Dragonpay Online Payment

Financial Institution Partner API

Version 0.15 – 26 November 2010

Table of Contents

1. About this Document

This document describes the Application Programming Interface (API) between Payment Switch (PS) and the Financial Partner's Payment Gateway (PG) system. The PS is responsible for communicating with the e-commerce merchant's system for payment requests using a separate API. Upon validating the request, it redirects the end-user to his bank of choice. The information needed by the bank to process this 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 **rchiang@mozcom.com**.

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	Nov 27, 2009	Alpha Version
0.11	Jan 12, 2010	Changed terminology: Payment Gateway (PG) to Payment Switch (PS) and Financial Partner (FP) to PG
		Changed **bankid** to **procid** and varchar(4)
		Added PENDING status
0.12	May 25,2010	Removed **secretKey** from GetTxnToken
0.13	Jun 28, 2010	Made email a required parameter
0.14	Aug 9, 2010	Changed URL's to api.dragonpay.ph
		Changed billerid to varchar(80)
		Added frame=none optional parameter
0.15	Nov 26, 2010	Added **digest** to Web Service methods
		Removed **procId** from PostTxnResult

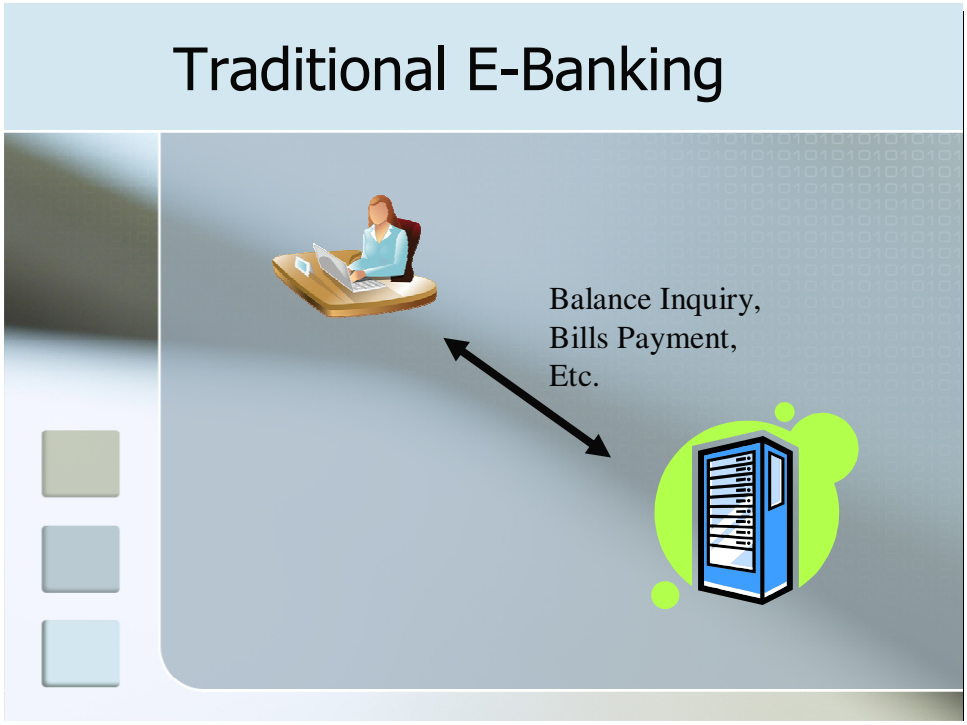
4. Introduction

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).

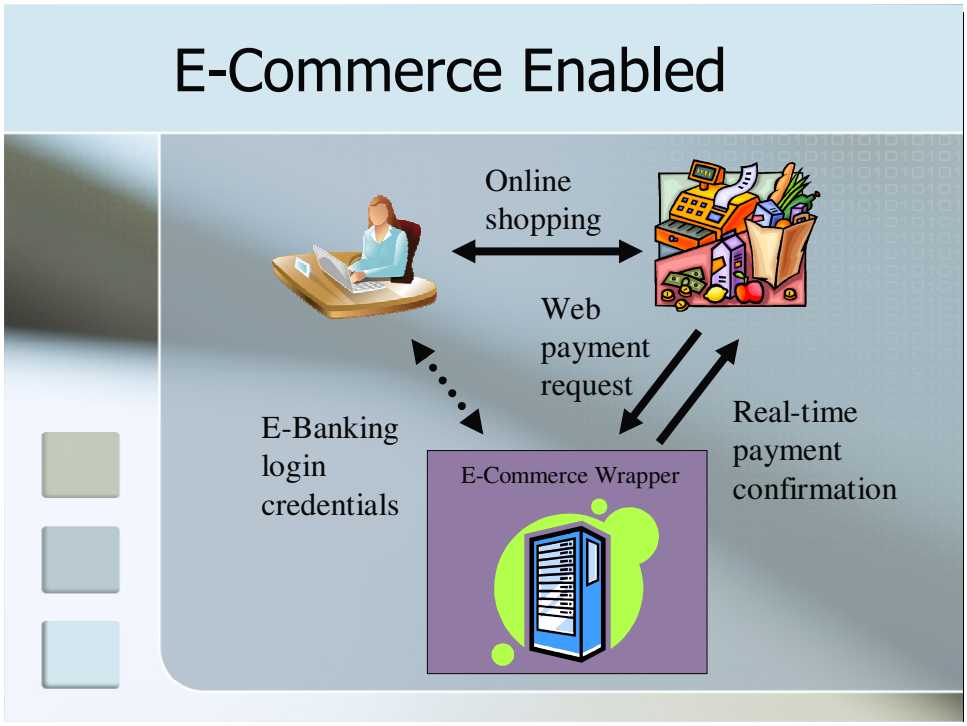


Under this scenario, the bank's system assumes that it is transacting with a live person. It responds to the requests sent by the bank customer over the browser. These requests are made by navigating through the web interface's menu system and by filling up on-screen forms.

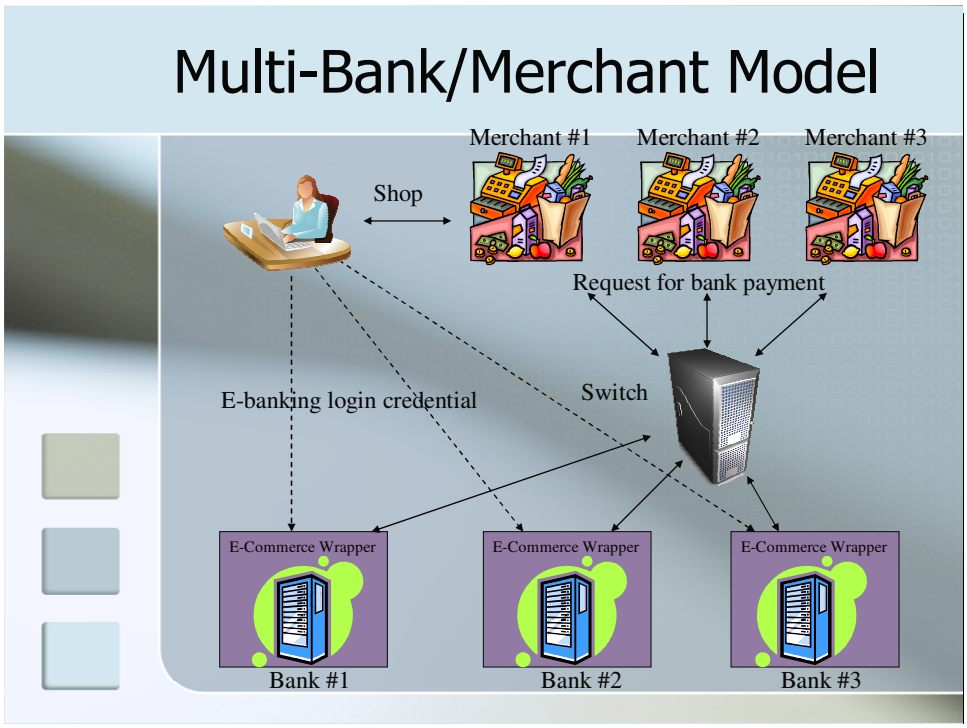
Online banking systems are normally not designed to work with e-commerce merchants or online stores which require machine-to-machine communication. They do not have the capability to accept requests programmatically from 3rd party websites or applications (ex. Shopping cart systems) for debiting the 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.

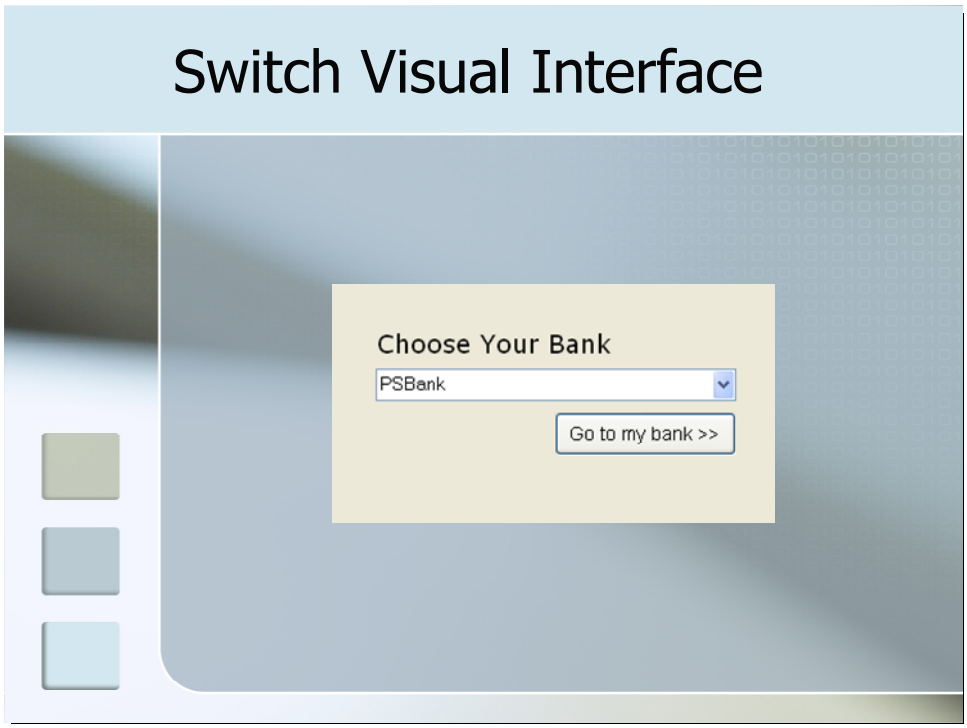
1. Customer surfs an online store
2. Customer clicks on items that he wants
3. Item is placed in an online shopping cart
4. Customer goes to *Checkout*
5. Customer is presented with several payment options
6. Customer clicks on the payment option he prefers
7. Payment processing is performed
8. Online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

This section of the document describes the bank or Financial Partner Payment Gateway (PG) 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, PG must fulfill the following prerequisites:

1. PG site can accept http request data from PS system when a customer wishes to pay an online store with his bank account.
2. PG payment site can prompt customer for his bank account and perform the necessary debit payment or funds transfer
3. PG payment site can provide real-time confirmation of payment
4. PG payment site can redirect buyer back to PS's Postback URL to fulfill payment procedure.

Each PG is assigned the following:

- proc id – unique code identifying the PG
- secret key – a unique password assigned to PG for checksum validation

Production Postback URL:

```
https://api.dragonpay.ph/Postback.aspx
```

Test Postback URL:

```
http://test.dragonpay.ph/Postback.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 (PS -> PG and PG->PS)

This section describes how the PS will pass a request to the PG 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, PS sends the request parameters using HTTP GET with a browser redirect. The PG system only needs to read and parse the GET Query String to extract all the necessary information.

The PG 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 PS.
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 PS to the PG via name-value pairs to request for a payment.

Parameter	Data Type	Description
refno	Varchar(20)	A common reference number identifying this specific transaction
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
billerid	Varchar(80)	The account details (acctno/name/type) or unique biller id to credit for this transaction
email	Varchar(40)	Email address of customer
digest	Char(40)	A sha1 checksum digest of all the parameters along with the secret key.
frame	Varchar(10)	If this optional parameter is set to *none*, display a minimal panel because the merchant is using the iframe tag to embed the payment page

An HTTP GET from PS may look something like this:

```
https://www.mybank.com.ph/Pay.aspx?refno=12345678&amount=1000.00&ccy=PHP&descri
ption=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();
}
```

To compare the integrity of the message, do the following steps:

1. Build the message string based on the parameters
2. Generate the SHA1 digest for that message string
3. Compare the generated digest with the digest that was passed and see if they match

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

```
string message = String.Format("{0}:{1}:{2}:{3}:{4}:{5}:{6}",
       Request["refno"].ToString(),
       Request["amount"].ToString(),
       Request["ccy"].ToString(),
       Request["description"].ToString(),
       Request["billerid"].ToString(),
       Request["email"].ToString(),
       Application["secretkey"].ToString());
```

Then compare against the passed digest:

```
if (GetSHA1Digest(message) != Request["digest"].ToString())
{
       // display some error message and abort processing
}
else
{
       // 1. prompt for bank id and password

       // 2. prompt for bank account to use

       // 3. perform funds transfer to billerid account
}
```

5.2.1.2 Response Parameters

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

Parameter	Description
refno	A common reference number identifying this specific transaction
procid	Unique ID assigned to this PG
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.

An HTTP GET from PG may look something like this:

```
https://api.dragonpay.ph/Postback.aspx?refno=12345678&procid=PSB&status=S&messa
ge=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(new String.Format("{0}:{1}:{2}:{3}:{4}",
       refno,
       procid,
       status,
       message,
       Application["secretkey"].ToString()));

String redirectString =
String.Format("{0}?refno={1}&procid={2}&status={3}&message={4}&digest={5}",
              postbackUrl,
              refno,
              procid,
              status,
              Server.UrlEncode(message),
              digest);

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

Note: since the *description* parameter will normally contain whitespaces or other symbols, make sure to apply the proper URL encoding/decoding before passing as parameter or computing digests.

In cases wherein the transaction *status* is PENDING, the PG may asynchronously send an HTTP GET in the future once the payment is completed. The format is similar to the normal call to the Postback URL as described above.

5.2.2 SOAP/XML Web Service Model

For greater security, the PG 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 PG and PS servers through SOAP calls.

To use this model, the flow will be as follows:

1. PS sends an HTTP GET string to PG containing a *refno* and a *digest*.
2. PG verifies the source IP of request and validates *digest*.
3. PG sends a Web Service request to PS with the method *GetTxnToken()* passing its *procid* and the *refno*. PS responds with a *tokenid* to be used for the session with this transaction.
4. PG calls method *GetTxnDetail()* using the assigned *tokenid* to extract the detailed parameters.

5. PG renders the webpage and performs payment processing
6. When processing is completed, PG calls method *PostTxnResult()* with the previously assigned *tokenid*.
7. PG redirects browser back to PS with an HTTP GET passing the *tokenid*.

The advantages of using this model are:

1. Parameters are not visible on the browser
2. PG server is the one retrieving the parameters directly from PS. Thus, any attempt by a 3rd party to fake a request will just result to an error when PG tries to confirm it with PS.
3. PG server is also the one posting the results directly back to PS

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.)

Web Service Production URL:

`https://api.dragonpay.ph/DragonPayWebService/PaymentGatewayService.asmx`

Web Service Test URL:

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

5.2.2.1 Request Parameters

The following is passed via HTTP GET during the browser redirection process.

Parameter	Description
refno	A common reference number identifying this specific transaction
digest	A SHA1 checksum digest of the refno parameter along with the secret key.
frame	If this optional parameter is set to *none*, display a minimal panel because the merchant is using the iframe tag to embed the payment page in his main website

An HTTP GET from PS may look something like this:

```
https://www.mybank.com.ph/Pay.aspx?refno=12345678&digest=a4b3d08462......
```

The PG will accept the request and check the digest like so:

```
string message = String.Format("{0}:{1}",
       Request["refno"].ToString(),
       ConfigurationManager.AppSettings["secretkey"].ToString());

if (GetSHA1Digest(message) != Request["digest"].ToString())
{
       // display some error message and abort processing
}
```

The above example assumes that the *secretkey* parameter is defined in the ASP.NET **web.config** configuration file. Upon confirmation of validity of HTTP GET request, PG should call the following Web Service to extract details for the transaction.

Web Method: GetTxnToken

Parameter	Data Type	Description
procid	Varchar(4)	Unique ID assigned to this PG
refno	Varchar(20)	A common reference number identifying this specific transaction
digest	Varchar(40)	SHA1 digest of *procid:refno:secretkey*

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 thirty (30) minutes. 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 invalid *refno*, incorrect *procid*, etc.

Web Method: GetTxnDetail

Parameter	Data Type	Description
tokenid	Varchar(40)	The id returned by *GetTxnToken*
digest	Varchar(40)	SHA1 digest of *tokenid:secretkey*

The *GetTxnDetail()* method will return an XML structure with the following fields:

Parameter	Data Type	Description
refno	Varchar(20)	A common reference number identifying this specific transaction
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 for Codes)
description	Varchar(128)	A brief description of what the payment is for
billerid	Varchar(80)	The account details (acctno/name/type) or unique biller id to credit for this transaction
email	Varchar(40)	Email address of customer
procid	Varchar(4)	A unique code assigned to PG (ex. BDO, BPI, MBTC, PSB)

A null record is returned otherwise.

```
<?xml version="1.0" encoding="UTF-8"?>
<TxnDetail>
       <refno>12345678</refno >
       <amount >1000.00</amount >
       <ccy>PHP</ccy>
       <description>Box of Chocolates</description>
       <billerid>7442001894</billerid>
       <email>rchiang@mozcom.com</email >
       <procid>PSB</procid>
</TxnDetail>
```

5.2.2.2 Response Parameters

When PG processing is completed, it should perform a *PostTxnResult()* to send the results directly to PS.

Web Method: PostTxnResult

Parameter	Data Type	Description
tokenid	Varchar(40)	The id returned by *GetTxnToken*
status	Char(1)	The result of the payment. Refer to Appendix 3 for codes.
message	Varchar(128)	If *status* is SUCCESS, this can be the PG's payment transaction reference number. If *status* is FAILURE, return one of the error codes described in Appendix 2. If *status* is PENDING, this can be the PG's reference number to complete the funding.
digest	Varchar(40)	SHA1 digest of *tokenid:status:message:secretkey*

PostTxnResult() returns one of the codes enumerated in Appendix 2.

NOTE: When calling *PostTxnResult()*, PG must make sure to check against request timeouts and perform the necessary re-attempts to post when the connection is restored. If successive re-attempts still fail, PG should generate an email to the administrator advising him of the failure. Manual intervention may be necessary at this stage to update the transaction status on PS's side.

After posting the transaction result via Web Service, PG performs an HTTP GET redirect with the following parameters:

Parameter	Data Type	Description
tokenid	Varchar(40)	The id returned by *GetTxnToken*
digest	Varchar(40)	SHA1 digest of *tokenid:secretkey*

The code may look like this:

```
String redirectString =
       String.Format("{0}?tokenid={1}&digest={2}",
              postbackUrl,
              tokenid, GetSHA1Digest(tokenid + ":" + secretKey));

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

5.3 Multi-Device Support

One of the goals of this online banking payment gateway project is to make itself accessible to as many devices as possible. Hence, it is recommended that the PG implementation should consider the type of environment or browser it is working with. It should not assume that it is only being called by a desktop web browser. As much as possible, it should determine the screen size it is working with and render its interface properly even on smaller screens (ex. PDA's, iPhone, Mobile phones).

Implementors should avoid the use of HTML tags that are not supported by majority of browsers. These include frames and nested tables. When using images, consider keeping different sizes and resolutions of the same image and display the appropriate version depending on the screen size. Do not rely on cookies to save state as PDA- or phone-based browsers may not support cookie management. Avoid opening of a new full-screen or pop-up window as these normally do not work well with non-desktop-based browsers.

The use of Cascading Stylesheets (css) is recommended as it can make HTML/XHTML pages more portable and easier to render in different devices without the need to modify the code everytime. Using client-side Javascripts, making heavy use of AJAX, embedding ActiveX controls, Java applets or Flash animation, are discouraged as these normally do not work well with lightweight browsers.

Implementation should stick to basic HTML 4.0 or XHTML Mobile Profile as much as possible for high portability.

Appendix 1 – Currency Codes

Code	Description
PHP	Philippine Peso
USD	US Dollar

Appendix 2 – Error Codes

Code	Description
000	Success
101	Invalid PG 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	Operation Error

Appendix 3 – Status Codes

Code	Description
S	Success
F	Failure
P	Pending
U	Unknown
R	Refund
K	Chargeback