latimes-table-stacker Documentation

Release 0.3

Los Angeles Times Data Desk

November 20, 2014

Contents

Publish spreadsheets as interactive tables. And do it on deadline.

CHAPTER 1

Features

- Convert a CSV file into an interactive HTML table that sorts, filters and paginates.
- Quickly create as static files to serve on the web.
- Sync static files with Amazon S3 for instant publishing.
- Syndicate data as CSV, XLS and JSON.
- Post an RSS feed and sitemap that promote the latest data.

CHAPTER 2

In the wild

- White-label deployment
- Everything at spreadsheets.latimes.com
- Numerous tables published by the Orlando Sentinel
- Census data from The New Jersey Star Ledger
- Campaign contributions at MinnPost.com
- Super PAC contributions from California Watch

2.1 Documentation

2.1.1 Getting started

This tutorial will walk you through the process of installing Table Stacker and publishing an example.

Requirements

- git
- python

01. Install the code on your computer

It's not required, but I recommend creating a virtual environment to store your application. I like to do this with the Python module virtualenv, which creates a walled-off garden for the Python code to work without distraction from the outside world. If you don't have it, you'll need to install it now, which just might be as easy as

```
$ pip install virtualenv
# Or maybe ...
$ sudo easy_install install virtualenv
# Or, if you're in Ubuntu ...
$ sudo apt-get install python-virtualenv
```

Once you have virtualenv installed, make it happen by navigating to wherever you keep your code and firing off the following. I'm going to call this project `my-table-stacker`, but you should substitute whatever you're calling your version.

```
$ virtualenv --no-site-packages my-table-stacker
```

Now jump into the directory it creates.

```
$ cd my-table-stacker
```

Activate the private environment with virtualenv's custom command.

```
$ . bin/activate
```

Download the latest version of the code repository into a directory called `project`.

```
$ git clone git://github.com/datadesk/latimes-table-stacker.git project
```

And jump in and get ready to work.

```
$ cd project
```

Install our app's Python dependencies.

```
$ pip install -r requirements.txt
```

Create the project's database

```
$ python manage.py syncdb
$ python manage.py migrate
```

02. Build the example tables

You'll learn how to layout your own data later, but for now we'll work with the example files. Jump back to your first terminal shell and drop the following line, which instructs our `build` management command to bake out a static site using the instructions in `settings.py` and the table recipes in the `yaml` directory.

```
$ python manage.py build
```

03. Launch the static version of the site

You'll want to run this step in a new terminal shell. So open up a new window or tab, navigate to the `project` directory and fire off the following. It is a Django management command that will start a test version of the site on your machine, tailored to serve the static files we used created.

```
$ python manage.py buildserver
```

04. Check it out

If everything clicked, you should see your demo site up and running with all the example tables at http://localhost:8000.

05. Deploy your app

The static files we've created in your `build` directory could probably be served from most common web servers. So, if you've already got yours worked out, you can just stop here and deploy that folder where you like.

However, the app is prepared to help you easily deploy to Amazon S3. To make that happen, you'll need to do a little set up. First, go to aws.amazon.com/s3 and set up an account. Then you'll need to create a bucket for storing our files. If you need help there are some basic instructions here.

Next configure the bucket to act as a website. Amazon's official instructions say to do the following:

```
In the bucket Properties pane, click the Website configuration tab.

Select the Enabled check box.

In the Index Document Suffix text box, add the required index document name (index.html).
```

Before you leave that pane, note the URL at the bottom. This is where your site will be published.

Now, set your bucket name in the *settings.py* file.:

```
AWS_BUCKET_NAME = 'table-stacker'
```

Next, install s3cmd, a utility we'll use to move files back and forth between your desktop and S3. In Ubuntu, that's as simple as:

```
$ sudo apt-get install s3cmd
```

If you're Mac or Windows, you'll need to download the file and follow the installation instructions you find there.

Once it's installed, we need to configure s3cmd with your Amazon login credentials. Go to Amazon's security credentials page and get your access key and secret access key. Then, from your terminal, run

```
$ s3cmd --configure
```

Finally, now that everything is set up, publishing your files to s3 is as simple as:

```
$ python manage.py publish
```

Once you do that, your site should appear at the the link provided in your AWS console. If you want to bind that to a subdomain of your site, say, www.tablestacker.com, you need to create a new CNAME record in your domain's DNS registration. You also need the name of your bucket to line up with the subdomain. Don't take it from me. Read the detailed instructions provided by Amazon.

```
For example, if you have registered domain, www.example-bucket.com, you
could create a bucket www.example-bucket.com, and add a DNS CNAME entry
pointing to www.example-bucket.com.s3-website-<region>.amazonaws.com.
All requests to http://www.example-bucket.com will be routed to
www.example-bucket.com.s3-website-<region>.amazonaws.com.
```

More documentation on that is available here.

06. Publish you own data table

Before you can publish your own data table, you'll need to learn about our YAML-based configuration system. But don't worry, it's not that hard. You can read about it in the configuration section or school yourself by mimicking the examples files in the project's `yaml` subdirectory folder. Then, doing the following:

```
$ python manage.py build
$ python manage.py publish
```

2.1.2 Configuration

Each published table is drawn from a CSV file you provide and styled according to the rules outlined in a configuration file written in YAML configuration file. CSV files are stored in the `csv` folder in the root directory. YAML configuration files are stored in the `yaml` folder, with one configuration per file.

Example

Here is an example YAML configuration that specifies how to layout this demonstration table.

```
table:
  title: Major U.S. coal mines, 2009
  file:  major-us-coal-mines-2009.csv
  slug: major-us-coal-mines-2009
  byline: Ben Welsh
  description: <p>A list of the largest coal-producing U.S. mines for the year 2009. The U.S. Energy
  column_options:
    columns:
      - Mine
      - Company
      - Type
      - State
      - Production (Short tons)
    style:
      Mine: 'text-align:left; width:250px;'
      Company: 'text-align:left; width:250px;'
      Type: "width:80px;"
      State: "width:100px;"
    sorted_by:
      - Production (Short tons): descending
    formatting:
      Production (Short tons):
        method: intcomma
  is_published: true
  publication_date: 2011-01-12
  sources: <a href="http://www.eia.doe.gov/cneaf/coal/page/acr/acr_sum.html">U.S. Energy Information
  credits: <a href="mailto:ben.welsh@latimes.com">Ben Welsh</a>
```

Metadata Options

The following YAML configuration options detail how to present a number of attributes about the table. All entries should be placed inside a dictionary titled `table`.

title
The headline that will appear in lists and at the top of the table's detail page. Required.

```
title: Major U.S. coal mines, 2009
```

file
The name of the CSV file the table will be based on. It should be in the `csv` directory with a header row included. Required.

```
file: major-us-coal-mines-2009.csv
```

slug
A string that serves as the unique identifier of the table in the database and doubles as the relative url of its web page. It cannot be used for more than one table in your database. It's recommended that you do not use spaces or strange characters. Required.

```
file: major-us-coal-mines-2009
```

byline
The name or list of names that will appear as a byline in lists and on the table's detail page. Optional.

```
byline: Bob Woodard and Carl Bernstein
```

description
A block of text describing the table that will appear above the table on its detail page. HTML can and should be included. Optional.

```
description: <p>A list of the largest coal-producing U.S. mines for the year 2009.</p>
```

kicker
A brief string to run above the headline in all capital letters. "SPREADSHEET" by default. Optional.

```
kicker: data table
```

legend
A slot above the table where you can stick an HTML block containing a legend. Empty be default. Optional.

```
legend: "<img src='http://example.com/legend.png'>"
```

footer
A slot below the table where you can stick and HTML block containing footnotes, corrections or other extra information. Optional.

```
footer: "<p>We regret the error.</p>"
```

is_published
A boolean `true` or `false` that indicates whether the table should be published. If set to `false`, the table will be loaded in the database but will not appear on the site. Required.

```
is_published: true
```

publication_date
The date that will appear alongside with the byline. Should be provided in `YYYY-MM-DD` format. Required.

```
publication_date: 2011-01-12
```

publication_time
The time that will appear alongside the date with the byline. Should be provided in `HH:MM:SS` format. Optional.

```
publication_time: "11:58:00"
```

sources
A block of text describing where the data came from. Will appear at the bottom of the table detail page after the phrase `Sources:`. HTML can and should be included. Optional.

```
sources: <a href="http://www.eia.doe.gov/cneaf/coal/page/acr/acr_sum.html">U.S. Energy Informati
```

credits
A block of text listing all the people who helped make the page. Will appear at the bottom of the table detail page after the phrase `Credits:`. HTML can and should be included. Optional.

```
credits: <a href="mailto:russ.stanton@latimes.com">Russ Stanton</a>
# Or ...
credits: Bob Woodward and Carl Bernstein
```

per_page
How many records should appear in each page of the data table. 50 by default. Optional.

```
per_page: 50
```

show_download_links

Whether download links for CSV, XLS and JSON data should be made available on the table detail page. The default is true, so you only need to include it when you want to turn downloads off.

```
show_download_links: false
```

show_search_field

Whether or not to show a search box on the table detail page that filters the table. The default is true, so you only need to include it when you want to turn the search off.

```
    show_search_field: false
```

show_in_feeds

Whether the table will show in the sitemap, RSS feeds and public-facing list pages. The default is true, so you only need to include it when you want to set it to false.

```
show_in_feeds: false
```

Column Options

The following YAML configuration options specify how to present the columns in the data table. They should appear as entries in a dictionary titled `column_options`.

columns

A list of the columns from the CSV that should appear in the published table. They will appear in the order specified here. Key names should correspond to headers in the CSV file. Optional.

```
columns:
  - Mine
  - Company
  - Type
  - State
  - Production (Short tons)
```

style

A dictionary that specifies custom CSS to be applied to columns in the data table. CSS declarations should be included just as they would in an HTML `style` attribute. Key names should correspond to headers in the CSV file. Optional.

```
style:
  Mine: 'text-align:left; width:250px;'
  Company: 'text-align:left; width:250px;'
  Type: "width:80px;"
  State: "width:100px;"
```

sorted_by

A single item list that specifies which column that table should be sorted by default, and which directions. Key names should correspond to headers in the CSV file. The direction can be either `ascending` or `descending`. Optional.

```
sorted_by:
  - Production (Short tons): descending
```

sorters

A dictionary that specifies how to properly sort columns in the interactive table. The JavaScript library that crafts the table attempts to guess the proper sorting method for each column, but sometimes it is wrong. Other times you might not want to sort a column at all, which can be done by setting the value to `false`. You can use these options to declare what you'd like it do. A full list of the available sorters can be found here. Optional.

```
sorters:
  Production (Short tons): fancyNumber
  Name: false
```

formatting

A dictionary that specifies formatting methods to be applied to all rows in a particular column. Each entry should include the column's name, followed by a dictionary requesting a particular method and, if necessary, customization options and other columns to be passed in as arguments. Optional.

```
formatting:
  Employees Affected:
    method: intcomma
  Company Name:
    method: title
  Title:
    method: link
    argument: url
```

If you'd like to add a new filter of your own, open the `table_fu/formatting.py` file and add it there. Formatting filters are simple functions that accept a value and return the transformed value we'd like to present.

```python
def title(value):
    """
    Converts a string into titlecase.

    Lifted from Django.
    """
    value = value.lower()
    t = re.sub("([a-z])'([A-Z])", lambda m: m.group(0).lower(), value.title())
    return re.sub("\d([A-Z])", lambda m: m.group(0).lower(), t)
```

After you've written a new filter, add it to the DEFAULT_FORMATTERS dictionary in that same file and you should now be available for use in YAML configuration files.

Available formatting filters

ap_state(*value*)

Converts a state's name, FIPS code or postal abbreviation to A.P. style. Returns the submitted string if a conversion cannot be made.

```
formatting:
  ColumnName:
    method: ap_state
```

bubble(*value, yes_icon="/static/img/bubble_yes.png", no_icon="/static/img/bubble_no.png", empty="&mdash;"*)

Returns one of two "Consumer Reports" style bubbles that indicate yes (a filled bubble) or no (an empty bubble). The first letter of each type is what should be provided (i.e. Y, N). If a match cannot be made the empty argument is returned.

```
formatting:
  ColumnName:
    method: bubble
```

You can customize the output by overriding the defaults

```
formatting:
  ColumnName:
    method: bubble
    options:
      yes_icon: "http://example.com/yes.png"
      no_icon: "http://example.com/no.png"
```

capfirst (*value*)

Changes a string so that only the first character is capitalized.

```
formatting:
  ColumnName:
    method: capfirst
```

checkbox (*value*, *yes_icon='/static/img/checkbox_yes.png'*, *no_icon='/static/img/checkbox_no.png'*)

Returns one of two checkbox images that indicate yes (a checked box) or no (an empty box). The first letter of each type is what should be provided (i.e. Y, N). If a match cannot be made an empty string is returned.

```
formatting:
  ColumnName:
    method: checkbox
```

You can customize the output by overriding the defaults

```
formatting:
  ColumnName:
    method: checkbox
    options:
      yes_icon: "<img src='http://example.com/yes.png'>"
      no_icon: "<img src='http://example.com/no.png'>"
```

date_and_time (*value*, *formatting="N j, Y, h:i a"*)

Reformats a date string in a humanized format, AP style by default.

```
formatting:
  ColumnName:
    method: date_and_time
```

You can override the output format by specifying an alternative in the formatting in the options. You must use Django's datetime formatting style.

```
formatting:
  ColumnName:
    method: date_and_time
    options:
      formatting: "Y-m-d P"
```

dollar_signs (*value*)

Converts an integer into the corresponding number of dollar sign symbols (ie. 3 -> "$$$"). Meant to emulate the illustration of price range on Yelp. If something besides an integer is submitted, "N/A" is returned.

```
formatting:
  ColumnName:
    method: dollar_signs
```

dollars (*value*, *decimal_places=2*)

Converts an number to a string containing commas every three digits with a dollar sign at the front. Returns "N/A" if the something besides a number if submitted.

```
formatting:
  ColumnName:
    method: dollars
```

The number of decimal places the number is rounded at can controlled with an option. The default is two decimal places.

```
formatting:
  ColumnName:
    method: dollars
    options:
      decimal_places: 0
```

intcomma (*value*)

Converts an integer to a string containing commas every three digits.

```
formatting:
  ColumnName:
    method: intcomma
```

image (*value*, *width=''*, *height=''*)

Accepts a URL and returns an HTML image tag ready to be displayed.

```
formatting:
  ColumnName:
    method: image
```

Optionally, you can set the height and width with keyword arguments.

```
formatting:
  ColumnName:
    method: image
    options:
      height: "30px"
      width: "30px"
```

link (*title*, *url*)

Wraps a string in an HTML hyperlink. The URL from another column passed as an argument.

```
formatting:
  TextColumnName:
    method: link
    arguments:
      - LinkColumnName
```

percentage (*value*, *decimal_places=1*, *multiply=True*)

Converts a floating point value into a percentage value. An empty string is returned if the input triggers an exception.

```
formatting:
  ColumnName:
    method: percentage
```

The number of decimal places set by the `decimal_places` option. The default is one. Also by default the number is multiplied by 100. You can prevent it from doing that by setting the `multiply` option to False.

```
formatting:
  ColumnName:
    method: percentage
    options:
      decimal_places: 0
      multiply: false
```

percent_change (*value*, *decimal_places=1*, *multiply=True*)

Converts a float into a percentage value with a + or - on the front and a percentage sign on the back. "N/A" is returned if the input cannot be converted to a float.

```
formatting:
  ColumnName:
    method: percent_change
```

The number of decimal places set by the `decimal_places` option. The default is one. Also by default the number is multiplied by 100. You can prevent it from doing that by setting the `multiply` option to False.

```
formatting:
  ColumnName:
    method: percent_change
    options:
      decimal_places: 0
      multiply: false
```

short_ap_date (*value*, *date_format=None*)

Reformats a date string in an abbreviated AP format.

```
formatting:
  ColumnName:
    method: short_ap_date
```

The method tries to parse the datestring automatically, but in some cases (i.e. dates in the first century) or less common date formats you might need to specifiy the date format using strptime standards.

```
formatting:
  ColumnName:
    method: short_ap_date
    options:
      date_format: "%Y-%m-%d"
```

simple_bullet_graph (*actual*, *target*, *width='95%'*, *max=None*)

Renders a simple bullet graph that compares a target line against an actual value. Unlike a conventional bullet graph, it does not shade the background into groups. Instead, it's all one solid color.

```
formatting:
  ActualValueColumn:
    method: simple_bullet_graph
    arguments:
      - TargetValueColumn
    options:
      max: 60
```

title (*value*)

Converts a string into titlecase.

```
    formatting:
      ColumnName:
        method: title
```

tribubble (*value*, *yes_icon='/static/img/tribubble_yes.png'*, *partly_icon='/static/img/tribubble_partly.png'*, *no_icon="/static/img/tribubble_no.png"*, *empty="&mdash;"*)

Returns one of three "Consumer Reports" style bubbles that indicate yes (filled bubble), partly (half-filled bubble), no (empty bubble). The first letter of each type is what should be provided (i.e. Y, N, P). If a match cannot be made the empty argument is returned.

```
formatting:
  ColumnName:
    method: tribubble
```

You can customize the output by overriding the defaults

```
formatting:
  ColumnName:
    method: tribubble
    options:
      yes_icon: "http://example.com/yes.png"
      no_icon: "http://example.com/no.png"
      partly_icon: "http://example.com/partly.png"
```

vote(*value*, *yes_vote='/static/img/thumb_up.png'*, *no_vote='/static/img/thumb_down.png'*, *did_not_vote="<b style='font-size:130%;'>&mdash;</b>"*)

Returns one of three icons representing the outcome a vote: Yes (thumbs up); No (thumbs down); Did not vote (Bolded emdash). The first letter of each type is what should be provided, i.e. Y, N, anything else.

```
formatting:
  ColumnName:
    method: vote
```

You can customize the output by overriding the defaults

```
formatting:
  ColumnName:
    method: vote
    options:
      yes_vote: "<img src='http://example.com/yes.png'>"
      no_vote: "<img src='http://example.com/no.png'>"
      did_not_vote: "<img src='http://example.com/didnotvote.png'>"
```

Override sort with URL

You can override the default sorting order of a table by appending a query string argument that provides an alternative. There are two required parameters.

sortColumn
The index of the column that the table should sort by. Starts with zero from the left. So to sort by the first column, you would provide `0`.

sortOrder
The direction the column should sort in. `0` will sort in the ascending. `1` will sort in the descending.

Here is an example of resorting a table by the first column in ascending order.

```
http://table-stacker.s3-website-us-west-1.amazonaws.com/california-layoffs-december-2010/?sortColumn=
```

2.1.3 Management commands

Interactions with the Table Stacker database are handled using custom Django management commands that allow you to create, update and delete tables.

Like other Django commands, they are run by interacting with the `manage.py` file in your project's root directory.

build [options]
Builds a static site with all the tables okayed for publication

```
$ python manage.py build
```

`buildserver [options]`

Delete the table outlined in the configuration file provided by the first argument.

```
$ python manage.py buildserver
# Optionally, set the port for the server.
$ python manage.py buildserver 8080
```

`publish [options]`

Sync the build directory with the Amazon S3 bucket specified in settings.py

```
$ python manage.py publish
```

`unbuild [options]`

Empties the build directory

```
$ python manage.py unbuild
```

`unpublish [options]`

Empties the Amazon S3 bucket defined in settings.py

```
    $ python manage.py unpublish
```

2.1.4 Customization

Table Stacker is published with minimal styling. If you want to adapt it for your site, you'll probably want to change the appearance and layout. The CSS styles that regulate the appearance of Table Stacker are stored in the `/static/css` directory. Change them and you'll change the appearance of the site. Table Stacker's layout is managed using Django's templating system and configured through a series of files in the `templates` directory. Change them and you'll change the layout of the site.

Global settings

`SITE_NAME`

A `settings.py` configuration that sets the site's name in meta data around the site, like the title tag and Facebook open graph tags.

`FACEBOOK_ADMINS`

A list of Facebook user ids included in the open graph tags in each page's head. Useful for configuring the site's footprint on Facebook. Set in `settings.py`.

2.1.5 Credits

This project would not be possible without the generous work of people like:

- ProPublica's News Application Desk, and particularly Jeff Larson, who developed the Ruby libraries table-fu and table-setter.
- Chris Amico, who did the noble work of porting table-fu to Python.
- Christian Bach, the man who gave us tablesorter.
- Thomas Suh Lauder, who has suggested many style improvements and formatting options.

2.2 Contributing

- Code repository: https://github.com/datadesk/latimes-table-stacker
- Issues: https://github.com/datadesk/latimes-table-stacker/issues