



Socrata was acquired by Tyler Technologies in 2018 and is now the Data and Insights division of Tyler. The platform is still powered by the same software formerly known as Socrata but you will see references to Data & Insights going forward.

[Learn more...](https://www.tylertech.com/solutions/transformational-technology/data-insights)

(<https://www.tylertech.com/solutions/transformational-technology/data-insights>)

# Authentication

## Overview

[API Endpoints \(/docs/endpoints\)](/docs/endpoints)

[Row Identifiers \(/docs/row-identifiers\)](/docs/row-identifiers)

[RESTful Verbs \(/docs/verbs\)](/docs/verbs)

[Application Tokens \(/docs/app-tokens\)](/docs/app-tokens)

[Authentication \(/docs/authentication\)](/docs/authentication)

[Response Codes & Headers \(/docs/response-codes\)](/docs/response-codes)

[System Fields \(/docs/system-fields\)](/docs/system-fields)

[CORS & JSONP \(/docs/cors-and-jsonp\)](/docs/cors-and-jsonp)

## Filtering & Querying

[Simple Filters \(/docs/filtering\)](/docs/filtering)

[SoQL Queries \(/docs/queries/\)](/docs/queries/)

[Paging Through Data \(/docs/paging\)](/docs/paging)

[SoQL Function and Keyword Listing \(/docs/functions/\)](/docs/functions/)

[Data Transform Functions \(/docs/transforms/\)](/docs/transforms/)

## Data Formats (/docs/formats/)

[JSON \(/docs/formats/json\)](/docs/formats/json)

[GeoJSON \(/docs/formats/geojson\)](/docs/formats/geojson)

[CSV \(/docs/formats/csv\)](/docs/formats/csv)

[RDF-XML \(/docs/formats/rdf-xml\)](/docs/formats/rdf-xml)

## Datatypes (/docs/datatypes/)

[Checkbox \(/docs/datatypes/checkbox\)](/docs/datatypes/checkbox)

[Fixed Timestamp \(/docs/datatypes/fixed\\_timestamp\)](/docs/datatypes/fixed_timestamp)

[Floating Timestamp \(/docs/datatypes/floating\\_timestamp\)](/docs/datatypes/floating_timestamp)

[Line \(/docs/datatypes/line\)](/docs/datatypes/line)

[Location \(/docs/datatypes/location\)](/docs/datatypes/location)

[MultiLine \(/docs/datatypes/multiline\)](/docs/datatypes/multiline)

[MultiPoint \(/docs/datatypes/multipoint\)](/docs/datatypes/multipoint)

[MultiPolygon \(/docs/datatypes/multipolygon\)](/docs/datatypes/multipolygon)

[Number \(/docs/datatypes/number\)](/docs/datatypes/number)

[Point \(/docs/datatypes/point\)](/docs/datatypes/point)

[Polygon \(/docs/datatypes/polygon\)](/docs/datatypes/polygon)

[Text \(/docs/datatypes/text\)](/docs/datatypes/text)

[URL \(/docs/datatypes/url\)](/docs/datatypes/url)

## Other APIs (/docs/other/)



*Wait a second!* Authentication is only necessary when accessing datasets that have been marked as *private* or when making write requests ( PUT , POST , and DELETE ). For reading datasets that have not been marked as private, simply use an application token

(/docs/app-tokens).

There are two methods available for authentication: HTTP Basic and OAuth 2.0. For non-interactive applications, we only support HTTP Basic Authentication. We encourage all our developers of interactive applications to use the OAuth 2.0 workflow to authenticate their users.

- HTTP Basic Authentication is required when you are authenticating from a script that runs without interaction with the user, like your ETL tool, an update script, or any other data management automation.
- OAuth 2.0 is the preferred option for cases where you are building a web or mobile application that needs to perform actions on behalf of the user, like accessing data, and the interaction model allows you to present the user with a form to obtain their permission for the app to do so.

## Authenticating using HTTP Basic Authentication

Requests can be authenticated using HTTP Basic Authentication ([https://en.wikipedia.org/wiki/Basic\\_access\\_authentication](https://en.wikipedia.org/wiki/Basic_access_authentication)). You can use your HTTP library's Basic Auth feature to pass your credentials. All HTTP-basic-authenticated requests must be performed over a secure (https) connection. Authenticated requests made over an insecure connection will be denied.

Users may use their username and password or an API key and secret pair to authenticate using Basic Authentication. Documentation on how to create and manage API keys can be found here (</docs/other/api-keys>).

We recommend using API keys! They provide the following benefits:

- Access Socrata APIs without the risk of embedding your username and password in scripts or code
- Users on domains that require SSO (and thus without passwords) can access Socrata APIs
- Create individual keys for different apps or jobs so that if any one needs to be revoked or rotated, other apps are unaffected
- Change your account password without disrupting apps or rotate API keys without disrupting logins

Here is a sample HTTP session that uses HTTP Basic Authentication:

```
POST /resource/4tka-6guv.json HTTP/1.1
Host: soda.demo.socrata.com
Accept: */*
Authorization: Basic [REDACTED]
Content-Length: 253
Content-Type: application/json
X-App-Token: [REDACTED]

[ {
  ...
} ]
```

Note that the `Authorization` header in this request will usually be generated via your HTTP library's Basic Auth feature (as opposed to manually constructing the Base64 encoding of your credentials yourself). For example, if you're using Python's `requests` module, it supports Basic Authentication (<https://requests.readthedocs.io/en/master/user/authentication/#basic-authentication>) out of the box. Similarly, an API tool like Postman (<https://learning.postman.com/docs/getting-started/introduction/>) also handles Basic Authentication (<https://learning.postman.com/docs/sending-requests/authorization/#basic-auth>).

## OAuth 2.0

*Note:* When developing applications that make use of OAuth, you must provide a web-accessible callback URL when registering your application token. This can make it difficult to develop on a machine that isn't directly exposed to the Internet. One great option is to use a tool like ngrok (<https://ngrok.com>) to create a secure tunnel to expose your web application in a secure manner.

## Workflow

We support a subset of OAuth 2.0 (<https://en.wikipedia.org/wiki/Oauth>) – the server-based flow with a callback URL – which we believe is more secure than the other flows in the specification. This OAuth flow is used by several other popular API services on the web. We have made the authentication flow similar to Google AuthSub (<https://code.google.com/apis/gdata/docs/auth/authsub.html>).

To authenticate with OAuth 2.0, you will first need to register your application (<https://support.socrata.com/hc/en-us/articles/210138558-Generating-an-App-Token>), which will create an app token and a secret token. When registering your application, you must preregister your server by filling out the `Callback Prefix` field), so that we can be sure that access through your application is secure even if both your tokens are stolen. The `Callback Prefix` is the beginning of the URL that you will use as your redirect URL. Generally, you'll want to provide as much of your callback URL as you can. For example, if your authentication callback is `https://my-website.com/socrata-app/auth/callback`, you might want to specify `https://my-website.com/socrata-app` as your callback URL.

Once you have an application and a secret token, you'll be able to authenticate with the SODA OAuth 2.0 endpoint. You'll first need to redirect the user to the Socrata-powered site you wish to access so that they may log in and approve your application. For example:

```
https://soda.demo.socrata.com/oauth/authorize?client_id=YOUR_AUTH_TOKEN&response_type=code &redirect_uri=YOUR_REDIRECT_URI
```

Note that the `redirect_uri` here must be an absolute, secure ( `https:` ) URI which starts with the `Callback Prefix` you specified when you registered your application. If any of these cases fail, the user will be shown an error indicating as much.

Should the user authorize your application, they will be redirected back to the your `redirect_uri` . For example, if I provide `https://my-website.com/socrata-app/auth/callback` as my `redirect_uri` , the user will be redirected to this URL:

```
https://my-website.com/socrata-app/auth/callback?code=CODE
```

where `CODE` is an authorization code that you will use later.

If your `redirect_uri` contains a querystring, it will be preserved, and the `code` parameter will be added onto the end of it. Likewise, if you provide the optional `state` parameter in the original redirect to `/authenticate` , it will be preserved and sent back to you.

Now that the user has authorized your application, the next step is to retrieve an `access_token` so that you can perform operations on their behalf. You can do this by making the following `POST` request from your server:

```
https://soda.demo.socrata.com/oauth/access_token
?client_id=YOUR_AUTH_TOKEN
&client_secret=YOUR_SECRET_TOKEN
&grant_type=authorization_code
&redirect_uri=YOUR_REDIRECT_URI
&code=CODE
```

where `YOUR_AUTH_TOKEN` and `YOUR_SECRET_TOKEN` are the tokens you received when registering your app, `YOUR_REDIRECT_URI` is the same value as what you used previously, and `CODE` is the value of the `code` query parameter of the URL that the user was redirected to.

You'll receive the following response:

```
{ access_token: ACCESS_TOKEN }
```

Use this `access_token` in your requests when you have to do work on behalf of the now-authenticated user, as described below in the **Using an OAuth 2.0 Access Token** section.

We have a sample app ([https://github.com/socrata/oauth\\_sample\\_app\\_ruby](https://github.com/socrata/oauth_sample_app_ruby)) available on GitHub that illustrates how to do all of the above with the Ruby `OAuth2` gem.

## Using an OAuth 2.0 Access Token

Once you have obtained an `access_token` , you should include it on requests which need to happen on behalf of the user. The token must be included in the `Authorization` HTTP Header field as follows:

```
Authorization: OAuth YOUR_ACCESS_TOKEN
```

**Note:** All authenticated requests *must* be performed over a secure connection ( `https` ). Any attempt to use an `access_token` over a non-secure connection will result in immediate revocation of the token.

## Who am I?


One quirk of authenticating via OAuth 2.0 is that the entire process happens without the 3rd party application (that's you!) having any knowledge of who, exactly, the user is that just authorized the application. To remedy this, we have set up an endpoint that simply returns the information of the current user. To return the data in JSON:

```
https://soda.demo.socrata.com/api/users/current.json
```

To return the data in XML:

```
https://soda.demo.socrata.com/users/current.xml
```

---

 ([http://creativecommons.org/licenses/by-nc-sa/3.0/deed.en\\_US](http://creativecommons.org/licenses/by-nc-sa/3.0/deed.en_US)) Licensed by Tyler Technologies (<https://www.tylertech.com/solutions/transformational-technology/data-insights>) under CC BY-NC-SA 3.0 ([http://creativecommons.org/licenses/by-nc-sa/3.0/deed.en\\_US](http://creativecommons.org/licenses/by-nc-sa/3.0/deed.en_US)). Learn how you can contribute! (/contributing)