Tarbell Documentation

Release 0.8

News Apps and David Eads

November 19, 2013

Contents

1 Building a project **3**
1.1 Install Tarbell . . . 3
1.2 Create a Project . . . 3
1.3 Build a Project . . . 9
1.4 Publish a Project . . . 12
1.5 Reference . . . 13

The Tarbell template uses Python Flask and Google Spreadsheets to create simple, static sites that can be baked out to Amazon S3 or your local filesystem.

Tarbell is named after Ida Tarbell, a distinguished muckraking journalist whose 1904 The History of the Standard Oil Company is a masterpiece of investigative reporting. Read more about her on Wikipedia.

CHAPTER 1

Building a project

1.1 Install Tarbell

Clone repository, install virtual environment, install requirements, configure your system for Amazon S3, and run a test server.

Tarbell is a Python library based on Flask which powers static sites. Truth be told, it doesn't do much on its own except read a directory and render templates in any subdirectory it finds a `config.py` file. To see Tarbell in action, you should probably start with the Tarbell template, which sets up an Amazon S3 publishing workflow and basic framework for building modern web apps using Tarbell.

Make sure you have `python` (2.6+), `git`, `pip`, `virtualenv` and `virtualenv-wrapper` installed on your system.

```
git clone https://github.com/newsapps/tarbell
cd tarbell
mkvirtualenv tarbell
pip install -r requirements.txt
python runserver.py
```

Now visit http://localhost:5000/readme in your browser. You should see the latest version of this page.

1.1.1 How do I install these tools on my system?

For a very basic guide, see the Chicago Birthrates installation docs.

For more detailed, Mac-specific information, see Brian Boyer's Lion dev environment notes.

1.2 Create a Project

Get the client_secrets.json file if you don't have it already. Use the fab newproject command to kick off a new project by copying a basic project structure and setting up a Google spreadsheet.

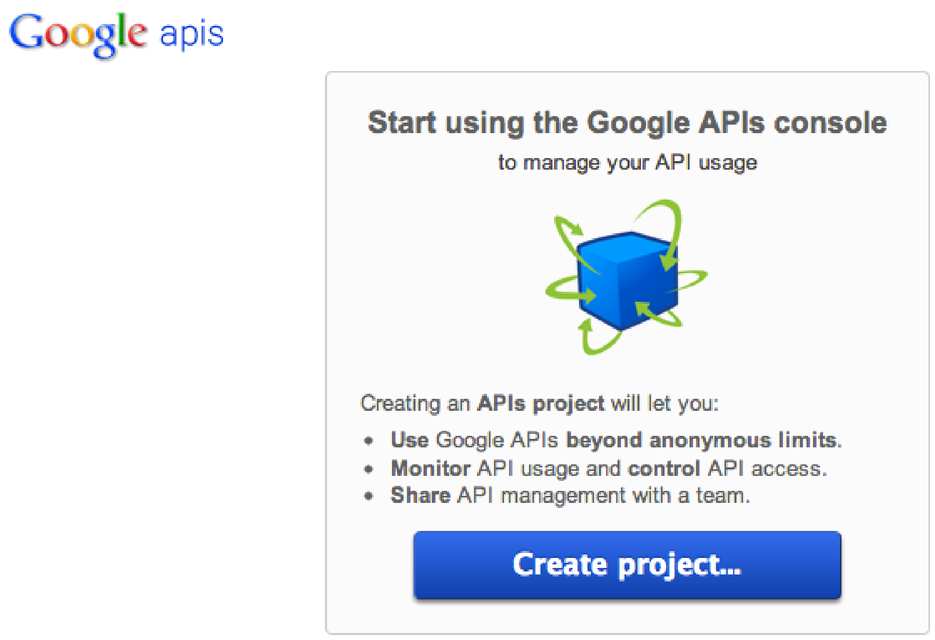
1.2.1 Prerequisite: Authenticating with Google with client_secrets.json

Tarbell uses the Google Drive API to create new spreadsheets, which requires going through a little OAuth2 song-and-dance. This is optional but highly recommended, in part because Tarbell will probably use this technique for all authentication and access in the future. If you want to skip this step and configure your spreadsheet manually, see Manually creating Google spreadsheets.

You ready? Let's go.

In order to allow Tarbell to create new Google Spreadsheets, you'll need to download a client_secrets.json file to access the Google Drive API. You can share this file with collaborators and within your organization, but do not share this file anywhere public.

Log in to the Google API Developer Console and create a new project:



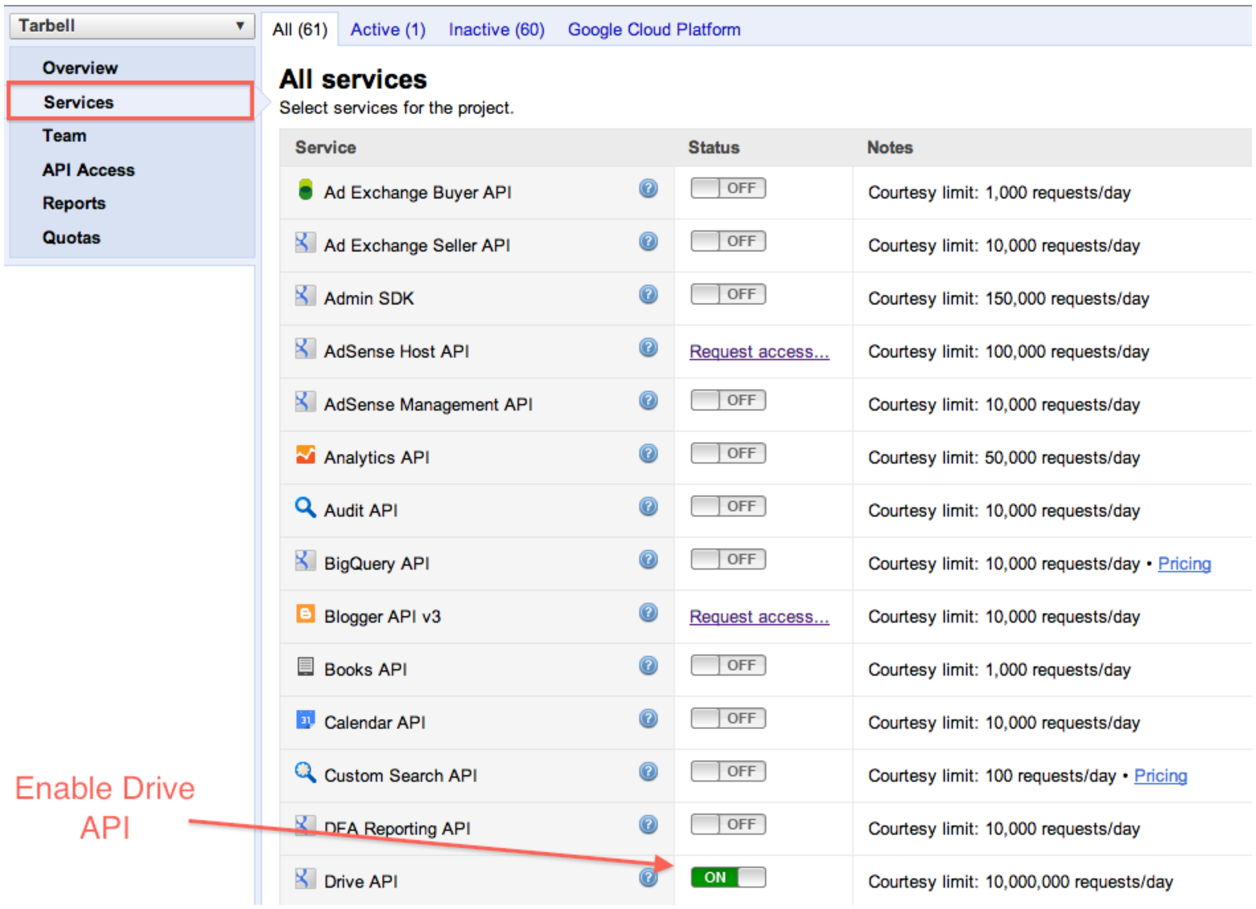
Start using the Google APIs console

to manage your API usage

Creating an **APIs project** will let you:

- **Use** Google APIs **beyond anonymous limits.**
- **Monitor** API usage and **control** API access.
- **Share** API management with a team.

Now click the "Services" tab and enable Google Drive API.

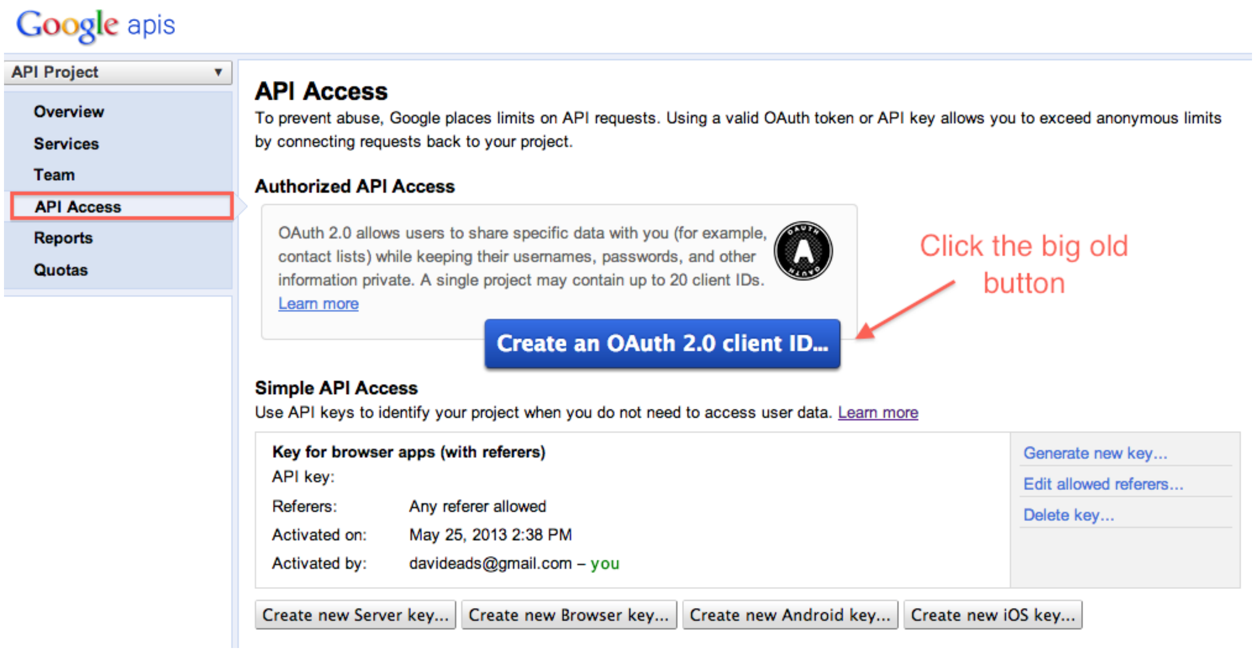


All services

Select services for the project.

Service	Status	Notes
Ad Exchange Buyer API	OFF	Courtesy limit: 1,000 requests/day
Ad Exchange Seller API	OFF	Courtesy limit: 10,000 requests/day
Admin SDK	OFF	Courtesy limit: 150,000 requests/day
AdSense Host API	Request access...	Courtesy limit: 100,000 requests/day
AdSense Management API	OFF	Courtesy limit: 10,000 requests/day
Analytics API	OFF	Courtesy limit: 50,000 requests/day
Audit API	OFF	Courtesy limit: 10,000 requests/day
BigQuery API	OFF	Courtesy limit: 10,000 requests/day • Pricing
Blogger API v3	Request access...	Courtesy limit: 10,000 requests/day
Books API	OFF	Courtesy limit: 1,000 requests/day
Calendar API	OFF	Courtesy limit: 10,000 requests/day
Custom Search API	OFF	Courtesy limit: 100 requests/day • Pricing
DFA Reporting API	OFF	Courtesy limit: 10,000 requests/day
Drive API	ON	Courtesy limit: 10,000,000 requests/day

Click the "API Access" tab to create a client ID:



API Access

To prevent abuse, Google places limits on API requests. Using a valid OAuth token or API key allows you to exceed anonymous limits by connecting requests back to your project.

Authorized API Access

OAuth 2.0 allows users to share specific data with you (for example, contact lists) while keeping their usernames, passwords, and other information private. A single project may contain up to 20 client IDs. Learn more

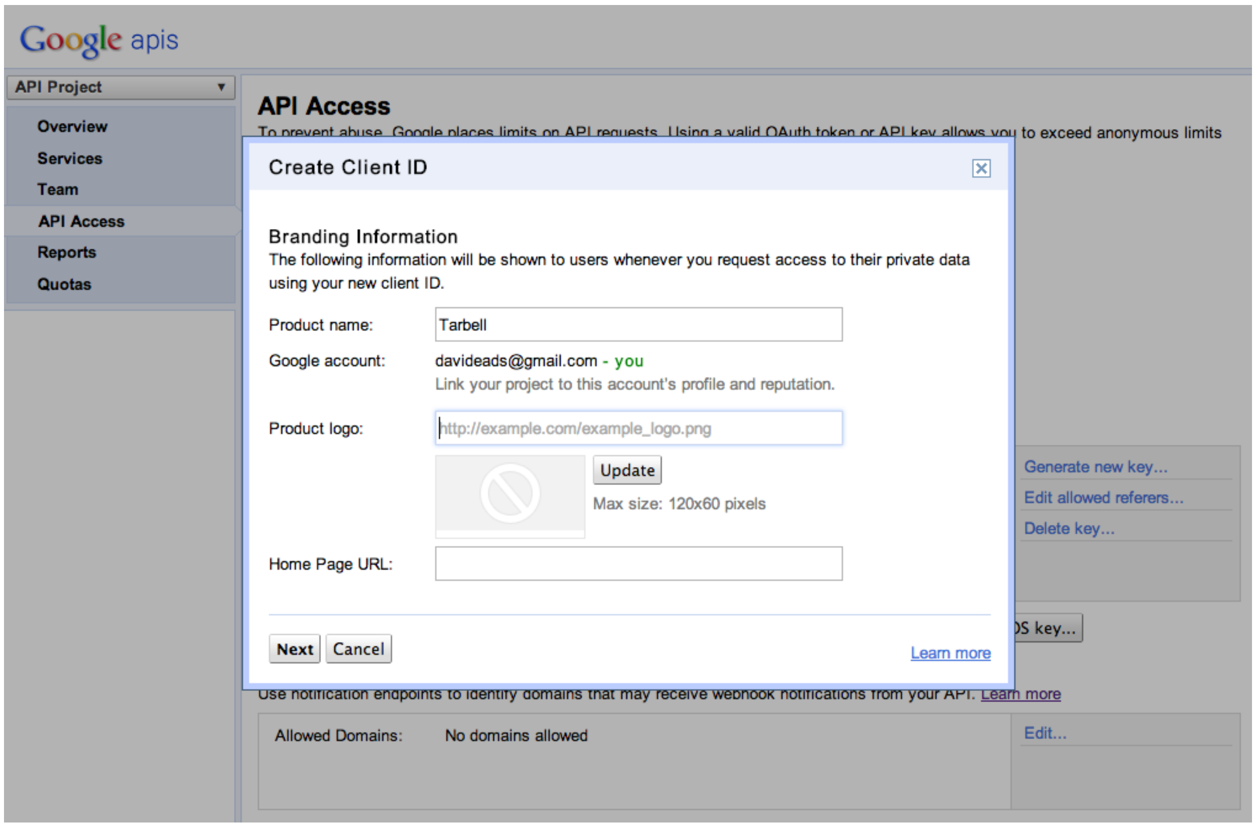
Simple API Access

Use API keys to identify your project when you do not need to access user data. Learn more

Key for browser apps (with referers)

API key:	
Referers:	Any referer allowed
Activated on:	May 25, 2013 2:38 PM
Activated by:	davideads@gmail.com – you

Add some project details. These don't really matter:



Create Client ID

Branding Information

The following information will be shown to users whenever you request access to their private data using your new client ID.

Product name: Tarbell

Google account: davideads@gmail.com - you

Link your project to this account's profile and reputation.

Product logo: http://example.com/example_logo.png

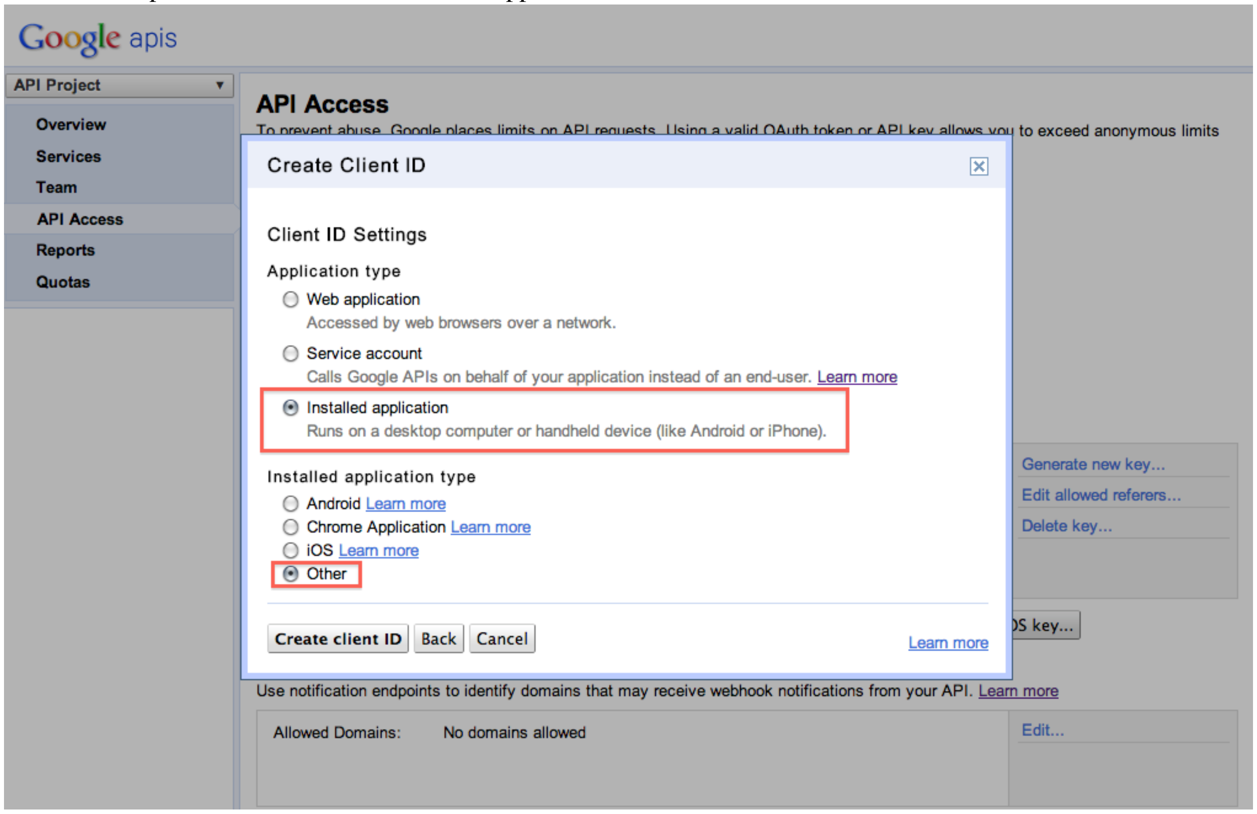
Max size: 120x60 pixels

Home Page URL:

Use notification endpoints to identify domains that may receive webhook notifications from your API. Learn more

Allowed Domains: No domains allowed

This is the important screen. Select "installed app" and "other":



Create Client ID

Client ID Settings

Application type

Web application — Accessed by web browsers over a network.

Service account — Calls Google APIs on behalf of your application instead of an end-user. Learn more

[✓ Installed application — Runs on a desktop computer or handheld device (like Android or iPhone).]

Installed application type

Android Learn more

Chrome Application Learn more

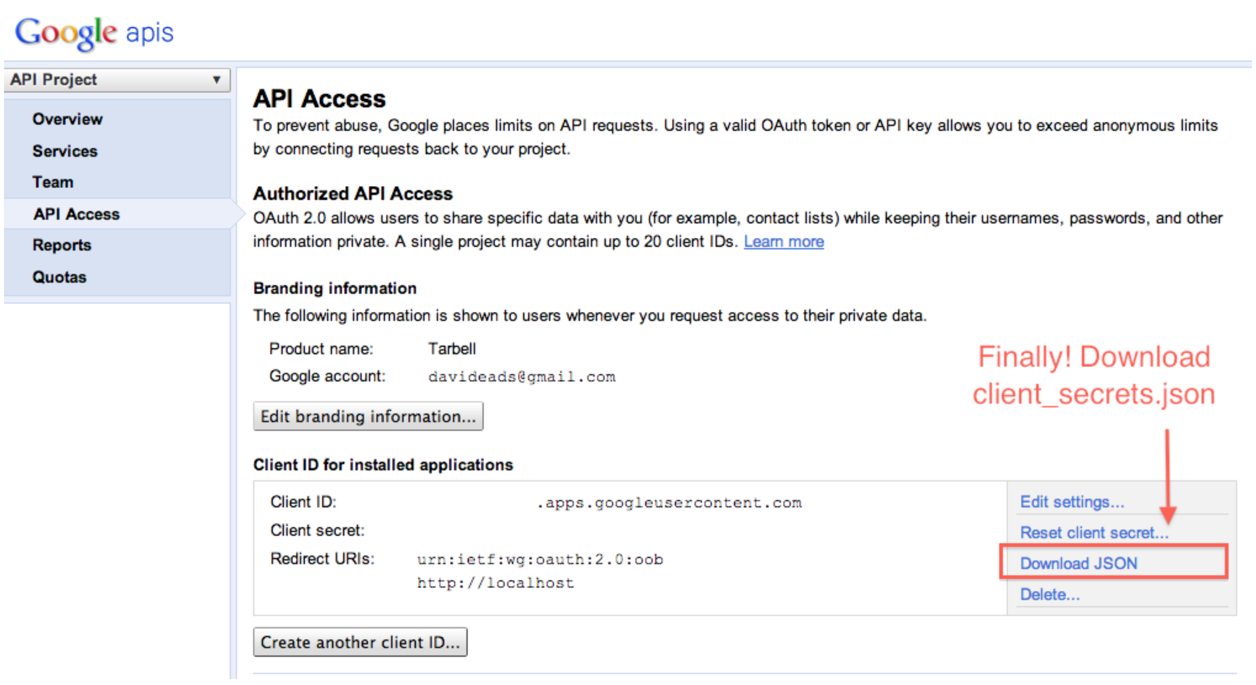
iOS Learn more

[✓ Other]

Use notification endpoints to identify domains that may receive webhook notifications from your API. Learn more

Allowed Domains: No domains allowed

Whew! Now you can download the `client_secrets.json` file:



API Access

To prevent abuse, Google places limits on API requests. Using a valid OAuth token or API key allows you to exceed anonymous limits by connecting requests back to your project.

Authorized API Access

OAuth 2.0 allows users to share specific data with you (for example, contact lists) while keeping their usernames, passwords, and other information private. A single project may contain up to 20 client IDs. Learn more

Branding information

The following information is shown to users whenever you request access to their private data.

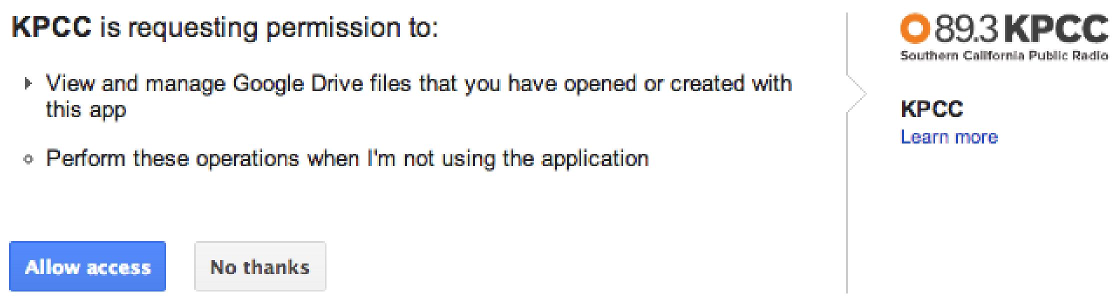
Product name:	Tarbell
Google account:	davideads@gmail.com

Client ID for installed applications

Client ID:	.apps.googleusercontent.com
Client secret:	
Redirect URIs:	urn:ietf:wg:oauth:2.0:oob http://localhost

Now put the file in the root directory of your Tarbell installation.

The first time you run `fab newproject` and answer yes to create a Google spreadsheet, your default browser will open and you will be prompted to grant your Tarbell client access to your API key.



KPCC is requesting permission to:

- View and manage Google Drive files that you have opened or created with this app
- Perform these operations when I'm not using the application

The `fab newproject` command will prompt you if the `client_secrets.json` file doesn't exist.

The first time you create a new project and spreadsheet, make sure you are not running any services on port 8080, such as MAMP. The Python Google API client library fires up a tiny little server on port 8080 to receive and store an access token during this cycle. Because the access token is stored, you won't need to do again unless your token is revoked. You can restore any port 8080 services indefinitely.

Help us improve! We know this step is a little rocky. We'd like to make it smoother. If you are an OAuth or Google Drive API expert, we need your help. See #21 Improve OAuth workflow for newproject command and #22 Use Drive API in Tarbell library.

1.2.2 Create a project

To create your first project, use the handy `fab` command:

```
fab newproject
```

You’ll be prompted with a series of questions. Here’s what you’ll see the first time you it with user input highlighted.

```
What is the directory name for the project? awesomeproject
What is your project’s full title? Awesome project
Do you want a Google doc associated with this project? [Y/n]: y
Generating Google spreadsheet
What Google account should have access to this spreadsheet initially? (e.g. my.name@gmail.com) somebo
Authenticating your Google account to use Tarbell. If any services are running on
port 8080, disable them and run this command again.

Your browser has been opened to visit:

    https://accounts.google.com/o/oauth2/auth?scope=https%3A%2F%2Fwww.googleapis.com%2Fauth%2Fdrive.

If your browser is on a different machine then exit and re-run this
application with the command-line parameter

    --noauth_local_webserver

Authentication successful.
Success! View the spreadsheet at https://docs.google.com/spreadsheet/ccc?key=BIGLONGSPREADSHEETKEY90

This spreadsheet is published in public on the web. To make it private
you’ll need to configure the project’s secrets.py file, disable
publishing using the ’Publish to the web’ settings from the file menu,
and share the document with the account specified in secrets.py.

Created /Users/davideads/Repos/tarbell/awesomeproject/config.py
Created /Users/davideads/Repos/tarbell/awesomeproject/secrets.py
Created directory /Users/davideads/Repos/tarbell/awesomeproject/static/css
Created /Users/davideads/Repos/tarbell/awesomeproject/static/css/style.css
Created directory /Users/davideads/Repos/tarbell/awesomeproject/static/js
Created /Users/davideads/Repos/tarbell/awesomeproject/static/js/app.js
Created directory /Users/davideads/Repos/tarbell/awesomeproject/templates
Created /Users/davideads/Repos/tarbell/awesomeproject/templates/index.html
Would you like to create a new branch and initial commit for this project? [Y/n]: y
[localhost] local: git checkout master;
git checkout -b awesomeproject
M   fabfile.py
M   readme/docs/create.md
Already on ’master’
M   fabfile.py
M   readme/docs/create.md
Switched to a new branch ’awesomeproject’
[localhost] local: git add awesomeproject
[localhost] local: git commit -m "Started new project awesomeproject"
[awesomeproject cc2502a] Started new project awesomeproject
 5 files changed, 212 insertions(+), 0 deletions(-)
 create mode 100644 awesomeproject/config.py
 create mode 100644 awesomeproject/secrets.py
 create mode 100644 awesomeproject/static/css/style.css
 create mode 100644 awesomeproject/static/js/app.js
 create mode 100644 awesomeproject/templates/index.html

Welcome to Awesome project. Great work! What’s next?
```

```
- Edit awesomeproject/config.py to set up template values and adjust project settings.
- Edit awesomeproject/secrets.py to configure Google spreadsheet authentication variables.
- Edit awesomeproject/templates/index.html to edit your default template.
- Edit awesomeproject/static/js/app.js to edit your default Javascript app.
- Run `python runserver.py` and view your project at http://localhost:5000/awesomeproject/

Run `fab deploy` and `fab project:projectname deploy` to deploy to S3 if you have a bucket configured

Done.
```

1.2.3 Manually creating Google Spreadsheets

To manually set up a Google spreadsheet for your project:

- Create a new Google spreadsheet
- Rename "Sheet1" to "values"
- Add 'key' and 'value' column headers in the first row
- Add the spreadsheet key in projectname/config.py
- **Public access:**
  - Set the spreadsheet to 'publish to the web'
- **Private access:**
  - Grant access to a special user account (you'll be storing password in the clear, so set up a new account for this)
  - Add credentials to projectname/secrets.py

1.3 Build a Project

Project layout, edit templates and manage Google spreadsheet, tweak CSS, and take a peek at the Javascript app.

Now that you've created a new project, let's look at how Tarbell projects are constructed.

1.3.1 Project layout

A Tarbell template project directory structure looks like this:

- `config.py`: Configuration file. Required to detect the project.
- `secrets.py`: Set `GOOGLE_AUTH` variable to configure authentication. Not tracked by Git.
- **`templates`: The templates directory contains Jinja templates that will be published at `/projectname/TEMPLATENAM`**
  - `index.html`: A basic template to start building with.
- **`static`: The static directory contains static assets like images, CSS, and Javascript. They are published at `/projectna`**
  - `js/app.js`: An skeleton Javascript application for your project that is automatically loaded by base template.
  - `css/style.css`: An empty stylesheet for your project.

1.3.2 What's the difference between static assets and templates?

Static assets are simply served as-is, while templates are provided with context variables and rendered using Jinja.

1.3.3 Editing templates

Every file that ends in `.html` in `projectname/templates` will be published to `projectname/TEMPLATENAME.html` and can be previewed at http://localhost:5000/projectname/TEMPLATENAME.html.

Template basics

Tarbell uses Jinja2 for templating and supports all Jinja2 features.

A basic template looks like:

```
{% extends '_base.html' %}

{% block css %}
{{ super() }} {# Load base styles #}
<link rel="stylesheet" type="text/css"
    href="{{ static_url('MYPROJECT', '/css/style.css') }}" />
{% endblock css %}

{% block content %}
<h1>{{ title }}</h1>
<p class="credit">{{ credit }}</p>
{{ body|process_text }}
{% endblock content %}
```

1.3.4 What's `_base.html`?

The Tarbell template comes with a base template file that sets up some simple blocks and manages Javascript app loading.

The `static_url()` template function

The `static_url(projectname, path)` function constructs the path to an asset stored under `projectname/static` based on the project's output URL.

Working with Google spreadsheets: The "values" worksheet

The **values** worksheet must have "key" and "value" columns. These key-value pairs will be provided as global variables to templates. So if there's a row with a key column value of "foo" and a value of "bar", `{{ foo }}` in a template will print `bar`.

Working with Google spreadsheets: Other worksheets

Other worksheets can hold freeform data, namespaced by the worksheet name. Unlike the **values** worksheet, data in these worksheets can be accessed by iterating through a list or, if a column named "key" is present, by reference to the value in that column. Some examples with a worksheet named **updates** should help make this clear.

A worksheet called "updates"

key	title	date	url
hadiya	Hadiya's friends	05-05-2013	http://graphics.chicagotribune.com/hadiyas-friends
grace	His Saving Grace	02-14-2013	http://graphics.chicagotribune.com/grace

Get worksheet values in template

The worksheet will be passed to your context as an iterable list, with each column in the worksheet representing a separate item in the context dictionary. So in your template, the following code displays the contents of each row in your spreadsheet:

```
{% for row in updates %}
<p> <a href="{{ row.url }}">{{ row.title }}</a> </p>
{% endfor %}
```

Directly accessing a row

If there's a header named "key" that contains only unique, simple string values we can directly access individual rows in that worksheet:

```
<p> <a href="{{ updates.grace.url }}">{{ updates.grace.title }}</a> </p>
```

1.3.5 Editing Javascript app

Every project comes with a barebones Javascript app in `projectname/static/js/app.js`.

The app uses RequireJS and provides Backbone, jQuery, and Underscore libraries by default.

Wrap your app code in a `require(['dependency', ...], function(DepObj) { ... })` call to include necessary libraries and modules.

```
// Additional RequireJS configuration
require.config( {
    paths: {
        moment: '//cdnjs.cloudflare.com/ajax/libs/moment.js/2.0.0/moment.min',
    },
} );

// Start our project's app
require([ 'jquery', 'base/views/NavigationView', 'moment' ],
function($, NavigationView, moment) {
    console.log("Creating navigation view");
    var nav = new NavigationView({
        el: $('#header'),
        title: { label: 'Tarbell Readme', url: '#top' },
    }).render();

    console.log("Demonstrating momentJS:");
    console.log(new moment());
});
```

1.4 Publish a Project

Use fab deploy and fab project:<projectname> deploy to upload your project to Amazon S3. Customize the publishing process.

1.4.1 Amazon S3 setup

An Amazon S3 publishing workflow is included in the Tarbell template. To use it, you'll need your Amazon S3 credentials.

Create a file called `s3config.py` in your Tarbell template directory.

```
S3CONFIG = {
    'BUCKETNAME': {
        'bucket': 'mybucket.domain.com',
        'key': 'KEY',
        'key_id': 'KEYID',
    }
}
```

1.4.2 Help! I don't have an Amazon S3 account.

Amazon S3 is simply online file storage – think of it as FTP on steroids. Setting up an Amazon S3 account is easy. Just check out this beginners guide. If you want to use your S3 "bucket" as a website, read Amazon's guide to S3 website hosting.

1.4.3 Deploying

Once your Amazon S3 access credentials are configured, deploying all projects is very simple:

```
fab target:BUCKETNAME deploy
```

This will deploy to the bucket specified by BUCKETNAME in s3config.py.

To simplify deploying to the bucket named production, simply run:

```
fab deploy
```

When deploying you'll see something like:

```
[localhost] local: python render_templates.py
Rendering templates.

Generating project 'base' in /Users/davideads/Repos/tarbell/out/
-- No Google doc configured for base.

Generating project 'readme' in /Users/davideads/Repos/tarbell/out/readme
-- Created JSON /Users/davideads/Repos/tarbell/out/readme/json/values.json
-- Created JSON /Users/davideads/Repos/tarbell/out/readme/json/LAST_UPDATED.json
-- Created JSON /Users/davideads/Repos/tarbell/out/readme/json/projects.json
-- Created page /Users/davideads/Repos/tarbell/out/readme/index.html

[localhost] local: python s3deploy.py
Deploying to tarbell.tribapps.com
Uploading css/style.css
Uploading js/app.js
Uploading js/templates/nav.jst
Uploading js/views/NavigationView.js
Uploading readme/index.html
Refreshing Facebook info for: http://tarbell.tribapps.com/readme/index.html?fbrefresh=CANBEANYTHING
Uploading readme/bootstrap/css/bootstrap.css
Uploading readme/bootstrap/css/bootstrap.min.css
Uploading readme/bootstrap/img/glyphicons-halflings-white.png
Uploading readme/bootstrap/img/glyphicons-halflings.png
Uploading readme/bootstrap/js/bootstrap.js
Uploading readme/bootstrap/js/bootstrap.min.js
Uploading readme/css/ir_black.css
Uploading readme/css/style.css
Uploading readme/img/google-screenshot.jpg
Uploading readme/img/html-edit-screenshot.jpg
Uploading readme/img/ida-tarbell.jpg
Uploading readme/img/s3-publish-screenshot.jpg
Uploading readme/js/app.js
Uploading readme/json/LAST_UPDATED.json
Uploading readme/json/projects.json
Uploading readme/json/values.json
```

To deploy a specific project, use the `project:PROJECTNAME` flag:

```
fab project:PROJECTNAME deploy
```

In the following example, we'll publish a project called `basketball` using a bucket configuration named `sports`:

```
fab project:basketball target:sports deploy
```

Please note: The base template is always published – it is assumed most projects will use some base components.

1.5 Reference

Configure Tarbell, set up a Flask Blueprint, special base project.

1.5.1 Configuring Tarbell

When your project was created, a `config.py` file was created in the project directory, which lets Tarbell find your project. This file can be empty, but also accepts several configuration options:

- `GOOGLE_DOC`: A dict of Google docs parameters to access a spreadsheet.

  Takes `key`, `account`, and `password` parameters.

  The default template stores account and password variables in a file called `secrets.py` in variable called `GOOGLE_AUTH`. **Use secrets.py to keep your authentication information out of version control.**

  ```
  GOOGLE_DOC = {
      'key': "BIGLONGSTRINGOFLETTERSANDNUMBERS",
      'account': "some+account@gmail.com",
      'password': "++GmailPassWord++",
  }
  ```

- `DEFAULT_CONTEXT`: Default context variables to make available to all project templates.

```
DEFAULT_CONTEXT = {
    'ad_path': '',
    'analytics_path': '',
}
```

- `DONT_PUBLISH`: If `True`, this project will not be published to S3.

```
DONT_PUBLISH=True
```

Default: `False`

- `URL_ROOT`: Override the published URL to differ from the directory name.

```
URL_ROOT='totally-awesome-project'
```

Default: `None` (publish using name of directory)

- `CREATE_JSON`: If `False`, do not publish JSON data. Useful if spreadsheets contain secrets or sensitive information, and so should not be public.

```
CREATE_JSON = False
```

Default: `True`

For advanced uses, you can turn your project into a Flask Blueprint in order to register template filters or dynamically set the template context.

```
from flask import Blueprint
blueprint = Blueprint('awesome_project', __name__)

# Register template filter
@blueprint.app_template_filter('my_filter')
def my_filter(text):
   return text.strip()

@blueprint.app_context_processor
def context_processor():
    """
    Add "my_variable" to context
    """
    context = {
        'my_variable': 'My variable would be more awesome in real life, like reading a file or API da
    }

    return context
```

Now you can reference `{{ my_variable }}` in your templates, or call your filter on a template variable `{{ my_variable|my_filter }}`.

1.5.2 Base project

If any project contains a `URL_ROOT = ''` configuration, that project will:

- Be available at the root URL (`/index.html`, `/css/style.css`, etc).
- Always be published when deploying.

1.5.3 JSON publishing

By default, every project's Google spreadsheet will be baked out to a JSON file representing each worksheet. For example, most projects will have a `myproject/json/values.json` that represents the contents of the "values" worksheet.

This means you can build pure Javascript apps using Tarbell in the framework of your choice. Just AJAX load or bootstrap the JSON data.

To disable this behavior, add a line to your `config.py`:

```
CREATE_JSON = False
```

If you disable this behavior and need data available to Javascript applications, simply bootstrap the dataset provided it isn't too big. Here's something you might put in `myproject/index.html`:

```
{% block scripts %}
<script type="text/javascript">
    // Convert whole worksheet to JSON
    var authors = {{ authors|tojson }}

    // Filter a worksheet
    var locations = [ {% for address in locations %}
        { state: '{{ address.state }}' },
    {% endfor %} ];

    // Now process or display 'authors' and 'locations' ...
</script>
{% endblock %}
```