

## Writing A Simple Twitter Client Using the PHP Zend Framework's OAuth Library (Zend\_Oauth)

During yesterday, I finally got around to patching and finishing Zend\_Oauth's Consumer implementation for the OAuth Core 1.0 Revision A specification. Once I had it finished, I used it to write a quick and simple interface to post some Tweets on [Twitter](#) while I was testing it out. With some documentation and a few extra unit tests, the Consumer implementation should find its way into Zend Framework 1.10...along with the Server implementation I think. In this article I'll explore how to write a quick Twitter client so you can post tweets (those short messages of less than 140 characters) once authorised across the OAuth protocol.

You can download all the necessary files (just be sure to edit them as described) or pull them from git from: <http://github.com/padraic/Tweet-Lite/tree/master>

## What is OAuth?

If you're not aware of what [OAuth](#) is, the OAuth specification puts it this way:

The OAuth protocol enables websites or applications (Consumers) to access Protected Resources from a web service (Service Provider) via an API, without requiring Users to disclose their Service Provider credentials to the Consumers. More generally, OAuth creates a freely-implementable and generic methodology for API authentication.

In other words, it's a means of allowing websites to access your data on other services via a service API, like Twitter's API or Google Gdata, without actually providing those websites with your username and password. Instead, OAuth allows you to authorise such websites to access your data so that they don't need your username or password - they just use an Access Token supplied by your service provider - and you can easily deauthorise them if desired. The benefit is immediately obvious - your username and password are not shared or handed out to potentially untrustworthy sites. The glut of services using Twitter are a prime example - until recently they all needed your Twitter username and password and honestly, how would you know they wouldn't misuse that? Because they said so? OAuth eliminates this problem.

The protocol works like this. The website (consumer) that wants to access your data from a service provider, contacts the provider using HTTP to retrieve a Unauthorised Request Token. The consumer will then redirect you, the user, back to your service provider so you can authorise the consumer's access. The redirect URL will contain the Unauthorised Request Token as a parameter. If you approve the access, you are redirected back to the original website with a verification code attached to the URL. The website now knows you approved its access, so it contacts the service provider, including both the newly approved Request Token (once again) and the verification code in the URL. The response to this should be a fully authorised Access Token (associated with the User) which the consumer can use in all future requests when accessing your data (until either it times out or you deauthorise the access). The Request Access token can be discarded now - in OAuth parlance you exchanged an unauthorised Request Token for an authorised Access Token.

## Preparations Are Always Inevitable!

With this understanding in hand, let's get to writing this small Twitter client as an example! You will need to download the Zend Framework (whether the latest release or via subversion). You will also need to download/checkout the Zend Framework Incubator since Zend\_Oauth is not yet part of the main trunk. Once you have stored both somewhere, make a note of the paths to their "library" directories so you can add them to the PHP include\_path later.

On the Twitter side there are three steps.

First, get a Twitter account! Hopefully you already have one and are following @padraicb (i.e. me!).

Second, you need to configure your operating system for a new local domain. Under Linux, this is done by editing /etc/hosts. You'll need root privileges here so use "sudo myeditor /etc/hosts". Under Windows, the same file is located at C:\Windows\System32\drivers\etc\hosts and will require Administrator privileges to edit. Adding a local domain is dead simple, and it's needed to use Twitter's API on a local machine - Twitter refuse all requests from localhost or 127.0.0.1 but they won't filter out a local domain name since it's unfeasible and would make a lot of application developers extremely crazy. Edit the file to include a new entry like:

```
127.0.0.1 mytwitterclient.tld
```

Once saved, your browser should immediately respond to any address in the form of http://mytwitterclient.tld by attempting to call localhost/127.0.0.1 on your system. If you have a default web document root configured with a web server running, this is where the request will be mapped to and where you should store any files. If you are really troublesome and can't take how easy that was, you can run off to play with Virtual Host configurations to use a different path.

Third! OAuth providers like Twitter don't hand out access tokens to every Tom, Dick and Harry. Well, they do - but they like to know whether your name is Tom, Dick or Harry! You need to register all applications with them for this purpose and to get hold of a key to access their OAuth service. This is pretty much standard for all similar providers. Visiting [http://twitter.com/oauth\\_clients](http://twitter.com/oauth_clients) and logging in using your Twitter account opens up a menu where you can register new applications/clients or delete them. Create a new client record here. Naming is not important - just make absolutely sure that a) you select "Browser" as the Application Type, b) set a callback URL of http://mytwitterclient.tld/callback.php (edit domain and path for your preferred location), and c) select "Read & Write" as the Default Access since we will be sending new Tweets and need that write access. We will not be using Twitter for logins. Once registered - it's immediate - you'll be faced with a page of details including URLs to various OAuth endpoints, and most importantly, the Consumer Key and Consumer Secret you should use for your application. Don't worry! You can revisit this page at any time.

## A Glimmer Of Actual Source Code

Since we're sticking with the concept of "simple" here, this will be a scripted effort not an exercise in writing the next Zend Framework powered super app. Any permanent (these can be refreshed indefinitely so don't worry about losing them) Access Token will be stored to the current session for reuse so we can skip a database.

Our mini application is comprised of six files (none of them big): config.php, common.php, index.php, tweet.php, callback.php and clear.php. Splitting this out across a few files makes it easier to follow, understand and edit. Let's start with config.php which contains our OAuth configuration which you will need to edit for your own details.



```

require_once 'Zend/Oauth/Consumer.php';
// Start up the ol' session engine
session_start();
// Include the configuration data for our OAuth Client (array $configuration)
include_once './config.php';
// Instantiate an instance of the Consumer for use
$consumer = new Zend_Oauth_Consumer($configuration);

```

The common file is pretty simple stuff. You'll note we set the include path here (you can do it in php.ini either), start our session, and instantiate a new OAuth Consumer. Zend\_Oauth currently offers a full Consumer, the Server/Provider implementation will follow in the very near future.

Time to look at the starting point to our little app, index.php:

```

<?php
// include some common code
include_once './common.php';
// Do we already have a valid Access Token or need to go get one?
if (!isset($_SESSION['TWITTER_ACCESS_TOKEN'])) {
    // Guess we need to go get one!
    $token = $consumer->getRequestToken();
    $_SESSION['TWITTER_REQUEST_TOKEN'] = serialize($token);
    // Now redirect user to Twitter site so they can log in and
    // approve our access
    $consumer->redirect();
}
// Got past that if block! Must have an Access Token. Let's kick out a simple
// form for the user to submit their tweet with.
// echo the xml declaration in case shorty tags enabled - grrr. They're
// a bloody nuisance at times.
echo '<?xml version="1.0" encoding="UTF-8"?>';
?><!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.1//EN"
"http://www.w3.org/TR/xhtml11/DTD/xhtml11.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en" >
<head>
<title>Tweet Lite Script</title>
<script language="javascript" type="text/javascript">
<!--
function imposemax(Object)
{
    return (Object.value.length <= 140);
}
-->
</script>
</head>
<body>
<?php if (isset($_GET['result']) && $_GET['result'] == 'true'): ?>
    <p style="background-color: lightgreen;">You successfully sent your tweet!</p>
<?php elseif (isset($_GET['result']) && !empty($_GET['result'])): ?>
    <p style="background-color: red;">Oops! Tweet wasn't accepted by Twitter. Probable
failure:</p>
    <div style="background-color: red;"><?php echo $_GET['result']; ?></div>

```

```

<?php endif; ?>
<p>All that work on Zend_Oauth, and all you do is send Tweets with it? <br/><br/></p>
<form action="tweet.php" method="post" id="statusform">
  <p>What do you want to say to Twitterland using 140 characters?</p>
  <!-- Bit of a JS hack without dumping in more code to do it right,
  to impose 140 char limit. It'll prevent deleting when limit reached -->
  <textarea name="status" id="status" rows="2" cols="70%" onkeypress="return
imposemax(this);"></textarea>
  <br/><input type="submit" id="submit" value="Tweet!"/>
</form>
<p><br/><br/>Click below to delete the Access Token and force start another authorisation
leg:<br/>
<a href="clear.php">Clear Access Token</a></p>
</body>
</html>

```

In the top half of this script, we're checking to see if we previously stored an Access Token (needed to use the Twitter API on behalf of the current user) as a serialized session variable. If it's there, we'll print the form, accept a status textfield, and send it to tweet.php. If we don't have an Access Token, we need to get one. Calling "\$consumer->getRequestToken();", asks the Zend\_Oauth\_Consumer to fire a request to Twitter asking for a new Unauthorised Request Token, and we should get one back. With this in hand, we may then redirect the user to Twitter (the redirect URI will include the token) so they can approve our access. Before we redirect the user, we'll serialise the token and store it as a session variable. Note that all tokens in Zend\_Oauth are objects of type Zend\_Oauth\_Token.

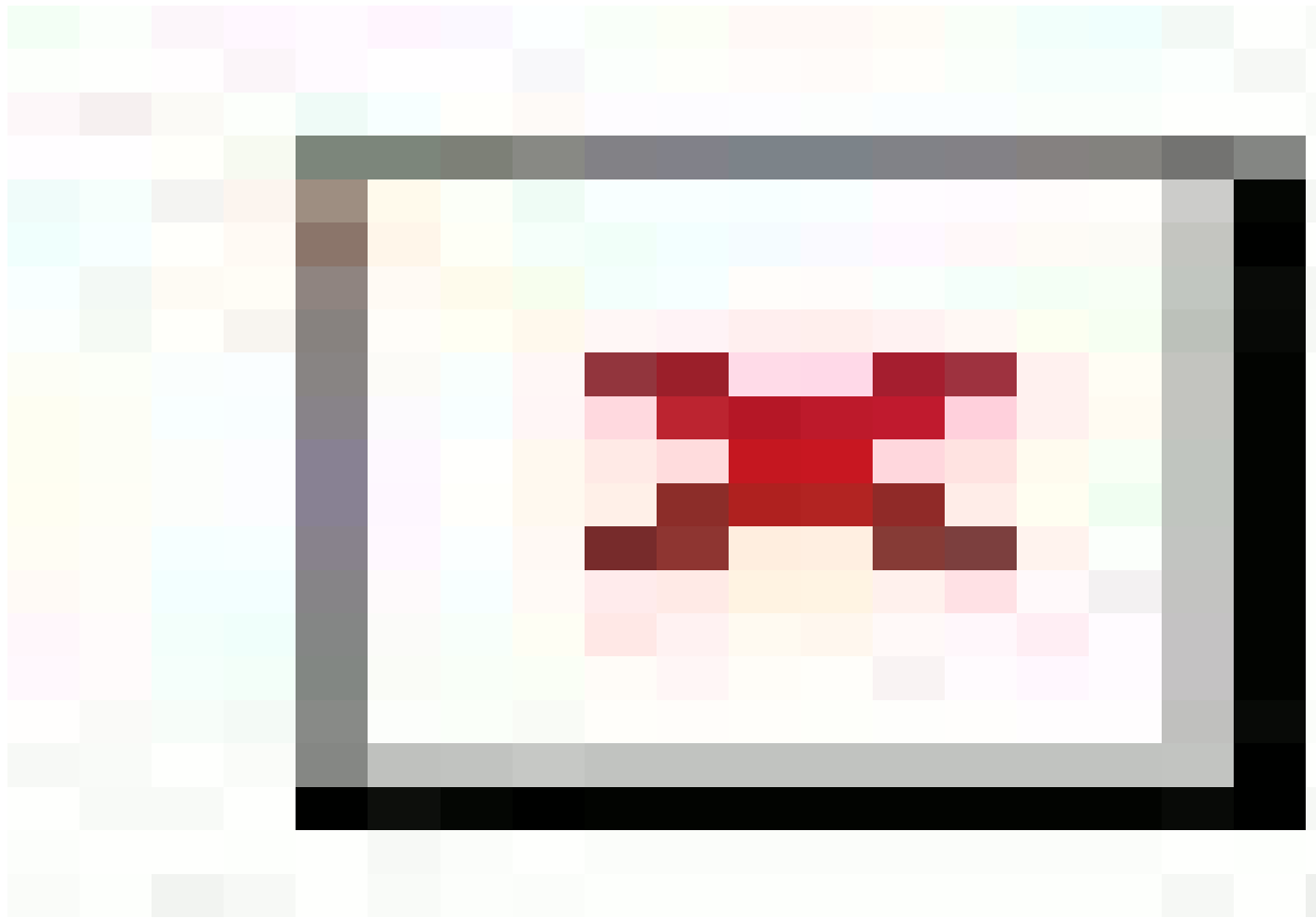
The next step happens outside the application since we redirected the user back to the provider.

Here, a user is given the option to approve our access or deny it. As Twitter notes, users may also retract their authorisation at any time. Once the user clicks the big "Allow" button, Twitter will redirect the user to us. Within the redirect URI should be a verification code (think of it like a PIN number) with which we can now request an authorised Access Token. Now, as we configured, and as it was added during registration, a user should be redirected to callback.php:

```
<?php
```

```
// include some common code
include_once './common.php';
// Someone's knocking at the door using the Callback URL - if they have
// some GET data, it might mean that someone's just approved OAuth access
// to their account, so we better exchange our current Request Token
// for a newly authorised Access Token. There is an outstanding Request Token
// to exchange, right?
if (!empty($_GET) && isset($_SESSION['TWITTER_REQUEST_TOKEN'])) {
    $token = $consumer->getAccessToken($_GET, unserialize($_SESSION[
'TWITTER_REQUEST_TOKEN']));
    $_SESSION['TWITTER_ACCESS_TOKEN'] = serialize($token);
    // Now that we have an Access Token, we can discard the Request Token
    // Keep on eye on gathering RTs in real life which are never used.
    $_SESSION['TWITTER_REQUEST_TOKEN'] = null;
    // With Access Token in hand, let's try accessing the client again
    header('Location: ' . URL_ROOT . '/index.php');
} else {
    // Mistaken request? Some malfeasant trying something?
    exit('Invalid callback request. Oops. Sorry.');
```

In the callback file, we check the incoming request for a query string and if we also have an outstanding Request Token, we pass the \$\_GET array and the Request Token (unserialised from the session) to the Zend\_Oauth\_Consumer::getAccessToken() method. This sets up another request from the application to Twitter which includes both the Request Token and the verification (PIN) code from the user redirect. Twitter should now respond with an authorised Access Token, which we'll serialise to a session variable for future use (in a serious app, this would be associated with the user account more permanently), and with an Access Token in hand we can discard the now useless Request Token. With this done, we can redirect the user back to index.php where they will be greeted by our application in all it's glory (or not).



Now that we are here, our OAuth authorisation has been completed. But wait, there's more!

Users may now input a status message of no more than 140 characters and submit it to Twitter. The form on the page is sent to `tweet.php`, so let's see what it is doing:

```
<?php
// include some common code
include_once './common.php';
// Check for a POSTed status message to send to Twitter
if (!empty($_POST) && isset($_POST['status'])
&& isset($_SESSION['TWITTER_ACCESS_TOKEN'])) {
    // Easiest way to use OAuth now that we have an Access Token is to use
    // a preconfigured instance of Zend_Http_Client which automatically
    // signs and encodes all our requests without additional work
    $token = unserialize($_SESSION['TWITTER_ACCESS_TOKEN']);
    $client = $token->getHttpClient($configuration);
    $client->setUri('http://twitter.com/statuses/update.json');
    $client->setMethod(Zend_Http_Client::POST);
    $client->setParameterPost('status', $_POST['status']);
    $response = $client->request();
    // Check if the json response refers to our tweet details (assume it
    // means it was successfully posted). API gurus can correct me after.
    $data = json_decode($response->getBody());
    $result = $response->getBody(); // report in error body (if any)
    if (isset($data->text)) {
```

```

    $result = 'true'; // response includes the status text if a success
}
// Tweet sent (hopefully), redirect back home...
header('Location: ' . URL_ROOT . '?result=' . $result);
} else {
// Mistaken request? Some malfeasant trying something?
exit('Invalid tweet request. Oops. Sorry.');
```

When I said there's more, I meant it. From now on, every single API request concerning our user must be authorised using our OAuth Access Token. This is done, handily enough, by signing all requests using HMAC-SHA1. Now, doing this manually can be a pain, so using the Access Token object you can simply retrieve an instance of Zend\_Http\_Client (subclasses by Zend\_Oauth\_Client) which will handle all the OAuth protocol parameters and request signing transparently. Just use as normal as you would Zend\_Http\_Client.

So using this new client, we merely implement part of the Twitter API. Sending status messages is done by sending a POST request to `http://twitter.com/statuses/update.format` (where "format" must be changed to one of "json", "xml", "atom" or "rss") which includes a "status" parameter (in the POST body) containing our tweet. Once the response is received, we do a little checking to ascertain whether it was a success or failure, and redirect the user back to `index.php` once more.

The final file we haven't mentioned is linked to from the bottom of `index.php`'s output. `clear.php` is a simple script to delete the current Access Token and trigger a new OAuth authorisation process back through Twitter (just in case you missed it the first time around):

```


<?php
// include some common code
include_once './common.php';
// Clear the Access Token to force the OAuth protocol to rerun
$_SESSION['TWITTER_ACCESS_TOKEN'] = null;
// Redirect back to index and the protocol legs should run once again
header('Location: ' . URL_ROOT . '/index.php');
```

Now go and do some Twittering! In any other Twitter client, your new tweets should also display nearby the name of the application you registered with Twitter.

## Other Considerations on OAuth?

There are a few things I'd like to mention before closing. The first is that Zend\_Oauth implements Revision A of the OAuth specification (1.0a) which accounts for a session fixation vulnerability in the original specification. There is no need to worry about that here, and 1.0a is being rolled out to other service providers as we speak, if not already. Twitter implemented the improvements back in June. The component is still compatible with the original specification for service providers who haven't yet updated their implementation - it's completely transparent. Zend\_Oauth also makes use of Zend\_Crypt (in the Incubator also) so we offer full support for HMAC and RSA hashing via its subcomponents.

## Conclusion


Well, there you have it. A simple little app for making tweets using the Twitter API over OAuth. Obviously this is a very simple example to show off OAuth but I think I kept it neat enough to explore its operation without confusing everyone thoroughly 

. In reality, tokens will need to be managed with much more care since they are valid for extended periods (in fact many times the provider won't expire them for the foreseeable future). You also need to ensure they are associated with the correct user.

## Have fun with OAuth!

Posted by Pádraic Brady in PHP General, PHP Security, Zend Framework at 20:39

Great post and great work. Looking forward to ZF 1.10 Anonymous on Jul 29 2009, 21:04

Great article & implementation, simple to use for the developer, all the hard work hidden away. That should help uptake (at least by me 


) Anonymous on Jul 29 2009, 22:12

One of the nice things in Zend is there is already a decent twitter client. The Zend\_Oauth can be used with it!

see the following (adapted from a class in use at twitgoo, but I have not run it directly, but you'll get the picture):

```
class App_Service_Twitter_Oauth extends Zend_Service_Twitter {  
    //$username can be gleaned from extended response from getting access token - needed for a couple twitter fns  
    public function __construct($username, Zend_Oauth_Token_Access $token) {  
        //oauthoptions as defined in original body  
        self::setHttpClient($token-&gt;getHttpClient(self::$oauthOptions));  
        parent::__construct($username, null);  
    }  
}
```

Anonymous on Jul 29 2009, 22:43

Shhhh. This is the OAuth topic - Twitter is just the example 

. But true, the next Twitter component iteration should hopefully by ZF 1.10 have OAuth implemented in this or a similar fashion. Anonymous on Jul 29 2009, 22:55

Mostly, I'm just noting that any service that uses a Zend\_Http\_Client could be 'oauthized' with your adapters, which is totally awesome. Anonymous on Jul 30 2009, 16:50

Yeah, hopefully that makes using it in anything where OAuth is an option as easy as possible. The client code itself is also fairly standard stuff - easy to adapt to non-Zend clients. Anonymous on Jul 30 2009, 18:20


Thanks for the article explaining how oauth is getting implemented in the Zend framework. Any chance we will see an article on how to setup your own Zend\_Oauth server? Anonymous on Aug 5 2009, 20:20

Wouhou it is really simple to have it working <http://twitter.com/kanjiroushi/status/3229018313>

I would also be really interested in an example of the server side because I have difficulties to put pieces together only with the class code. Anonymous on Aug 11 2009, 15:32

Nice tutorial. How would I use this with Zend\_Service\_Twitter? I already have a model that uses Zend\_Service\_Twitter. Anonymous on Aug 17 2009, 18:06

I'll have a tutorial on this in a while. Basically pass the OAuth generated HTTP Client to the Twitter component using the static setHttpClient() method if available. See OnyxRaven's response earlier. I haven't tried this yet - it probably will not work off the bat if the Twitter component sets an Raw POST data (my client impl. will have raw POST parsing added as a feature soon so the correct signature is generated where that is used). If the Twitter components uses normal setParameterPost() method, it should work out of the box this way. Anonymous on Aug 18 2009, 01:19

Thanks Padraic. I will wait for the tutorial 

A couple of questions...

- How would I get the OAuth generated HTTP Client?
  - And how would "set" the Http Client in Zend\_Service\_Twitter?
  - Wouldn't Zend\_Service\_Twitter require a username and password? I would have to "edit" the Zend\_Service\_Twitter class?
- Anonymous on Aug 18 2009, 04:46

Ah, I see what you mean. This a problem with Zend\_Twitter\_Service - it's password based. OAuth doesn't use passwords - you would configure it as in this tutorial, get your access token and store it. As above, the access token is an object that generates an OAuth enabled instance of Zend\_Http\_Client (it's actually a subclass returned). After that you don't need a password - just make API requests.

As I said, I'm not familiar with the Twitter component. It could require a subclass to get rid of the username/password usage. Anonymous on Aug 18 2009, 11:19

Thanks for the guidance Padraic. I just took a peek at the Twitter component of ZF and found out that I can work out (i think) something to over-ride the \_\_construct and \_init which are setting the password, etc.

The rest of the code in the Twitter component seem to be harmless REST calls to Twitter. Anonymous on Aug 18 2009, 12:16

Hi again Padraic. Have other users reported this error?

```
[lang="php"]
Catchable fatal error: Argument 2 passed to Zend_Oauth_Token_Access::toQueryString() must implement interface
Zend_Oauth_Config_Interface, instance of Zend_Oauth_Client given, called in /htdocs/twitter/library/Zend/Oauth/Client.php on line
131 and defined in /htdocs/twitter/library/Zend/Oauth/Token/Access.php on line 55
[/lang]
```

I finally got it working, except for the error above. Anonymous on Aug 21 2009, 18:13

I downloaded the zend framework and I don't see oauth in the library. I also downloaded all of the files for this tutorial and the corresponding zend framework files are not there either. where can I get these files. for instance;

Zend/Oauth/Consumer.php

-boice Anonymous on Aug 24 2009, 19:35

They are in the Incubator - check the website for where to find this in subversion. Zend\_Oauth wouldn't be in any of the CDN downloads since it's not a released component just yet. Anonymous on Aug 25 2009, 10:33

From the commandline (assuming svn is installed), run:

svn checkout http://framework.zend.com/svn/framework/standard/incubator/library/ Anonymous on Sep 30 2009, 15:32