BLOG

HOW TO SET UP PAYPAL INTEGRATION WITH PHP & MYSQL

14 FEBRUARY, 2011

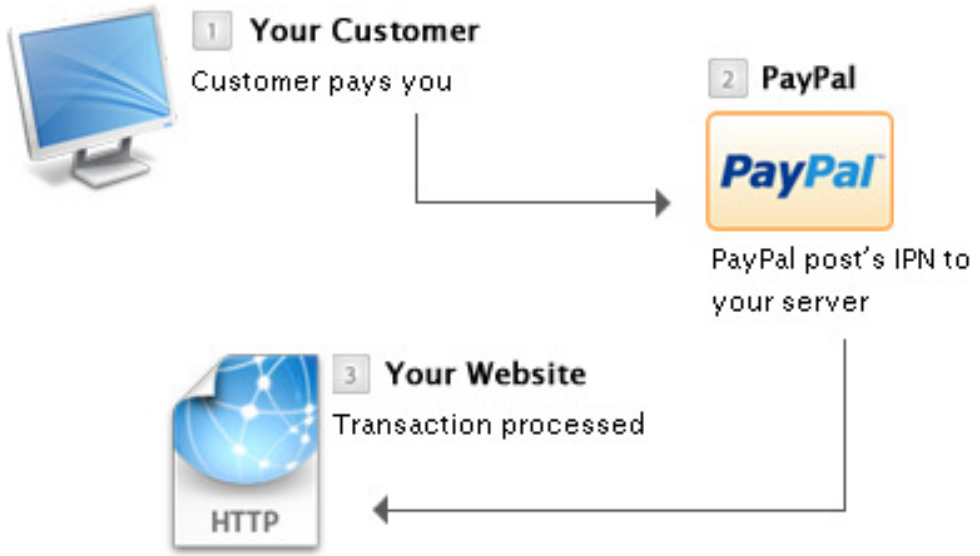
This post was last updated on Thursday 26th July 2018

PayPal is the most popular payment service on the web so being able to integrate your website with PayPal's Instant Payment Notification Service (IPN) is essential if you need to process payments through your website.

There are 3 main parts to the PayPal IPN system:

1. A webpage that initiates a request to PayPal to make a payment.
2. A PHP page on your webserver that PayPal calls to notify you that payment has been made.
3. A webpage that confirms the above payment and continues on to the next phase of your web application, such as a 'Thank You' page.

Parts 1 and 3 are accessible by customers on your website. Part 2 is only visible to PayPal. The diagram below illustrates the interaction between your customer, PayPal and your website.



The following steps break down each part of the process into easy to follow chunks, it is assumed that you have knowledge of PHP and MySQL.

NOTE

- If you are not receiving the correct response from Paypal ensure that you are using the main test account (Verified Business Account) from your Paypal Sandbox account.
- Also ensure that you are testing the Paypal IPN Script on an online webserver (Not MAMP, Xampp etc..) as Paypal requires a reachable 'return url', 'cancel url' and 'notify url'.

STEP 1 – SETUP PAYPAL ACCOUNT

Sign up for a PayPal account if you don't already have one. In order to use IPN, the Paypal account you are selling from **must** be a Business Account.

Once you have a registered PayPal account your account must be setup correctly to use IPN. Select 'edit profile' from your PayPal account and check the following settings.

- Under 'My Selling Preferences' >> 'Getting paid and managing risk' >> 'Instant Payment Notification Preferences'
  - Set the IPN value to 'On'
  - Set the IPN URL to the PHP page containing the IPN code shown in steps 3 & 4 of this tutorial. (http://www.example.com/payment.php)

- Under 'My Selling Preferences' >> 'Getting paid and managing risk' >> 'Block payments'

- Block payments from users who pay with eCheque. (This is because these will not be instant payments)

- Under 'account information' >> 'email'
  - Note down your primary email address. This email will be visible to users so make it a professional one. User's may feel apprehensive about sending money to an e-mail address with the domain 'hotmail.com' or 'Yahoo.com' etc...

STEP 2 – SIMPLE HTML FORM

Your website must now send all the required values to PayPal so that the payment can be processed.

The following code example demonstrates a basic form that we will use to send the values:

```
<!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_f
        <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.g
        <input type="hidden" name="first_name" value="Customer's First Na
        <input type="hidden" name="last_name" value="Customer's Last Name
        <input type="hidden" name="payer_email" value="customer@example.c
        <input type="hidden" name="item_number" value="123456" / >
        <input type="submit" name="submit" value="Submit Payment"/>
    </form>

</body>
</html>
```

The business name, price, submit type, notify URL and other sensitive values will be sent during the next step.

A full list of the values to send can be found at the PayPal website under the title "**A Sample IPN Message and Response**".

STEP 3 – THE REQUEST

The payment.php page will be used to handle the outgoing request to PayPal and also to handle the incoming response after the payment has been processed.

To make a request for payment we need to first build up the parameters and pass these to PayPal via the query string.

We need to pass the following values:

- business - the email address of your PayPal account
- item_name - the name of the item being purchased
- amount - the price of the item
- return - the address to return to after a successful payment
- cancel_return - the address to return to after a cancelled payment
- notify_url - the address of the payments.php page on your website
- custom - any other data to be sent and returned with the PayPal request

```
// For test payments we want to enable the sandbox mode. If you want to pu
// payments through then this setting needs changing to `false`.
$enableSandbox = true;
```

```php
// 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
// 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/webs

// 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"])) {
```

```php
// Grab the post data so that we can set up the query string for PayP
// Ideally we'd use a whitelist here to check nothing is being injecte
// 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 a
// 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.
}
```

To construct the query string we assign the post data to an array that we then push some additional values to that we don't want to be altered by the post data. This way we can ensure that a user cannot manipulate the amount being paid or any other details that may be vulnerable. We then use http_build_query to convert the array to a query string and pass this to PayPal via the header.

STEP 4 - THE RESPONSE

We now want to handle the response from PayPal, this is the callback PayPal makes to our notify URL we configured earlier. We reassign the post response to a local variable and then verify the transaction is authentic and check we've not already processed this transaction before adding the payment to our database.

To do all this we want to add the following code to the else statement of our payments.php script.

```php
// Handle the PayPal response.

// Create a connection to the database.
$db = new mysqli($dbConfig['host'], $dbConfig['username'], $dbConfig['pas

// 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 no
// 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.
    }
}
```

To verify the authenticity of the response we call the function verifyTransaction. This will take the post data received from PayPal and validate this by making a curl request to PayPal with the transaction data received. If we get back the response VERIFIED then we know that everything is OK and can proceed to add the payment to our database.

The verifyTransaction function looks like this (it can be found in our functions.php file).

```
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}
        $req .= "&$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';
}
```

Once the transaction has been verified and before we add it to our database it is a good idea to check that we've not already processed it. That's what our call

to checkTxnid is going to do. This simply checks that the txn_id value from PayPal does not already exist in our database.

```
function checkTxnid($txnid) {
    global $db;

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

    return ! $results->num_rows;
}
```

This is also a good opportunity for you to add any additional checks you might want to put in place before accepting the payment on your site. For example, you might want to check the amount paid tallies with the amount you were charging.

STEP 5 - ADD THE PAYMENT

With the response from PayPal verified and any additional checks we want to make at our end complete it's time to add the payment to our database.

To store payment details in our database a payments table must be created. The following MySQL will create the payments table used in this example code.

```
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 in our code we're calling addPayment to insert a payment into the database. This function looks like this:-

```
function addPayment($data) {
    global $db;

    if (is_array($data)) {
        $stmt = $db->prepare('INSERT INTO `payments` (txnid, payment_amou
        $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;
}
```

SANDBOX TESTING / GOING LIVE

PayPal Sandbox offers all of the functionality of PayPal, but uses "Fake accounts" created by the developer within the sandbox. You can create fake buyer and fake seller profiles, then test your PayPal integration through your development website. Sandbox accounts can be created on the PayPal Developer website for free.

When testing with the sandbox we use the www.sandbox.paypal.com URL. For live payments we need to use www.paypal.com. Our example code is configured to use the sandbox, but you can switch it to live payments by changing $enableSandbox to false, this will update the URLs used to communicate with PayPal.

Once the payment process is ready to be used by real customers you will need to disable the sandbox mode in the code.

That's it; you're ready to start taking payments online through your website.

PAYPAL INTEGRATION - SOURCE FILES

You can find the complete code for this integration on GitHub.

UP NEXT...

FIVE WAYS NOT TO WRITE FUN MICROCOPY »

11 FEBRUARY, 2011

LIKE THIS? SIGN UP FOR REGULAR UPDATES!

* Name

* Email

Your email address will only be used to send you Evoluted's monthly newsletter.

SUBSCRIBE NOW! »

76 COMMENTS

- «
- 1
- 2
- 3
- 4
- 5
- »

LEAVE A COMMENT

* Name

* Email

Website

* Body

POST COMMENT

RECENT AWARDS & ACCOLADES

LIKE WHAT YOU SEE? GET IN TOUCH

REQUEST A QUOTE »

INFO@EVOLUTED.NET 0114 272 9753

SEE WHAT WE'RE UP TO

ABOUT EVOLUTED

Evoluted is an award-winning digital agency based in Sheffield, South Yorkshire. We specialise in bespoke web design & development, digital design and ROI-focused digital marketing.

With a focus on building long-term relationships with our clients, we deliver services including SEO, PPC, web design, bespoke content management systems, e-commerce websites and email marketing. Our

WEB DESIGN & MARKETING SERVICES

- Content Management Systems »
- Content Marketing »
- Conversion Rate Optimisation »
- Copywriting »
- Email Marketing »
- Front End Development »
- PPC »
- Search Engine Optimisation (SEO) »
- SEO Audits »
- Social Media »
- Video Marketing »
- White Label Websites »

extensive base of clients includes large UK and international organisations.

Privacy Policy