something like this:

```
https://gw.dragonpay.ph/MerchantRequest.aspx?op=VOID&merchantid=ABC&
merchantpwd=MySecret&txnid=12345678
```

5.3.2.2 Response Parameters using Name-Value Pair

MerchantRequest.aspx will respond to the request with a plain-text http reply:

Parameter	Description
status	Returns zero (0) if successful, else a negative number

5.3.2.3 Request Parameters using XML Web Service

These are the parameters passed by the Merchant to the PS via SOAP request for a transaction status.

Web Method: CancelTransaction

Parameter	Data Type	Description
merchantId	Varchar(20)	A unique code assigned to Merchant
password	Varchar(20)	The API password assigned to Merchant
merchantTxnId	Varchar(40)	A unique id identifying this specific transaction from the merchant side

5.3.2.4 Response Parameters using XML Web Service

The *CancelTransaction()* method will respond with a single *status* string:

Parameter	Description
status	Returns zero (0) if successful, else a negative number

5.3.3 Sending of Billing Information

For additional fraud checking, the merchant can programmatically send the customer's billing address by using this function.

5.3.3.1 Request Parameters using XML Web Service

These are the parameters passed by the Merchant to the PS via SOAP request.

Web Method: SendBillingInfo

Parameter	Data Type	Description
merchantId	Varchar(20)	A unique code assigned to Merchant
merchantTxnId	Varchar(20)	Mechant's unique transaction id
firstName	Varchar(60)	Firstname of customer
lastName	Varchar(60)	Lastname 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
email	Varchar(40)	Email address of customer

5.3.3.2 Response Parameters using XML Web Service

The *SendBillingInfo()* method will respond with a single *status* string:

Parameter	Description
status	Returns zero (0) if successful, else a negative number

5.3.4 Determining the Assigned Dragonpay Reference No

The merchant can programmatically determine the equivalent Dragonpay reference no given the merchant's transaction id by using this function.

5.3.4.1 Request Parameters using XML Web Service

These are the parameters passed by the Merchant to the PS via SOAP request.

Web Method: GetTxnRefNo

Parameter	Data Type	Description
merchantId	Varchar(20)	A unique code assigned to Merchant
merchantPwd	Varchar(20)	The API password assigned to Merchant
txnId	Varchar(40)	A unique id identifying this specific transaction from the merchant side

5.3.4.2 Response Parameters using XML Web Service

The *GetTxnRefNo()* method will respond with a single string representing the Dragonpay reference no assigned to the transaction.

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 – merchant redirects the page to Dragonpay for the buyer to make the payment selection. However, merchant can control to a certain degree which options appear in the payment selection list, or merchant 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)

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

```
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 very basic support to allow merchant 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
BAYD	Bayad Center
BITC	Bitcoins
CC	Credit Cards
CEBL	Cebuana Lhuillier
CUP	China UnionPay
DPAY	Dragonpay Prepaid Credits
ECPY	ECPay
GCSH	Globe Gcash
LBC	LBC
PYPL	PayPal
MLH	M. Lhuillier
RDS	Robinsons Dept Store
SMR	SM Payment Counters
BOL	Bancnet Online (if applicable)
711	7-Eleven (if applicable)

Merchants who want to receive Gcash or PayPal payments may put separate radio buttons at their checkout page to give user 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:

```
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 merchant id with the respective payment gateways. Contact our Sales for assistance.

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 method in MerchantService.asmx to retrieve a list of all supported payment channel.
2. Merchant dynamically render its checkout page depending on the result set of *GetAvailableProcessors()*.
3. Depending on which option user selected for checkout, Merchant calls the Payment Url passing the selected processor id (procid) as parameter.
4. Merchant may either redirect the browser to the specific procid page and let Dragonpay manage the UI, or perform a background HTTP GET and retrieve the instructions programmatically as JSON for customized displaying.
5. When payment is completed, Dragonpay invokes the Merchant's Postback / Return Url's.

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.

Merchant can query Dragonpay for the available payment channels at a particular point in time by using the SOAP 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 Request Parameters

To retrieve the list of available processor channels, the main starting point is *GetAvailableProcessors()* of MerchantService.asmx.

Web Service Production URL:

`https://gw.dragonpay.ph/DragonPayWebService/MerchantService.asmx`

Web Service Test URL:

`http://test.dragonpay.ph/DragonPayWebService/MerchantService.asmx`

Parameter	Data Type	Description
merchantid	Varchar(20)	A unique code assigned to Merchant
password	Varchar(40)	The password associated to the merchantid
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 Response Parameters

The web service will return an array of records in an 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. Merchant must pay attention to the **mustRedirect** field as it will tell you whether you really have to redirect the browser to work with this channel, or whether a background HTTP GET (ex. Wget / cUrl) can be invoked to create a transaction behind-the-scenes while keeping the browser in its current location. While most of the processor channels do not require a redirect, there are some that do.
6. It is recommended that the *GetAvailableProcessors()* web method 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.

5.4.2.2 Creating a Transaction and Retrieving the Instruction

Once a procid is selected, merchant can redirect to the payment gateway url (Pay.aspx) passing it as a parameter. This can be done in 1 of 2 ways:

1. Perform a browser redirect to Pay.aspx?procid=XXXX as described in Section 5.4.1.2.
2. Invoke Pay.aspx?procid=XXXX in the background programmatically using a mechanism similar to wget / cUrl. This method allows the user browser to stay at the merchant payment page with Dragonpay being called from behind-the-scenes.

Should you wish to take option 1, then proceed with a regular browser redirect as described in 5.4.1.2. Dragonpay takes over the user interface from there.

But should you decide to take the second option, then you must check the value of **mustRedirect**. If **mustRedirect** is **true**, then you have to perform a browser redirect no matter what, similar to option 1. But if **mustRedirect** is **false**, you may programmatically call Pay.aspx in the background. Once completed, you have to retrieve the Dragonpay reference no that was generated. This can be done using *GetTxnRefNo()* as described in 5.3.4.

Given the Dragonpay reference no, you can then retrieve the payment instruction by simply directing the user to the following url where "XXXXXXXX" is the Dragonpay reference no.

```
https://gw.dragonpay.ph/Bank/GetEmailInstruction.aspx?refno=XXXXXXXX
```

However, if you wish to further control how the instruction is rendered, you have the option of retrieving it in JSON format by calling:

```
https://gw.dragonpay.ph/Bank/GetEmailInstruction.aspx?refno=XXXXXXXX&format=json
```

The structure of the JSON response follows that of the default HTML layout:

Parameter	Description
introMsg	A 1- to 2- sentence basic description about paying through this channel. May be blank in some cases.
paymentDetails	This is a 2-dimensional array of name-value pairs containing a field and its corresponding value.
depositInstructions	An array of strings containing the deposit instructions
validateInstructions	An array of strings containing the instructions on how to validate a deposit. If this channel does not require validation, this may be set to null.
confirmInstructions	An array of strings containing information about how the payment for this channel is confirmed.

disclaimer	This is the standard Dragonpay disclaimer that merchants are required to display. Display of these information is mandatory when using the Advanced Control method.

Below is a sample JSON-formatted response for a GetEmailInstruction request:


```
{
"introMsg":"",
"paymentDetails":[
        ["Channel","BDO Over-the-Counter Deposit-with-Reference"],
        ["Reference No","LBUAP5W2"],
        ["Account No","1670333890"],
        ["Account Name","Dragonpay Corporation"],
        ["BDO Deposit Ref No","9766412721"],
        ["Amount","PHP 125.00"],
        ["Description","test"],
        ["Deadline","Friday, Jul 1, 2016 - 11:00 PM"]],
"depositInstructions":["Go to any BDO branch and fill-up the blue Cash
Deposit slip with the exact amount due."],
"validateInstructions":null,
"confirmInstructions":["Payments are normally processed within a few
minutes."],
"disclaimer":"Dragonpay is an independent third party payment
processor. Any terms, conditions or warranty for the product or service
that you purchase using our payment facility is strictly between you
and the merchant. Dragonpay shall not be held liable for failure of the
merchant to deliver the said product or service as advertised. You
hereby release and hold harmless Dragonpay from all liability arising
from the payment you are about to make, as all liability shall reside
with the merchant.  By using Dragonpay, you agree to be unconditionally
bound by its <a href=\"https://www.dragonpay.ph/terms-and-
conditions\">Terms of Use</a>. This email is only intended for
instructional purposes.  It is not a voucher nor receipt of a completed
payment."
}
```


The merchant is free to render the JSON content in whatever layout he prefers as long as the disclaimer (or a link to the disclaimer) is provided.

When using the background method to pass control to Dragonpay, the merchant is required to set two HTTP header variables when calling Pay.aspx. Dragonpay needs these additional details for fraud checking.

Header Variable	Description
HTTP_X_USERIP	The ip address of the actual user has to be passed in this header variable. Since merchant is programmatically invoking Pay.aspx, Dragonpay cannot see the actual IP address of the buyer unless the merchant forwards it through this header variable.
user-agent	Merchant must pass on the user-agent field that it retrieved from the user's browser and forward it to Dragonpay.

Appendix 1 – Currency Codes

Code	Description
PHP	Philippine Peso
USD	US Dollar
CAD	Canadian 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 cancelled
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
R	Refund
K	Chargeback
V	Void
A	Authorized