# Single-User Twitter OAuth Programming

By Adam Green

CEO, 140dev.com

# Single-User Twitter OAuth Programming

Adam Green

Adam Green Press

Lexington, Massachusetts

## Welcome to Twitter OAuth

The simplest way to explain Twitter OAuth is that it allows an application to interact with a user’s account without knowing the user’s password. Each Twitter application is given a pair of codes, called tokens or keys, when it is created. If a user wants the application to perform actions on their behalf, such as tweeting or following, they are sent to a Twitter authorization page. As part of the authorization process, the user gives Twitter their login user name and password, and Twitter gives the application a second pair of tokens that are unique to that user and that application. The application can then use the combination of its tokens and the user’s tokens to make requests to the Twitter API.

The benefit of OAuth for the user is that the application doesn’t get to see their Twitter password, and the user tokens received from Twitter may only be used by that application. This means that a user can cancel the authorization for a single application without losing the use of all the other applications they have authorized. This is much more secure and flexible than using the same account password with all applications.

The benefit for Twitter is that it can now control and rate limit all requests made to the API at a very fine level. It knows which application is making the request and which user the application is representing. Twitter is extremely generous with the free access it provides applications, but it is engaged in a continual war with rogue and spam applications. The level of control OAuth gives Twitter is essential to their ability to offer this access without being overwhelmed by bad actors.

OAuth was first introduced to the Twitter API version 1.0 in August, 2010, when it replaced basic authentication based on user name and password for API requests that modified an account. API calls that didn’t make any changes, such as getting a user’s followers, could still be done with no authentication. With the release of version 1.1, all Twitter API requests, even searching for tweets, now require OAuth. So if you have been putting off learning OAuth, the time has come to make the conversion.

## Single and multi-user OAuth programming

The process of allowing any user to login to a Twitter account through OAuth and authorize an app can be complicated. It requires the programmer to set up a website that manages the interaction between the user and Twitter, and that is able to identify the user through cookies when they return to the site. Since this may be more infrastructure than is needed for an application that will work with a single Twitter account the programmer already controls, Twitter also provides a simpler model called single-user OAuth. This lets a programmer obtain a set of user tokens without having to set up an authorization interface. This ebook will only cover the single-user approach to OAuth.

Single-user OAuth is the best way to get started with the Twitter API, because it lets you build an application quickly without any user account interface. Once you are familiar with the techniques needed to interact with the API for a single account, you can add the necessary machinery and UI to manage authorization for multiple visitors to your site. The good thing is that all of your single-user API code will still function when you move up to multi-user. You just have to add the necessary logic to select which set of user tokens you want to supply to the API when you make your requests.

## What will you learn?

The goal of this ebook is to give you a quick and easy path to managing the Twitter API with a single set of OAuth tokens. You will learn how to create your first application, call the Twitter API to post a tweet or request the account profile for any user, and then examine the values returned by the API. These techniques can be applied to the full set of the API requests, and will serve as a solid foundation for a wide range of application capabilities.

To help you move on to using all of the available API calls, you will learn how to translate the official API documentation into actual PHP code. Learning how to program with the API from the Twitter docs is like learning to drive a car by reading a map. It doesn’t help you until you know how to translate directions on a map into concrete actions like turning the wheel and pressing different pedals at the right time. You can think of this ebook as a first driving lesson for the API.

To make it easier to explore the API when you want to try a new request, you will also learn how to create a simple API console web page that will let you test out the possible arguments you can pass to each API call and then view the data returned. This is a much simpler approach than having to write test code each time you want to try something new.

## Programming tools and skills required

To use the example code shown here all you need is a computer running PHP 5.1 or higher that has access to the Internet. It can be a remote server or your own local computer. The choice of OS is up to you. PHP can run on any Linux/Unix, Windows, or Mac OS system. You won’t need a database for these tutorial examples, but real-world Twitter applications typically use some form of long-term storage, such as MySQL. If you want to build apps that others can reach, you’ll have to install web server software, such as Apache, on your computer, or use a remote server with this software installed.

This ebook assumes that you have a fundamental understanding of PHP programming. The source code is fully commented, so even if you aren't a PHP expert, you should be able to understand everything without any problems. It also assumes that you know how to place your code in the proper location on your server, configure it to be executable, and run it through a control panel or a command line client, such as SSH or Telnet.

### Source code is available for download

All of the source code shown in this ebook can be downloaded from the 140dev.com website at the URL: http://140dev.com/member. You will find detailed install instructions within the downloadable zip file, but basically all you have to do is place the downloaded code within a web accessible directory on any server.

## Creating a Twitter application

The first step in using OAuth is creating an application for one of your Twitter accounts. We will be doing single-user programming, which means you can only modify the account that owns the app.  The OAuth tokens you receive will also let you request information about any account. So if all you want to do is collect information about multiple accounts, such as follower stats or a user's timeline tweets, you can create the application within any account you control.

There is no published limit on the number of applications a Twitter account can have. You can create a single app and do all your programming with its tokens, or create applications for each major set of functionality. It is really a matter of personal style. The only limitation to be aware of is that all Twitter apps must have a unique name. Not just unique among your apps, but unique among all apps ever created on Twitter by any account. The simple solution if you find a conflict for your app's name is to start the name with the screen name of its Twitter account. Later on, when you decide to move up to multi-user OAuth, the name you give your app will be visible to all your users, so try to make the name descriptive.

When you create your application, you need to decide whether it will just collect information from the API or modify its account as well. Modifications include tweeting, following, and changing the account profile. If you just want to collect info, then the application can have a Read only access level. To modify an account with your app, it will need an access level of Read and Write. The highest level of access is Read, Write, and Direct Message. The example code we'll use here will need Read and Write access.

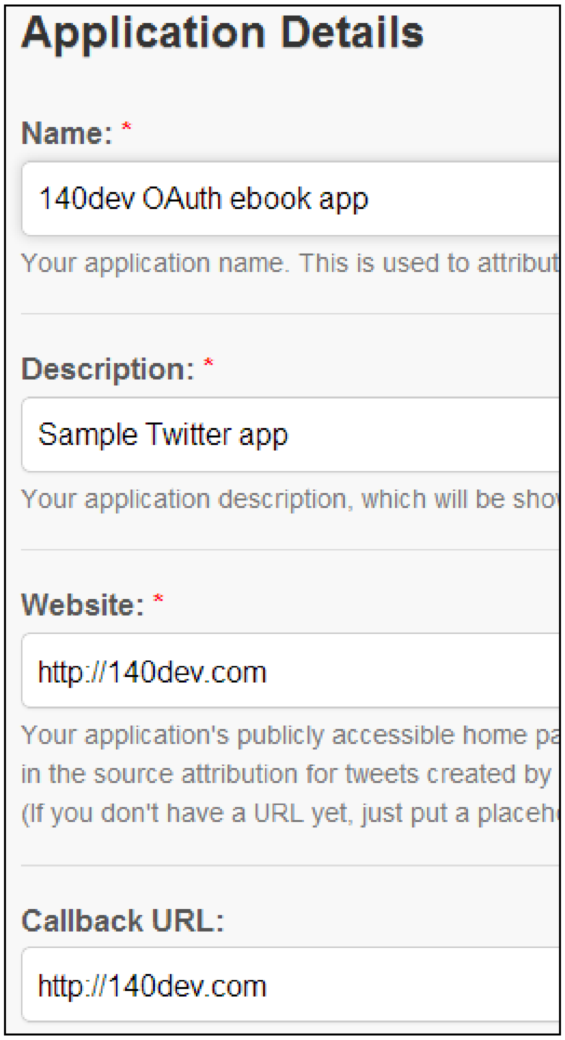
*Go to **http://dev.twitter.com** in your browser.*

*Click **Sign in** on the top right, and log into the account that you want to modify with the API.*

*Select **My applications** from the user pull-down menu on the top right.*

*Click the **Create Application** button.*

*Fill in the following fields: **Name**, **Description**, **Website**, and **Callback URL**.*



Application Details

Name: *
140dev OAuth ebook app
Your application name. This is used to attribut

Description: *
Sample Twitter app
Your application description, which will be sho

Website: *
http://140dev.com
Your application's publicly accessible home pa
in the source attribution for tweets created by
(If you don't have a URL yet, just put a placeh

Callback URL:
http://140dev.com

The lack of an asterisk on the form implies that the Callback URL isn't required. In practice, however, you must put a URL here, or the tokens Twitter creates will not work correctly. Just put any valid URL into this field, such as your personal blog or even http://twitter.com. One more tip, you must include the protocol portion of the URL for both website and callback, such as http:// or https://, or the form will be rejected and you will get an error that the URL is not valid.

*Scroll down and click the checkbox for the **Terms of Service**.*

*Fill in the **Captcha**.*

*Click the Create your Twitter application button.*

The Details page will then appear, but if you scroll down, you will see that your application has been given a default access level of Read-only. For some reason, you can't set the access level when you create your app. So we're not done yet.

*Click the **Settings** tab.*

*Scroll down and change Application Type to Read and Write.*

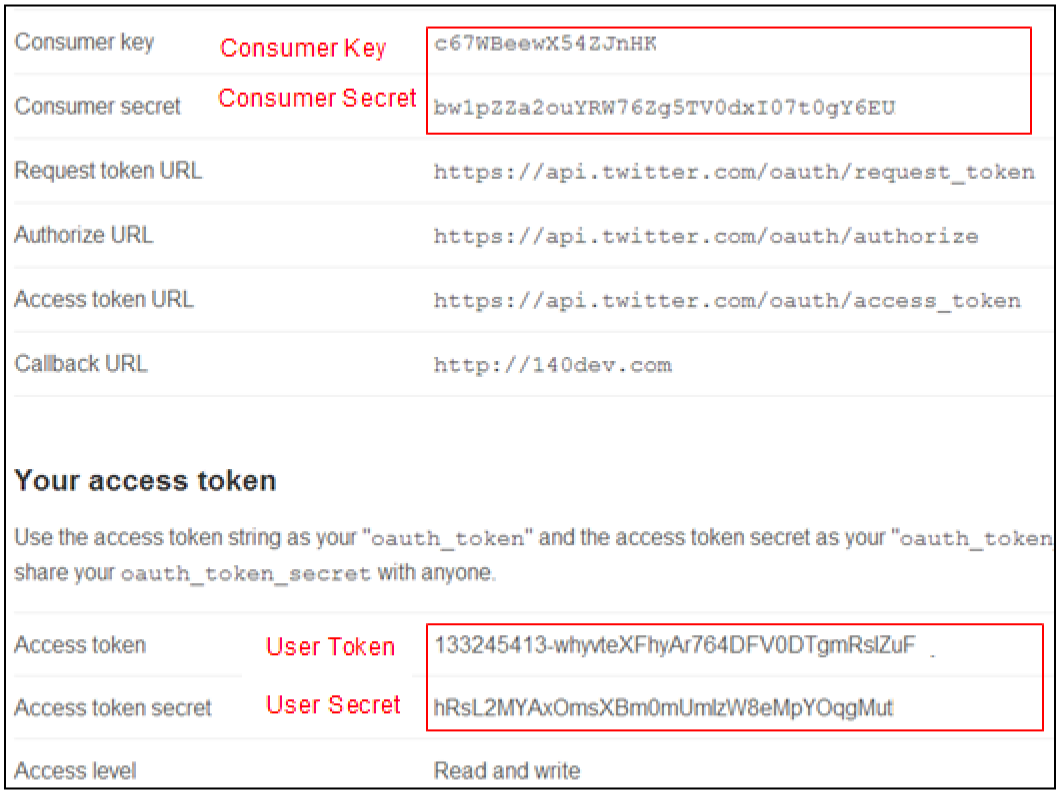
*Scroll down and click the Update this Twitter application's settings button.*

The access level may still say Read only, but if you wait a few minutes and then reload the page, it should have switched to Read and Write. Now you can get your full set of OAuth tokens and begin programming.

*Click the **Details** tab.*

*Scroll down and click the **Create my access token** button.*

Once again, you may have to wait a few minutes and reload the page to see the new tokens.



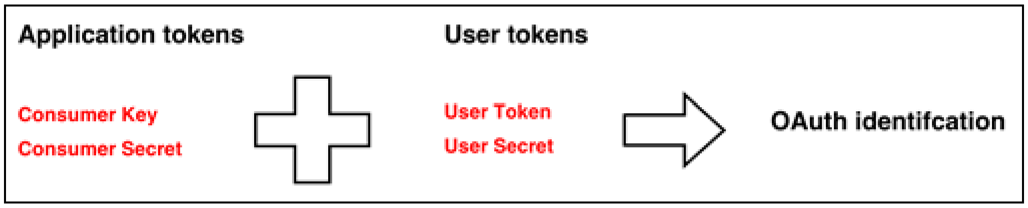
| | | |
|---|---|---|
| Consumer key | Consumer Key | c67WBeewX54ZJnHK |
| Consumer secret | Consumer Secret | bw1pZZa2ouYRW76Zg5TV0dxI07t0gY6EU |
| Request token URL | | https://api.twitter.com/oauth/request_token |
| Authorize URL | | https://api.twitter.com/oauth/authorize |
| Access token URL | | https://api.twitter.com/oauth/access_token |
| Callback URL | | http://140dev.com |

**Your access token**

Use the access token string as your "oauth_token" and the access token secret as your "oauth_token
share your oauth_token_secret with anyone.

| | | |
|---|---|---|
| Access token | User Token | 133245413-whyvteXFhyAr764DFV0DTgmRslZuF |
| Access token secret | User Secret | hRsL2MYAxOmsXBm0mUmlzW8eMpYOqgMut |
| Access level | | Read and write |

There are two pairs of tokens that we will now use in our OAuth code. Unfortunately the names of these tokens tend to vary within the Twitter docs. The names that we will use in this ebook have been labeled in red above. The pair that identify the application are the consumer key and consumer secret, and the pair that identify the user of the application are the user token and user secret. In multi-user OAuth, each authorized user of the app gets their own user token and user secret, while the consumer key and consumer secret are always the same for that application.



## Saving and protecting your OAuth tokens

Now that you have a complete set of tokens, you must be careful to protect them. If a malicious coder gets a copy of your OAuth tokens, they can use them in a script that breaks Twitter's rules, and your account will get the blame. Make sure you never reveal your tokens on a message board or Twitter's own developers forum. Just like any other login credentials you need to protect, it is best to put the tokens in a single file, and then keep that file secure. For the sample code we'll work with here the tokens are kept in a PHP script called **app_tokens.php**. The rest of the scripts will load the tokens with a **require** command when needed. NOTE: The tokens shown here are no longer valid. You must place your own OAuth tokens in this script for the example code to work.

*app_tokens.php*

```
<?php

// OAuth tokens for @140dev app: 140dev OAuth ebook app
// You *MUST* replace them with tokens for your app
$consumer_key = 'c67WBeewX54ZJnHK';
$consumer_secret = 'bw1pZZa2ouYRW76Zg5TV0dxI07t0gY6EU';
$user_token = '133245413-whyvteXFhyAr764DFV0DTgmRsIZuF';
$user_secret = 'hRsL2MYAxOmsXBm0mUmIzW8eMpYOqgMut';

?>
```

## Using an OAuth library

One aspect of OAuth that is particularly difficult is creating a complete HTTP header for an API request. The details are documented on the Twitter site, if you want to take a quick look: https://dev.twitter.com/docs/auth/authorizing-request.

Most PHP programmers aren't interested in working at that level of complexity. If we were, we wouldn't have become PHP programmers. Luckily there are a number of free libraries that will do all the dirty work for you. All you have to do is provide your tokens, the name of the API request, and the arguments specific to your needs, and the library will create a complete OAuth request for you. You can select a library for PHP from the Twitter website: https://dev.twitter.com/docs/twitter-libraries.

This ebook will use the tmhOAuth library by Matt Harris, which was selected because of its simplicity. The other OAuth libraries try to provide a complete framework for API programming, which means in practice that the actual API calls are buried within the library's code. The tmhOAuth library, on the other hand, keeps it as simple as possible. In fact, this library is so simple you only need to add 2 files to your code.

The source code provided with this ebook includes these two files: **cacert.pem** and **tmhOAuth.php**. The first file is an SSL certificate, needed because the API requests use SSL. The library handles the SSL transaction for you. The second file is the actual OAuth library. You'll learn how to use the library next, but you can also visit the Github repository for tmhOAuth to get additional examples, and to download the most recent version of the code: https://github.com/themattharris/tmhOAuth. NOTE: You MUST install cacert.pem in the same directory as tmhOAuth.php. Failure to do this will cause the code to fail.

## Posting a tweet

Let's try our first OAuth request by learning how to post a tweet. This is single-user OAuth, so the account in which the app was created will be the account that sends the tweet. There are four basic steps required to use the API with OAuth:

1. Load the tokens and library into memory.
2. Make a connection to the API using the tokens.
3. Send the API request.
4. Evaluate the results.

The **post_tweet.php** script puts all this together. Read it over, and then we'll break it down into these steps.

*post_tweet.php*

```
<?php
// Load the app's keys into memory
require 'app_tokens.php';

// Load the tmhOAuth library
require 'tmhOAuth.php';
```

```
// Create an OAuth connection to the Twitter API
$connection = new tmhOAuth(array(
 'consumer_key' => $consumer_key,
 'consumer_secret' => $consumer_secret,
 'user_token' => $user_token,
 'user_secret' => $user_secret
));

// Send a tweet
$code = $connection->request('POST',
        $connection->url('1.1/statuses/update'),
        array('status' => 'Hello Twitter'));

// A response code of 200 is a success
if ($code == 200) {
 print "Tweet sent";
} else {
 print "Error: $code";
}

?>
```

Pretty simple, right? Here is the code broken down in terms of the standard steps you will perform each time you make an OAuth request.

**Load the tokens and library into memory**

This assumes that the script that places the tokens into memory variables and the tmhOAuth library are in the same directory as the rest of your code. If this isn't true, you will have to adjust the path used in the **require** statements.

```
require 'app_tokens.php';
require 'tmhOAuth.php';
```

**Make a connection to the API using the tokens**

TmhOAuth is written as a class library, so you have to use the **new** operator to make the connection using your tokens. Once a connection object is created, you can use it repeatedly in your script.

```
$connection = new tmhOAuth(array(
 'consumer_key' => $consumer_key,
 'consumer_secret' => $consumer_secret,
 'user_token' => $user_token,
 'user_secret' => $user_secret
));
```

### Send the API request

```
$code = $connection->request('POST',
          $connection->url('1.1/statuses/update'),
          array('status' => 'Hello Twitter'));
```

As you can see, there are three parts to an API request when using this library:

1. The HTTP method, which is **POST** for this request.
2. The URL, which is **1.1/statuses/update** when you are sending a tweet.
3. The parameters for the request, which are formatted as an array.

You can see the Twitter documentation page for this request at: https://dev.twitter.com/docs/api/1.1/post/statuses/update. You'll learn how to decode these docs and translate them into the format needed for your PHP code soon.

### Evaluate the results

The API returns an HTTP code for each request. You can review all the possible codes here: https://dev.twitter.com/docs/error-codes-responses. The simplest way to approach this is to treat a code of 200 as  success, and anything else as an error.

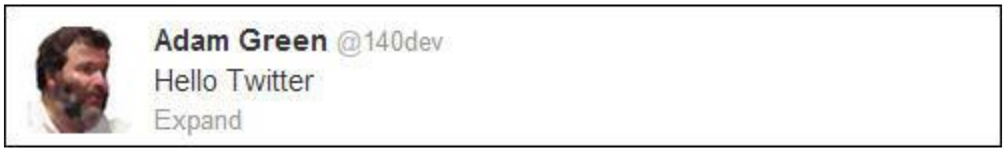
```
if ($code == 200) {
 print "Tweet sent";
} else {
 print "Error: $code";
}
```

## Running the post_tweet.php script

As long as you have copied the source code to an Internet enabled computer, stored your tokens to the **app_tokens.php** script, and made the scripts executable, you can run them with PHP. For the examples shown here, the scripts were loaded onto a Linux server and run at the command line with an SSH client.

```
$ php post_tweet.php
Tweet sent
```

The tweet will show up within the account used to create the app.



Adam Green @140dev
Hello Twitter
Expand

If you try running the **post_tweet.php** script a second time, however, you will get back an error code of 403. That is because the API rejects duplicate tweets.

```
$ php post_tweet.php
Error: 403
```

## Getting API results

The other half of sending an API request is working with the results. We've already seen that the HTTP result code is returned when you make a request with tmhOAuth. The data returned by the API is also stored in the **$connection** object. This can be retrieved with the **response** element.

```
$response_data = $connection->response['response'];
```

By default the API returns its results as a JSON string. To manipulate this response within a PHP program, you can convert it into an object with the **json_decode()** function. If you would rather work with the data as an array instead of an object, you can include a value of **true** as a second argument to this function.

```
$response_data = json_decode($connection->response['response'],true);
```

### User account info

We can try this out with the **users_show.php** script, which gets the account profile for **@justinbieber**. It will make a request for this account's info using the **users/show** API call, documented at: https://dev.twitter.com/docs/api/1.1/get/users/show. Single-user OAuth only allows us to modify a single account, but we can request info about any account. In this case, all that is needed is supplying an account name as the value of the **screen_name** parameter.

*users_show.php*

```
<?php

require 'app_tokens.php';
require 'tmhOAuth.php';

$connection = new tmhOAuth(array(
  'consumer_key' => $consumer_key,
  'consumer_secret' => $consumer_secret,
  'user_token' => $user_token,
  'user_secret' => $user_secret
));
```

```
// Get @justinbieber's account info
$connection->request('GET', $connection->url('1.1/users/show'),
array('screen_name' => 'justinbieber'));

// Get the HTTP response code for the API request
$response_code = $connection->response['code'];

// Convert the JSON response into an array
$response_data = json_decode($connection->response['response'],true);

// A response code of 200 is a success
if ($response_code <> 200) {
  print "Error: $response_code\n";
}

// Display the response array
print_r($response_data);
?>
```

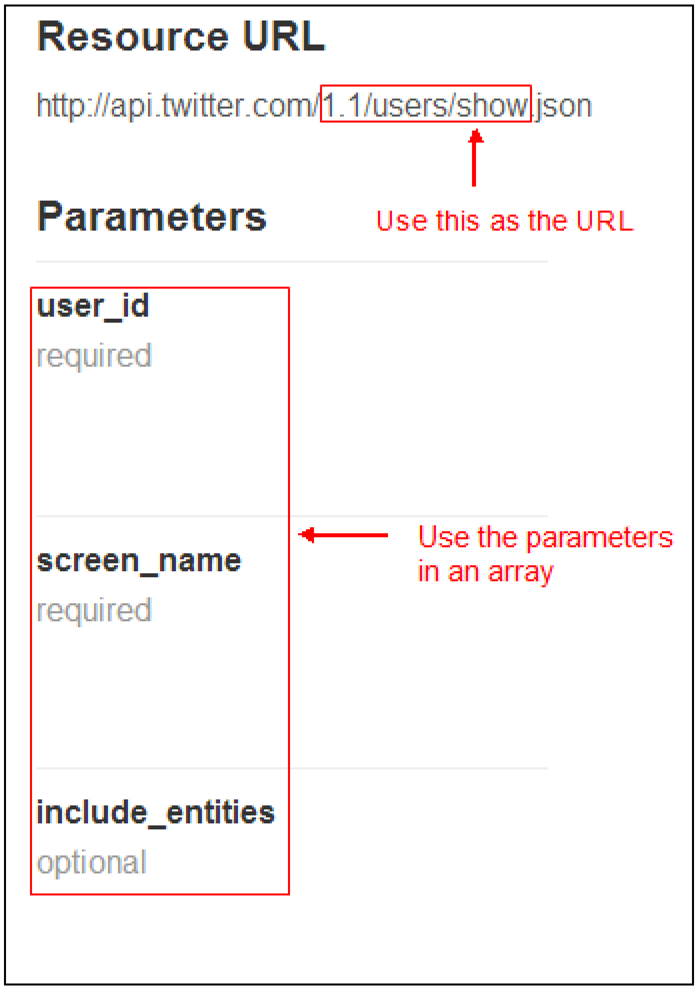
This API call returns a lot of data, including everything Twitter knows about this account, and everything it knows about the account's most recent tweet. There is too much to reproduce here, but you will find an abridged version below.

```
$ php users_show.php

array
(
  [id] => 27260086
  [id_str] => 27260086
  [name] => Justin Bieber
  [screen_name] => justinbieber
  [location] => All Around The World
  [description] => #BELIEVE is on ITUNES and in STORES WORLDWIDE! - SO MUCH LOVE FOR THE FANS...you are
always there for me and I will always be there for you. MUCH LOVE. thanks
  [url] => http://www.youtube.com/justinbieber
    ...
  [protected] =>
  [followers_count] => 31561089
  [friends_count] => 122325
  [listed_count] => 544399
  [created_at] => Sat Mar 28 16:41:22 +0000 2009
  [favourites_count] => 10
  [utc_offset] => -18000
  [time_zone] => Eastern Time (US & Canada)
  [geo_enabled] =>
  [verified] => 1
  [statuses_count] => 20041
  [lang] => en)
```

## Convert Twitter documentation to API calls

Now that you have a better idea of how to drive the API, you are ready to learn how to read the map provided by the Twitter API documentation. Once you see the pattern, it's actually pretty clear. Here are the key portions of the API description for the **/users/show** request found at: https://dev.twitter.com/docs/api/1.1/get/users/show.



Resource URL

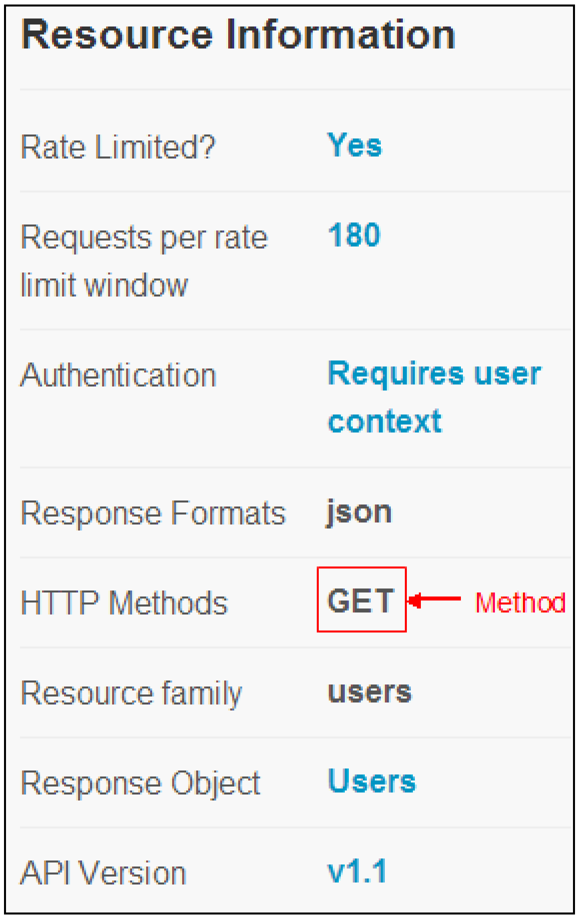
http://api.twitter.com/1.1/users/show.json

Parameters

user_id
required

screen_name
required

include_entities
optional



Resource Information

| | |
|---|---|
| Rate Limited? | Yes |
| Requests per rate limit window | 180 |
| Authentication | Requires user context |
| Response Formats | json |
| HTTP Methods | GET |
| Resource family | users |
| Response Object | Users |
| API Version | v1.1 |

You can place the highlighted parts of the docs into an API request using this format:

```
$connection->request('METHOD',
 $connection->url('URL'),
 array( 'PARAMETER' => 'VALUE'));
```

For example:

```
$connection->request('GET',
 $connection->url('1.1/users/show'),
 array( 'screen_name' => 'justinbieber'));
```

### Using multiple parameters

If you have more than one parameter you want to include in a request, you need to use the standard PHP array syntax of comma separated key-value pairs:

```
$connection->request('METHOD',
 $connection->url('URL'),
 array( 'PARAMETER' => 'VALUE',
  'PARAMETER' => 'VALUE',
  …));
```

For example, setting the **include_entities** parameter to **false**, which eliminates entities from the user's most recent tweet, is formatted as follows:

```
$connection->request('GET',
 $connection->url('1.1/users/show'),
 array( 'screen_name' => 'justinbieber'
  'include_entities' -> false));
```

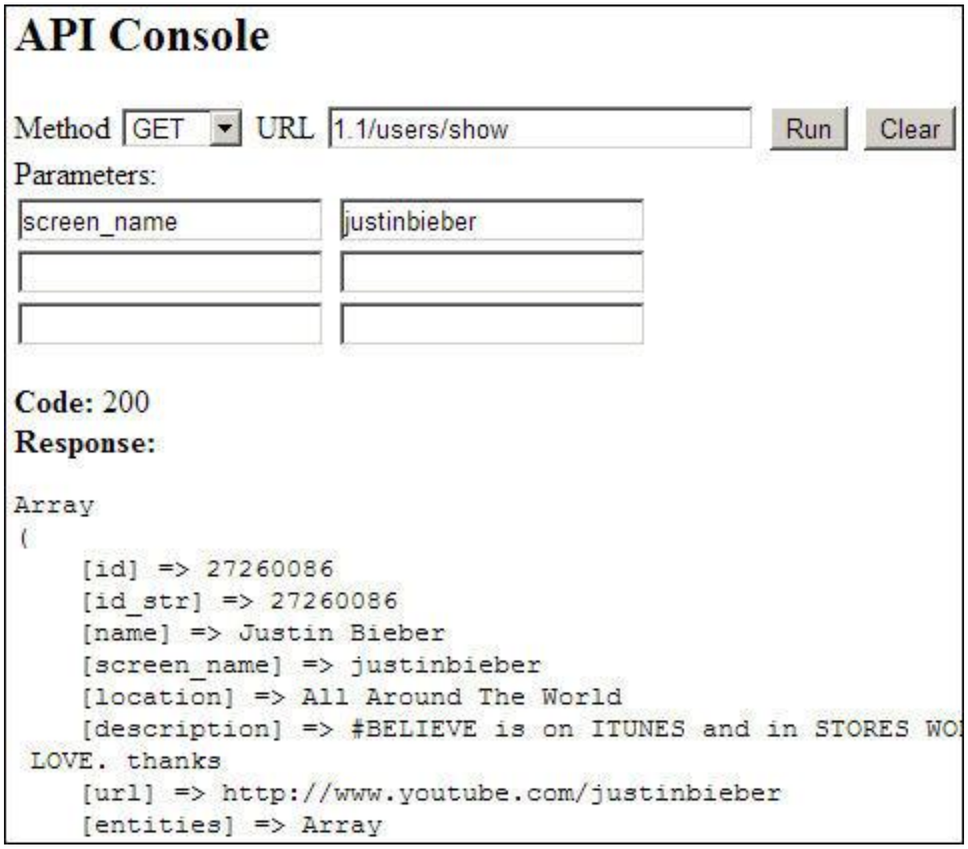
Entities are values such as tags, URLs, or @mentions that are found within tweets. Turning them off makes the response smaller, and therefore faster to deliver.

## API Console

Constructing a correct request is only half of the job of learning how to use a new API call. The other half is figuring out how the response is structured, so you can extract the right values from the nested arrays that are returned by the API. You could create test scripts, like the simple **users_show.php** we've just seen, and then print the responses, but this is a cumbersome and time consuming process. To make this job easier, we'll close with a simple console app that will let you quickly test any API call. It can also serve as a starting point for building other simple API apps.

The API console is composed of three scripts included in the downloadable source code: **console.php**, **console_form.php**, and **console_run.php**. If you placed the source code in a web accessible directory on your server, you can run the console by calling **console.php**. It will use the OAuth tokens you should have already stored in **app_tokens.php**. For example, if you placed the source code in mydomain.com/oauthcode, you can start the console with the URL: http://mydomain.com/oauthcode/console.php.

All you have to do to try any API call with this tool is select the proper HTTP method, and enter a valid API URL with any parameters that are needed. Then you can click **Run** to execute the request. You will see the HTTP code returned, and the complete response. The full response is not shown here, but it will fill the page.



**Code:** 200
**Response:**

```
Array
(
    [id] => 27260086
    [id_str] => 27260086
    [name] => Justin Bieber
    [screen_name] => justinbieber
    [location] => All Around The World
    [description] => #BELIEVE is on ITUNES and in STORES WO
 LOVE. thanks
    [url] => http://www.youtube.com/justinbieber
    [entities] => Array
```

The code architecture for this tool is straight-forward PHP form processing. The first time **console.php** is executed, the **$run** variable is **false** and the form is displayed with default values by **console_form.php**. When the user clicks the Run button, **console.php** is executed again. This time **$run** is set to **true** in **console_form.php**, and the form values are passed to **console_run.php**.

*console.php*

```
<?php

print "<h2>API Console</h2>";

// Display the input form
$run = false;
require 'console_form.php';

// If the user clicked the Run button
// Execute the API request they entered
if ($run) {
        require 'console_run.php';
}

?>
```

PHP form security can be done in many ways, and you can add your own enhancements to **console_form.php**. As a minimum security precaution the input variables for the form are escaped with the **htmlspecialchars** function before their values are redisplayed. This converts dangerous Javascript that a user might place in an input field into safe HTML.

*console_form.php*

```
<?php

// If the user clicked the Run button
if (isset($_POST['submit'])) {

        // Gather their input
        // Escape input values before redisplaying them in the form
        $run = true;
        $method = htmlspecialchars($_POST['method'], ENT_QUOTES);
        $url = htmlspecialchars($_POST['url'], ENT_QUOTES);
        $parameter1 = htmlspecialchars($_POST['parameter1'], ENT_QUOTES);
        $value1 = htmlspecialchars($_POST['value1'], ENT_QUOTES);
        $parameter2 = htmlspecialchars($_POST['parameter2'], ENT_QUOTES);
        $value2 = htmlspecialchars($_POST['value2'], ENT_QUOTES);
        $parameter3 = htmlspecialchars($_POST['parameter3'], ENT_QUOTES);
        $value3 = htmlspecialchars($_POST['value3'], ENT_QUOTES);

} else {

        // If the form is run for the first time,
        // or the Clear button is clicked,
        // clear the input fields
        $run = false;
        $method = 'GET';
        $url = '1.1/';
        $parameter1 = '';
        $value1 = '';
        $parameter2 = '';
        $value2 = '';
        $parameter3 = '';
        $value3 = '';
}

// Display the form with empty fields
// or the values entered before Run button was clicked
print "<form action='console.php' method='post'>";

print "Method <select name='method'>";
if ($method == 'GET') {
        print "<option selected='selected'>GET</option>";
} else {
        print "<option>GET</option>";
}
if ($method == 'POST') {
        print "<option selected='selected'>POST</option>";
} else {
        print "<option>POST</option>";
}
print "</select> ";

print "URL <input type='text' name='url' value='$url' size='50'> ";
print "<input type='submit' name='submit' value='Run' /> ";
print "<input type='submit' name='clear' value='Clear' /><br/>";

// Input fields for 3 parameters are included
// This can easily be changed to more if you wish
print "Parameters:<br/>";
print "<input type='text' name='parameter1' value='$parameter1'> ";
print "<input type='text' name='value1' value='$value1'><br/>";
print "<input type='text' name='parameter2' value='$parameter2'> ";
print "<input type='text' name='value2' value='$value2'><br/>";
print "<input type='text' name='parameter3' value='$parameter3'> ";
print "<input type='text' name='value3' value='$value3'><br/>";
print '</form>';
?>
```

The **console_run.php** script assembles a valid OAuth request based on the form input. It deliberately does little error checking. For example, you can enter a parameter name, but leave its value blank. This is passed along to the API, so you can see how it reacts.

*console_run.php*

```
<?php

// Connect to the API with OAuth tokens
require 'app_tokens.php';
require 'tmhOAuth.php';
$connection = new tmhOAuth(array(
  'consumer_key' => $consumer_key,
  'consumer_secret' => $consumer_secret,
  'user_token' => $user_token,
  'user_secret' => $user_secret
));

// Accumulate the parameters and their matching values
// in an array to be used in the API request
$array = array();
if (! empty($parameter1)) {
        $array = array_merge($array,array($parameter1=>$value1));
}
if (! empty($parameter2)) {
        $array = array_merge($array,array($parameter2=>$value2));
}
if (! empty($parameter3)) {
        $array = array_merge($array,array($parameter3=>$value3));
}
// Add more parameters here if you included them in the form

if (sizeof($array)==0) {
        // There were no parameters entered
        $connection->request($method,
                $connection->url($url));
} else {
        // Use the parameters entered on the form
        $connection->request($method,
                $connection->url($url),$array);
}

// Display the response HTTP code
$code = $connection->response['code'];
print "<strong>Code:</strong> $code<br/>";

// Display the API response as an array
print "<strong>Response:</strong><pre style='word-wrap: break-word'>";
print_r(json_decode($connection->response['response'],true));
print "</pre><br/>";
?>
```

## Testing for error codes

Because invalid inputs are still passed to the API, this console can be used to explore different error conditions. The official error codes are listed at: https://dev.twitter.com/docs/error-codes-responses. Over time you will discover that the API can be unpredictable when it returns error codes. For example, using an incorrect URL or parameter name can result in a range of errors at various times, such as 400, 401, 403, or 404.

You will also find that invalid OAuth tokens can result in either a 401 or 403 error. In fact, using the console to test your OAuth tokens is one of the fastest ways of making sure they  are still valid. You can do this with the API request **1.1/account/verify_credentials** and no parameters.

When you are debugging your API code and don’t understand why you are receiving a particular error code, the console app can be a useful tool for regaining your sanity.

## Additional resources

### 140dev.com

Source code download page - http://140dev.com/member

> You can download the source code for the examples in this ebook, as well as other sets of code and tutorials. This PDF is also available here.

140dev OAuth Google group - https://groups.google.com/d/forum/140dev-oauth-discussion

> Still have questions? OAuth driving you crazy? Share your questions and discoveries with other developers.

Streaming API framework - http://140dev.com/free-twitter-api-source-code-library

> This open source library will help you get up to speed with the Streaming API, allowing you to collect tweets, store them in a MySQL database, and display them on web pages.

### Official Twitter resources

API status - https://dev.twitter.com/status

> The next time you can't figure out if problems are due to your code, your server, or the Twitter API, come here to see how the API is behaving.

Field guide to API objects - https://dev.twitter.com/docs/platform-objects

> This is a great resource for the dirty details you can't find in the docs for individual API calls. You'll discover data buried in the API results you never noticed before.

OAuth discussion list - https://dev.twitter.com/discussions/authentication-oauth

> You probably should subscribe to all the Twitter.com discussion lists, but at a minimum you need to follow this one, if you are serious about OAuth programming.