dev@twitter

- API Status
- Documentation
- Discussions
- Your apps
- Sign out

Search

Basic Auth has been deprecated.

August 31, 2010 Basic Auth has been deprecated. All applications must now use OAuth. Read more »

Don't fret! @twitterapi is here to help! Feel free to reach out to us directly, or via our Twitter Development Talk group.

The switch to OAuth is a good thing! You, as the application developer,

- don't have the burden of keeping potentially damaging credentials for your users (especially considering that a lot of people use the same password for multiple services);
- don't have to worry about the user changing their password — a user can change his or her password and the OAuth "connection" to your app will still work;
- don't have to worry about other applications masquerading as your application as only your application can set the byline with your application name;
- will eventually have access to more trusted APIs from Twitter that will only be available to "trusted" OAuth-enabled applications; and
- will be contributing to the web of trust between users, service providers, and applications.

Choose your authorization path.

Authenticating Requests with OAuth

Home » Supplemental Documentation » **Authenticating Requests with OAuth**

We're here to help.

Many developers have trouble moving from Basic Auth to OAuth.

See Transitioning from Basic Auth to OAuth for an overview of the conversion process.

Overview

To use the Twitter API, the first thing you have to do is register a client application. Each client application you register will be provisioned a consumer key and secret. This key and secret scheme is similar to the public and private keys used in protocols such as `ssh` for those who are familiar. This key and secret will be used, in conjunction with an OAuth library in your programming language of choice, to *sign* every request you make to the API. It is through this signing process that we trust that the traffic that identifies itself is you is in fact you.

As of June 30, 2010, basic auth (directly passing a username and password for every API request) will no longer be supported. Web applications are encouraged to use full OAuth to authenticate users and act on their behalf. Desktop and mobile applications are encouraged to use OAuth. Mobile and desktop applications are also given the opportunity to use xAuth, a means to exchange a login & password for an access token. To use xAuth, send a request to api@twitter.com with plenty of details about your application and why xAuth is the best choice for it.

For applications with single-user use cases, we now offer the ability to issue an access token for your own account (and your own applications). You can generate these keys from your application details pages. Find out more about using a single access token.

This page is under construction. Sorry for being all pre-web 1.0 on you folks.

Registering an App

Create an application from the Client Applications page. It's very quick and easy to do. Registering your application allows us to identify your application. Remember to never reveal your consumer secrets.

While creating an application, you will be asked whether your application is a *desktop* or *web* application. Desktop applications need not enter a callback URL. In fact, web applications need not supply a callback URL either. A best practice is to always send us an `oauth_callback_url` on every request token step, explicitly declaring what you want the callback to be. In the case where you want to invoke what's called the PIN/out-of-band flow, you would supply an oauth_callback_url of "oob". See this section for more on the PIN-code flow.

Introduction to OAuth

Twitter uses the open authentication standard OAuth for authentication. To learn more about OAuth, we recommend reading the guide at Hueniverse, and this SlideShare presentation.

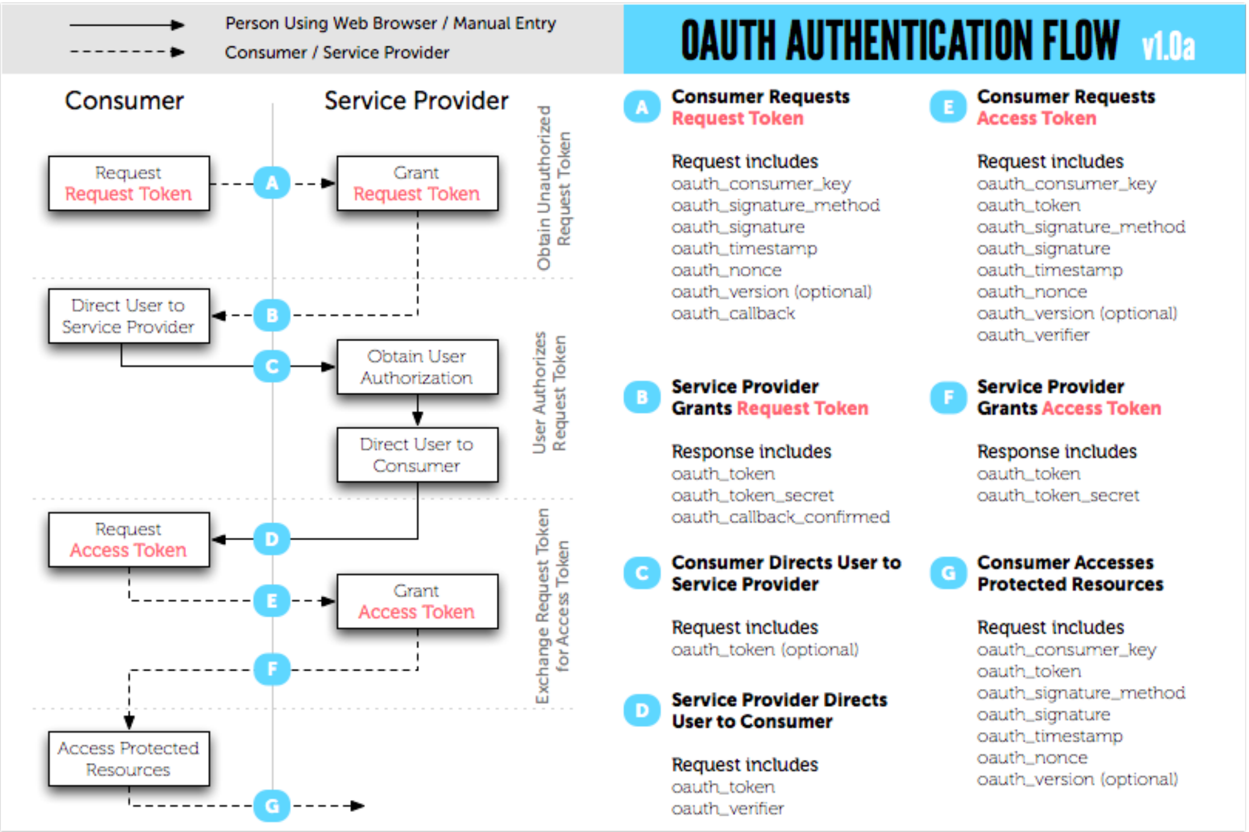
Authenticating with OAuth is easiest if you use a library in your favorite programming language.

The OAuth request cycle is roughly:

- Retrieve a request token
- Request user authorization by sending the user to a Twitter.com login page
- Exchange the request token for an access token

The flow varies slightly when using a desktop application with what's called the "PIN-mode flow" or exchanging login credentials using xAuth.

This diagram illustrates further:


Person Using Web Browser / Manual Entry
Consumer / Service Provider
OAUTH AUTHENTICATION FLOW v1.0a
Consumer
Service Provider
Request Request Token
A
Grant Request Token
Obtain Unauthorized Request Token
Direct User to Service Provider
B
C
Obtain User Authorization
Direct User to Consumer
User Authorizes Request Token
Request Access Token
D
E
Grant Access Token
F
Exchange Request Token for Access Token
Access Protected Resources
G

A Consumer Requests Request Token

Request includes
oauth_consumer_key
oauth_signature_method
oauth_signature
oauth_timestamp
oauth_nonce
oauth_version (optional)
oauth_callback

B Service Provider Grants Request Token

Response includes
oauth_token
oauth_token_secret
oauth_callback_confirmed

C Consumer Directs User to Service Provider

Request includes
oauth_token (optional)

D Service Provider Directs User to Consumer

Request includes
oauth_token
oauth_verifier

E Consumer Requests Access Token

Request includes
oauth_consumer_key
oauth_token
oauth_signature_method
oauth_signature
oauth_timestamp
oauth_nonce
oauth_version (optional)
oauth_verifier

F Service Provider Grants Access Token

Response includes
oauth_token
oauth_token_secret

G Consumer Accesses Protected Resources

Request includes
oauth_consumer_key
oauth_token
oauth_signature_method
oauth_signature
oauth_timestamp
oauth_nonce
oauth_version (optional)

OAuth offers two ways to provide OAuth authentication methods: query-string and HTTP headers. It is recommended that you use the header-based approach as it separates concerns and makes debugging much easier for you.

OAuth at Twitter

OAuth can be confusing because there are a few different variations on how it works. Here's a few key takeaways from the team:

- **Use header-based OAuth** - OAuth provides for means to pass OAuth-related parameters on a query string or to provide them as a HTTP Authorization header. Twitter prefers header-based auth because it separates concerns, makes debugging easier, and avoids common issues with under or over URL escaping parameters. Most OAuth libraries let you choose which kind of OAuth you are executing. Go for the gold. Go for header-based OAuth.
- **Use SSL for /oauth/* end points** - for all steps of the OAuth dance like request_token, access_token, and authorize, use SSL
- **Use api.twitter.com** - for all your OAuth steps, use api.twitter.com as the hostname, not just "twitter.com"
- **Always use an explicit oauth_callback** - It is recommended that you specify a default OAuth callback in your client record, but explicitly declare your oauth_callback on each request token fetch request your application makes. By dynamically setting your oauth_callback, you can pass additional state information back to your application and control the experience best. If using the PIN code flow, specify your oauth_callback as "oob".

Signing Requests

All OAuth 1.0A requests use the same basic algorithm for creating a signature base string and a signature.

The signature base string is often the most difficult part of OAuth for newcomers to construct. The signature base string is composed of the HTTP method being used, followed by an ampersand ("&") and then the URL-encoded base URL being accessed, complete with path (but not query parameters), followed by an ampersand ("&"). Then, you take all query parameters and POST body parameters (when the POST body is of the URL-encoded type, otherwise the POST body is ignored), including the OAuth parameters necessary for negotiation with the request at hand, and sort them in lexicographical order by first parameter name and then parameter value (for duplicate parameters), all the while ensuring that both the key and the value for each parameter are URL encoded in isolation. Instead of using the equals ("=") sign to mark the key/value relationship, you use the URL-encoded form of "%3D". Each parameter is then joined by the URL-escaped ampersand sign, "%26".

This algorithm is simply expressed in the pseudo-code:

```
  httpMethod + "&" +
  url_encode(  base_uri ) + "&" +
  sorted_query_params.each  { | k, v |
      url_encode ( k ) + "%3D" +
      url_encode ( v )
  }.join("%26")
```

No matter what kind of OAuth 1.0 request you are making, the rules for generating the signature base string remain constant.

Twitter requires that all OAuth requests be signed using the HMAC-SHA1 algorithm.

Acquiring a request token

The first step to authenticating a user is to obtain a request token from Twitter. This step serves two purposes: First, to tell Twitter what you're about to do. Second, to tell Twitter what you want to do for the OAuth callback.

Twitter's end point for the request token step is **`http://api.twitter.com/oauth/request_token`**.

You should use the **`POST`** HTTP method when using this end point. SSL is recommended.

Let's perform a request token request using HTTPS, POST, the following variables and the consumer secret "MCD8BKwGdgPHvAuvgvz4EQpqDAtx89grbuNMRd7Eh98":

- **`oauth_callback`** - http://localhost:3005/the_dance/process_callback?service_provider_id=11
- **`oauth_consumer_key`** - GDdmIQH6jhtmLUypg82g
- **`oauth_nonce`** - QP70eNmVz8jvdPevU3oJD2AfF7R7odC2XJcn4XlZJqk
- **`oauth_signature_method`** - HMAC-SHA1
- **`oauth_timestamp`** - 1272323042
- **`oauth_version`** - 1.0

First, we sort all the parameters used in our request and formulate a signature base string. Notice that my oauth_callback has query parameters in the URL. Because they query parameters are part of the callback URL, they don't get sorted alongside the other parameters in the request. Instead, the URL is URL-encoded and considered a single string. This builds the following signature base string:

```
POST&https%3A%2F
%2Fapi.twitter.com%2Foauth%2Frequest_token&oauth_callback%3Dhttp%253A%252F%252Floc
SHA1%26oauth_timestamp%3D1272323042%26oauth_version%3D1.0
```

Since this request doesn't have an oauth_token or oauth_token_secret, we don't inlcude an oauth_token field in our base string and we don't use an oauth_token_secret when calculating the composite signing key. Our signing key is (notice the dangling ampersand at the end):

```
MCD8BKwGdgPHvAuvgvz4EQpqDAtx89grbuNMRd7Eh98&
```

We then use the composite signing key to create an oauth_signature from the signature base string by signing the base string using the composite signing key. The resultant oauth_signature is:

```
8wUi7m5HFQy76nowoCThusfgB+Q=
```

Now that we have our signature, we have everything we need to make the request to the endpoint `https://api.twitter.com/oauth/request_token`. Now we just generate an HTTP header called "Authorization" with the relevant OAuth parameters for the request:

```
OAuth oauth_nonce="QP70eNmVz8jvdPevU3oJD2AfF7R7odC2XJcn4XlZJqk",
oauth_callback="http%3A%2F
%2Flocalhost%3A3005%2Fthe_dance%2Fprocess_callback%3Fservice_provider_id%3D11",
oauth_signature_method="HMAC-SHA1", oauth_timestamp="1272323042",
oauth_consumer_key="GDdmIQH6jhtmLUypg82g",
oauth_signature="8wUi7m5HFQy76nowoCThusfgB%2BQ%3D", oauth_version="1.0"
```

When Twitter.com receives our request, it will respond with an `oauth_token`, `oauth_token_secret` (collectively, your "request token"), and a field called `oauth_callback_confirmed` that will have the value of "true" if it's understood your OAuth callback. The response to this particular request would have looked like:

```
oauth_token=8ldIZyxQeVrFZXFOZH5tAwj6vzJYuLQpl0WUEYtWc&
oauth_token_secret=x6qpRnlEmW9JbQn4PQVVeVG8ZLPEx6A0TOebgwcuA&
oauth_callback_confirmed=true
```

We're now going to store the oauth_token and oauth_token_secret for a short duration as we send the user off to the authentication server. We'll need this oauth_token and oauth_token_secret until we've exchanged them for an access token. Hold onto those.

Sending the user to authorization

This is the easiest part of the standard OAuth flow. From the previous step, you'll need only your `oauth_token` (the request token) to complete.

The authorize step is where you send the user to a page on Twitter.com that will allow them to grant your application privileges to use their account with the API. Typically, a web application will simply use a redirect to send the user to this location. A desktop application might just present the URL and ask the user to self-navigate to the destination.

The endpoint for the authorization url is **`https://api.twitter.com/oauth/authorize`**. It must have a single query parameter attached called *oauth_token* with the value set to the oauth_token you received in the request token step. In the example above, the oauth_token was "8ldIZyxQeVrFZXFOZH5tAwj6vzJYuLQpl0WUEYtWc". Using this token to generate an authorization URL results in:

```
http://api.twitter.com/oauth
/authorize?oauth_token=8ldIZyxQeVrFZXFOZH5tAwj6vzJYuLQpl0WUEYtWc
```

If the user hasn't logged into Twitter recently, they'll be asked to enter their account credentials. Otherwise, they'll be presented with a streamlined flow that will allow one-click authorization for your application. Once the user has granted access, control will be returned to your application by redirecting the user to your specified oauth_callback. If you're using a desktop application and the out-of-band-flow, the user will instead be presented with a PIN code that they'll be asked to enter into your application. We'll see how the PIN code is used in the next step.

If you are using the callback flow, your oauth_callback should have received back your oauth_token (the same that you sent, your "request token") and a field called the `oauth_verifier`. You'll need that for the next step.

Here's the response I received:

```
oauth_token=8ldIZyxQeVrFZXFOZH5tAwj6vzJYuLQpl0WUEYtWc&
oauth_verifier=pDNg57prOHapMbhv25RNf75lVRd6JDsni1AJJIDYoTY
```

Exchanging a request token for an access token

Whew! We're almost done getting everything we need to actually make API requests on a user's behalf. All the good stuff has happened: you've started the dance by asking for a request token. You sent the user to authorize and then whatever happened, happened... Now you need an access token to continue. You'll need your request token (oauth_token and oauth_token_secret) and oauth_verifier from the last few steps.

Twitter's access token end point is **`https://api.twitter.com/oauth/access_token`**

Here are all the parameters that are going to go into our access token exchange request:

- **`oauth_consumer_key`** - GDdmIQH6jhtmLUypg82g
- **`oauth_nonce`** - 9zWH6qe0qG7Lc1telCn7FhUbLyVdjEaL3MO5uHxn8
- **`oauth_signature_method`** - HMAC-SHA1
- **`oauth_token`** - 8ldIZyxQeVrFZXFOZH5tAwj6vzJYuLQpl0WUEYtWc
- **`oauth_timestamp`** - 1272323047
- **`oauth_verifier`** - pDNg57prOHapMbhv25RNf75lVRd6JDsni1AJJIDYoTY
- **`oauth_version`** - 1.0

And though it is only used in the signing part of the request, our oauth_token_secret is still **`x6qpRnlEmW9JbQn4PQVVeVG8ZLPEx6A0TOebgwcuA`**.

We're going to perform another SSL call using POST. First we prepare our signature base string:

```
POST&https%3A%2F
%2Fapi.twitter.com%2Foauth%2Faccess_token&oauth_consumer_key%3DGDdmIQH6jhtmLUypg82
SHA1%26oauth_timestamp%3D1272323047%26oauth_token%3D8ldIZyxQeVrFZXFOZH5tAwj6vzJYuL
```

We create a composite signing key using both our oauth_consumer_secret and our oauth_token_secret (request token secret) by joining them with an ampersand:

```
MCD8BKwGdgPHvAuvgvz4EQpqDAtx89grbuNMRd7Eh98&x6qpRnlEmW9JbQn4PQVVeVG8ZLPEx6A0TOebgw
```

Then we sign our request, with the resultant OAuth signature:

```
PUw/dHA4fnlJYM6RhXk5IU/0fCc=
```

Now that we have our oauth_signature, we're ready to send the POST on to Twitter.com, so we create our HTTP Authorization header again using the relevant OAuth parameters, including the oauth_token (request token) we are exchanging for the access tokens. Your resource requests are going to look very similar to this one.

```
OAuth oauth_nonce="9zWH6qe0qG7Lc1telCn7FhUbLyVdjEaL3MO5uHxn8",
oauth_signature_method="HMAC-SHA1", oauth_timestamp="1272323047",
oauth_consumer_key="GDdmIQH6jhtmLUypg82g",
oauth_token="8ldIZyxQeVrFZXFOZH5tAwj6vzJYuLQpl0WUEYtWc",
oauth_verifier="pDNg57prOHapMbhv25RNf75lVRd6JDsni1AJJIDYoTY",
oauth_signature="PUw%2FdHA4fnlJYM6RhXk5IU%2F0fCc%3D", oauth_version="1.0"
```

When this handshake completes, Twitter will respond with more URL-encoded parameters, including the screen name and id of the user who just authorized, and an oauth_token and oauth_token_secret (collectively, your "access token"). You'll want to store these away. You can use an access token until the member severs the connection.

The response in this example was:

```
oauth_token=819797-Jxq8aYUDRmykzVKrgoLhXSq67TEa5ruc4GJC2rWimw&
oauth_token_secret=J6zix3FfA9LofH0awS24M3HcBYXO5nI1iYe8EfBA&user_id=819797&
screen_name=episod
```

I'm now going to use this access token to post a status on Twitter.com.

Making a resource request on a user's behalf

So now we're going to make a tweet. We're going to use the oauth_token and oauth_token_secret from the last step and tweet using some special characters just to make things more complicated for you.

This is what we're going to tweet: "**setting up my twitter 私のさえずりを設定する**"

This uses UTF-8 characters to tweet in both English and Japanese.

While posting this tweet, we use the resource URL **`http://api.twitter.com/1/statuses/update.json`** and the POST method. Here are the OAuth related params I'm going to use and the actual status update:

- **`POST body`** - status=setting+up+my+twitter+私のさえずりを設定する
- **`oauth_consumer_key`** - GDdmIQH6jhtmLUypg82gる
- **`oauth_nonce`** - oElnnMTQIZvqvlfXM56aBLAf5noGD0AQR3Fmi7Q6Y
- **`oauth_signature_method`** - HMAC-SHA1
- **`oauth_token`** - 819797-Jxq8aYUDRmykzVKrgoLhXSq67TEa5ruc4GJC2rWimw
- **`oauth_timestamp`** - 1272325550
- **`oauth_version`** - 1.0

And now, the deja vu experience. You're going to create a signature base string, following all the same rules as before -- except, we're using UTF-8 characters here and need to properly URL-encode them. My base string becomes:

```
POST&http%3A%2F
%2Fapi.twitter.com%2F1%2Fstatuses%2Fupdate.json&oauth_consumer_key%3DGDdmIQH6jhtmL
SHA1%26oauth_timestamp%3D1272325550%26oauth_token%3D819797-
Jxq8aYUDRmykzVKrgoLhXSq67TEa5ruc4GJC2rWimw%26oauth_version%3D1.0%26status%3Dsettin
```

Now I'm going to create a composite signing key using both the oauth_consumer_secret and the oauth_token_secret tied to my access token:

```
MCD8BKwGdgPHvAuvgvz4EQpqDAtx89grbuNMRd7Eh98&J6zix3FfA9LofH0awS24M3HcBYXO5nI1iYe8Ef
```

And after using that secret to sign my base string, I get the following signature:

```
yOahq5m0YjDDjfjxHaXEsW9D+X0=
```

And now I need to create my HTTP Authorization header, which of course means all those values need to be URL encoded..

```
OAuth oauth_nonce="oElnnMTQIZvqvlfXM56aBLAf5noGD0AQR3Fmi7Q6Y",
oauth_signature_method="HMAC-SHA1", oauth_timestamp="1272325550",
oauth_consumer_key="GDdmIQH6jhtmLUypg82g", oauth_token="819797-
Jxq8aYUDRmykzVKrgoLhXSq67TEa5ruc4GJC2rWimw",
oauth_signature="yOahq5m0YjDDjfjxHaXEsW9D%2BX0%3D", oauth_version="1.0"
```

And I send that POST to end point and hope everything works. On a successful tweet, my response includes the usual payload for a successful tweet -- the tweet itself:

```
{
    "geo":null,
    "created_at":"Mon Apr 26 23:45:50 +0000 2010",
    "in_reply_to_user_id":null,
    "truncated":false,
    "place":null,
    "source":"<a href=\"http://dev.twitter.com\" rel=\"nofollow\">OAuth
Dancer</a>",
    "favorited":false,
    "in_reply_to_status_id":null,
    "contributors":null,
    "user":
    {
        "contributors_enabled":false,
        "created_at":"Wed Mar 07 22:23:19 +0000 2007",
        "description":"Reality Technician,
         Developer Advocate at Twitter,
         Invader from the Space.",
        "geo_enabled":true,
        "notifications":false,
        "profile_text_color":"000000",
        "following":false,
        "time_zone":"Pacific Time (US & Canada)",
        "verified":false,
        "profile_link_color":"731673",
        "url":"http://bit.ly/5w7P88",
        "profile_background_image_url":"http://a3.twimg.com
/profile_background_images/19651315/fiberoptics.jpg",
        "profile_sidebar_fill_color":"007ffe",
        "location":"San Francisco,
         CA",
        "followers_count":1220,
        "profile_image_url":"http://a3.twimg.com/profile_images/836683953
/zod_normal.jpg",
        "profile_background_tile":true,
        "profile_sidebar_border_color":"bb0e79",
        "protected":false,
        "friends_count":1275,
        "screen_name":"episod",
        "name":"Taylor Singletary",
        "statuses_count":5733,
        "id":819797,
        "lang":"en",
        "utc_offset":-28800,
        "favourites_count":89,
        "profile_background_color":"000000"
    },
    "in_reply_to_screen_name":null,
    "coordinates":null,
    "id":12912397434,
    "text":"setting up my twitter \u79c1\u306e\u3055\u3048\u305a\u308a\u3092
\u8a2d\u5b9a\u3059\u308b"
}
```

We'll talk about when things go wrong in a moment..

OAuth Glossary

- **oauth_nonce** - a unique identifier for your request, generated by you. Twitter will only allow a nonce to be used once by your application. Prevents replayed requests.
- **oauth_timestamp** - an integer representing the number of seconds that have passed since the unix epoch.
- **oauth_verifier** - a string sent to you on your OAuth callback or provided to a user on the authentication flow, depending on whether you are using out-of-band mode or not. It is required to be sent back on the access token step.
- **out of band mode** - Instead of providing a URL-based callback when acquiring a request token, "oob" is supplied. Once the user has given Twitter their account credentials, they are presented with a screen containing a PIN code and are asked to enter this code into the application. The application then sends this PIN as an *oauth_verifier* to the access token step to complete the exchange.
- **signature base string** - an assembled string that is signed using a signing key to create a signature. In OAuth 1.0A,

the signature base string is computed as:

```
httpMethod + "&" +
url_encode(  base_uri ) + "&" +
sorted_query_params.each  { | k, v |
    url_encode ( k ) + "%3D" +
    url_encode ( v )
}.join("%26")
```

Query parameters in this case would include both query parameters passed to the base_uri on the query string or in URL-encoded post bodies, as well as all relevant OAuth parameters pertinent to the request in motion, sorted in lexicographical order. If multiple keys are present with differing values, they should be sorted by value first. POST bodies that are not of the URL-encoded type are not calculated as part of the OAuth signature base string.

- **signing key** - the string used as the "secret key" to sign a request. In OAuth 1.0A, the signing key is computed with the following simple algorithm:

```
url_encode( consumer_secret ) + "&" +
url_encode( oauth_token_secret || nil )
```

Some OAuth requests involve an oauth_token_secret, and some do not. The signing key always is the composite of both your consumer_secret and an oauth_token_secret, separated by an ampersand. When there is no oauth_token_secret, you still must create a composite signing key including the ampersand followed by "nothing".

- © 2010 Twitter
- About Us
- Contact
- Blog
- Status
- API Status
- Goodies
- Business
- Help
- Jobs
- Terms
- Privacy
- @TwitterAPI