**How to do PayPal Integration in PHP?**

**(Step by Step Tutorial)**

* [By Faizan Fahim](https://serverguy.com/author/faizan/)
* |
* January 7, 2023

This is a tutorial for PayPal integration in PHP.

If you are using a [PHP based eCommerce platform](https://serverguy.com/ecommerce/php-open-source-ecommerce-platforms/), then you can follow this tutorial to implement PayPal payment on your store.

* [What is PayPal?](https://serverguy.com/learn/paypal-integration-in-php/#paypal)
* [How PayPal Works?](https://serverguy.com/learn/paypal-integration-in-php/#paypalworks)
* [Lean Basic Terms](https://serverguy.com/learn/paypal-integration-in-php/#basic)
* [PayPal Integration in PHP Step by Step](https://serverguy.com/learn/paypal-integration-in-php/#php)
* [Testing](https://serverguy.com/learn/paypal-integration-in-php/#tets)

**What is PayPal?**

PayPal is a popular [payment gateway](https://serverguy.com/magento/best-payment-gateways/) that offers secure and easy to use payment processing globally.

A customer can add his bank account, debit card, or credit card and start with PayPal. Currently, 286 Million users are using it, and nearly 87.5% of online buyers make the payment with PayPal.

So having PayPal on your site is one of the suggestion we give to our clients..

Best features that PayPal offers include:

* Credit Cards, Debit Cards
* Create an online invoice
* Buyer and seller accounts
* Cart
* Mobile Application
* International money transfer

**How PayPal Works?**

Using PayPal for transactions and online shopping is secure. If the user already has an account on PayPal, then he can quickly checkout.

He has to login to the account and then confirm the amount. After selecting the bank/card, the payment will be made in one click.



It is easy for the customer.

The merchant has to set up PayPal in the store. And if it is a software product, and requires recurring payments, you have to do the PayPal integration in PHP.

In this tutorial, I will tell you how to do PayPal Integration in PHP, and in the end, you will be able to integrate PayPal into your store.

**Basic Terms**

Here are a few basic terms you need to know before you read the tutorial.

**Instant Notification Process:** Instant Payment Notification (IPN) is a notification service that automatically sends the notification of any event related to PayPal transactions to merchants. eCommerce store owners can use it in many ways, such as fulfilling orders automatically and providing customers with order status.

**Payment Form:** A form on your site will send the values such as customer name/email address to PayPal.

**payment.php page:** The payment.php page on your website is that page that will handle all the outgoing requests to the PayPal, and the incoming request coming from PayPal.

**PayPal Integration in PHP Step by Step**

**STEP 1 – Setup PayPal Account**

The first step is to sign up for a PayPal account. Go to PayPal and click on SignUp.



You must open the business account to be able to access IPN. After setting up the PayPal account properly, you would be able to access IPN.

Go to “edit profile” option of your PayPal account and look for the following settings:

Navigate to the “My Selling Preferences” and find the Getting Paid and Managing Risk.

* Open the Instant payment Notification Process
* Turn ON the IPN Value
* Change the URL of IPN to the PHP page containing the IPN code (http://www.example.com/payment.php)

Navigate to the “My Selling Preferences” and find the Getting Paid and Managing Risk.

* Open the Block Payments
* Turn off the payment from the users who pay with eCheque by blocking it

Navigate to the ‘account information’ and find ’email’

* Enter your primary email address. This email will be visible to users, so keep it professional.

**STEP 2 – Creating HTML Form**

You need to set a form on your site to send the basic information about the payee to PayPal.

You can use this HTML code to create the HTML form.

<!DOCTYPE html>

<html>

<head>

 <meta charset="utf-8">

 <meta http-equiv="X-UA-Compatible" content="IE=edge">

 <meta name="viewport" content="width=device-width, initial-scale=1">

 <title>Paypal Integration Test</title>

</head>

<body>

 <form class="paypal" action="payments.php" method="post" id="paypal\_form">

 <input type="hidden" name="cmd" value="\_xclick" />

 <input type="hidden" name="no\_note" value="1" />

 <input type="hidden" name="lc" value="UK" />

 <input type="hidden" name="bn" value="PP-BuyNowBF:btn\_buynow\_LG.gif:NonHostedGuest" />

 <input type="hidden" name="first\_name" value="Customer's First Name" />

 <input type="hidden" name="last\_name" value="Customer's Last Name" />

 <input type="hidden" name="payer\_email" value="customer@example.com" />

 <input type="hidden" name="item\_number" value="123456" / >

 <input type="submit" name="submit" value="Submit Payment"/>

 </form>

</body>

</html>

More sensitive information like price, business name, etc., will be sent in the next step.

**STEP 3 – Sending Request**

As I mentioned, the payment.php page will handle the outgoing request to the PayPal, and the incoming response after the processing for the Payment.

However, you have to build up the parameters and pass them through PayPal via query strings before sending them.

* Values that need to be sent:
* business: email address of your PayPal account
* Item\_name: name of the item
* Amount: the price of the product
* Return: return address after payment success
* Cancel\_return: if payment is canceled, the return address
* Notify\_url: payment.php page URL on your site
* Custom: any other data you want to send via PayPal request

 // For test payments we want to enable the sandbox mode. If you want to put live

// payments through then this setting needs changing to `false`.

$enableSandbox = true;

// Database settings. Change these for your database configuration.

$dbConfig = [

 'host' => 'localhost',

 'username' => 'user',

 'password' => 'secret',

 'name' => 'example\_database'

];

// PayPal settings. Change these to your account details and the relevant URLs

// for your site.

$paypalConfig = [

 'email' => 'user@example.com',

 'return\_url' => 'http://example.com/payment-successful.html',

 'cancel\_url' => 'http://example.com/payment-cancelled.html',

 'notify\_url' => 'http://example.com/payments.php'

];

$paypalUrl = $enableSandbox ? 'https://www.sandbox.paypal.com/cgi-bin/webscr' : 'https://www.paypal.com/cgi-bin/webscr';

// Product being purchased.

$itemName = 'Test Item';

$itemAmount = 5.00;

// Include Functions

require 'functions.php';

// Check if paypal request or response

if (!isset($\_POST["txn\_id"]) && !isset($\_POST["txn\_type"])) {

 // Grab the post data so that we can set up the query string for PayPal.

 // Ideally we'd use a whitelist here to check nothing is being injected into

 // our post data.

 $data = [];

 foreach ($\_POST as $key => $value) {

 $data[$key] = stripslashes($value);

 }

 // Set the PayPal account.

 $data['business'] = $paypalConfig['email'];

 // Set the PayPal return addresses.

 $data['return'] = stripslashes($paypalConfig['return\_url']);

 $data['cancel\_return'] = stripslashes($paypalConfig['cancel\_url']);

 $data['notify\_url'] = stripslashes($paypalConfig['notify\_url']);

 // Set the details about the product being purchased, including the amount

 // and currency so that these aren't overridden by the form data.

 $data['item\_name'] = $itemName;

 $data['amount'] = $itemAmount;

 $data['currency\_code'] = 'GBP';

 // Add any custom fields for the query string.

 //$data['custom'] = USERID;

 // Build the query string from the data.

 $queryString = http\_build\_query($data);

 // Redirect to paypal IPN

 header('location:' . $paypalUrl . '?' . $queryString);

 exit();

} else {

 // Handle the PayPal response.

}

**STEP 4 – Incoming Response**

Now the PayPal will send the response, and the payment.php page we made will read the response.

We have to add this code to the else statement of our payment.php script. It will check out if we have already processed this transaction before adding the payment to the database.

// Handle the PayPal response.

// Create a connection to the database.

$db = new mysqli($dbConfig['host'], $dbConfig['username'], $dbConfig['password'], $dbConfig['name']);

// Assign posted variables to local data array.

$data = [

 'item\_name' => $\_POST['item\_name'],

 'item\_number' => $\_POST['item\_number'],

 'payment\_status' => $\_POST['payment\_status'],

 'payment\_amount' => $\_POST['mc\_gross'],

 'payment\_currency' => $\_POST['mc\_currency'],

 'txn\_id' => $\_POST['txn\_id'],

 'receiver\_email' => $\_POST['receiver\_email'],

 'payer\_email' => $\_POST['payer\_email'],

 'custom' => $\_POST['custom'],

];

// We need to verify the transaction comes from PayPal and check we've not

// already processed the transaction before adding the payment to our

// database.

if (verifyTransaction($\_POST) && checkTxnid($data['txn\_id'])) {

 if (addPayment($data) !== false) {

 // Payment successfully added.

 }

}

verifyTransaction function will verify the authenticity of the response. After calling it, the function will take the data received from the PayPal, and will validate it by making the curl request to PayPal with the transaction data received.

If the response is the same, we will get the VERIFIED message, and we will know that everything is OK. Then the payment will be added to the database.

The verifyTansaction function:

function verifyTransaction($data) {

 global $paypalUrl;

 $req = 'cmd=\_notify-validate';

 foreach ($data as $key => $value) {

 $value = urlencode(stripslashes($value));

 $value = preg\_replace('/(.\*[^%^0^D])(%0A)(.\*)/i', '${1}%0D%0A${3}', $value); // IPN fix

 $req .= "&$key=$value";

 }

 $ch = curl\_init($paypalUrl);

 curl\_setopt($ch, CURLOPT\_HTTP\_VERSION, CURL\_HTTP\_VERSION\_1\_1);

 curl\_setopt($ch, CURLOPT\_POST, 1);

 curl\_setopt($ch, CURLOPT\_RETURNTRANSFER, 1);

 curl\_setopt($ch, CURLOPT\_POSTFIELDS, $req);

 curl\_setopt($ch, CURLOPT\_SSLVERSION, 6);

 curl\_setopt($ch, CURLOPT\_SSL\_VERIFYPEER, 1);

 curl\_setopt($ch, CURLOPT\_SSL\_VERIFYHOST, 2);

 curl\_setopt($ch, CURLOPT\_FORBID\_REUSE, 1);

 curl\_setopt($ch, CURLOPT\_CONNECTTIMEOUT, 30);

 curl\_setopt($ch, CURLOPT\_HTTPHEADER, array('Connection: Close'));

 $res = curl\_exec($ch);

 if (!$res) {

 $errno = curl\_errno($ch);

 $errstr = curl\_error($ch);

 curl\_close($ch);

 throw new Exception("cURL error: [$errno] $errstr");

 }

 $info = curl\_getinfo($ch);

 // Check the http response

 $httpCode = $info['http\_code'];

 if ($httpCode != 200) {

 throw new Exception("PayPal responded with http code $httpCode");

 }

 curl\_close($ch);

 return $res === 'VERIFIED';

}

We can also call to checkTxnid. The function’s purpose is simple; it checks if the txn\_id value from PayPal already exists in our database.

After verifying the payment, it is a good idea to call this function and check if you have already added the transaction to the database.

Txn\_id function:

function checkTxnid($txnid) {

 global $db;

 $txnid = $db->real\_escape\_string($txnid);

 $results = $db->query('SELECT \* FROM `payments` WHERE txnid = \'' . $txnid . '\'');

 return ! $results->num\_rows;

}

Here, you can add as many checks as you like. This is the opportunity to verify the various aspects of the payment.

**STEP 5 – Adding Payment**

The final step after verifying the payment, doing the additional checkup is adding the payment to the [database](https://serverguy.com/servers/how-to-backup-mysql-database/).

We will create a payment table to store the payment details in the database system.

Here is the code to create the payments table in MySQL:

CREATE TABLE IF NOT EXISTS `payments` (

 `id` int(6) NOT NULL AUTO\_INCREMENT,

 `txnid` varchar(20) NOT NULL,

 `payment\_amount` decimal(7,2) NOT NULL,

 `payment\_status` varchar(25) NOT NULL,

 `itemid` varchar(25) NOT NULL,

 `createdtime` datetime NOT NULL,

 PRIMARY KEY (`id`)

 ) ENGINE=MyISAM DEFAULT CHARSET=utf8 AUTO\_INCREMENT=1 ;

Then we have to set the function to call the addPayment.

Here is the function to do this task:

function addPayment($data) {

 global $db;

 if (is\_array($data)) {

 $stmt = $db->prepare('INSERT INTO `payments` (txnid, payment\_amount, payment\_status, itemid, createdtime) VALUES(?, ?, ?, ?, ?)');

 $stmt->bind\_param(

 'sdsss',

 $data['txn\_id'],

 $data['payment\_amount'],

 $data['payment\_status'],

 $data['item\_number'],

 date('Y-m-d H:i:s')

 );

 $stmt->execute();

 $stmt->close();

 return $db->insert\_id;

 }

 return false;

}

**Testing**

Lastly, we will check if the integration is done correctly and working correctly.

[PayPal Sandbox](https://sandbox.paypal.com/) is similar to PayPal and provides all the functionality of PayPal. But instead of using real properties, you can use fake properties in Sandbox.

You can create fake accounts of buyers and sellers. Set the development site, and the test the PayPal integration through the stage.

Creating Sandbox accounts is free, and it can be done from the PayPal Developer website.

The code provided in the post is optimized for the Sandbox. The URL in the request is www.sandbox.paypal.com.

You have to change the value of $enableSandbox from true to false. This will update the PayPal URLs in the coding.

**Also Read:**

* [How to Disable PHP Execution in WordPress Directories?](https://serverguy.com/wordpress/how-to-disable-php-execution-in-wordpress-directories/)
* [Best PHP Editors and PHP IDE for the Development](https://serverguy.com/comparison/best-php-editor-php-ide/)
* [Top 12 PHP Based Open Source Ecommerce Platform](https://serverguy.com/ecommerce/php-open-source-ecommerce-platforms/)

**Final Words**

The process is tricky if you do not know coding and php. But for the developers, it is not that complicated.

However, if you are on WordPress or Magento, you should use a plugin or extension.

In this tutorial, you learned about how you can do PayPal integration in PHP step by step. This is the basic guide.

Still, if there are any doubts or issues, you can leave it in the comments section.

\*you can find the source code on this [GitHub](https://github.com/EvolutedNewMedia)

Bas du formulaire