
biweeklybudget Documentation

Release 0.1.2

Jason Antman

May 28, 2017

Contents

1	Overview	3
1.1	Important Warning	3
1.2	Main Features	3
2	Requirements	5
3	Installation	7
4	License	9
5	Contents	11
5.1	Screenshots	11
5.1.1	Index Page	11
5.1.2	Pay Periods View	12
5.1.3	Single Pay Period View	12
5.1.4	Accounts View	13
5.1.5	Account Details	13
5.1.6	OFX Transactions	14
5.1.7	Transactions View	14
5.1.8	Transaction Detail	15
5.1.9	Budgets	15
5.1.10	Single Budget View	16
5.1.11	Scheduled Transactions	16
5.1.12	Specific Date Scheduled Transaction	17
5.1.13	Monthly Scheduled Transaction	17
5.1.14	Number Per-Period Scheduled Transactions	18
5.1.15	Reconcile Transactions with OFX	18
5.1.16	Drag-and-Drop Reconciling	19
5.2	Getting Started	19
5.2.1	Requirements	19
5.2.2	Installation	20
5.2.3	Configuration	20
5.2.3.1	Settings Module	20
5.2.3.2	Environment Variables	20
5.2.4	Usage	20
5.2.4.1	Setup	20
5.2.4.2	Flask	21

	5.2.4.3	Command Line Entrypoints and Scripts	21
5.3		Docker	21
	5.3.1	Environment Variable File	21
	5.3.2	Containerized MySQL Example	22
	5.3.3	Host-Local MySQL Example	22
	5.3.4	Settings Module Example	22
	5.3.5	Note on Locales	23
	5.3.6	Running ofxgetter in Docker	23
5.4		Flask Application	23
	5.4.1	Running	23
	5.4.2	Security	23
5.5		OFX Transaction Downloading	24
	5.5.1	Important Note on Transaction Downloading	24
	5.5.2	ofxgetter entrypoint	24
	5.5.3	Vault Setup	24
	5.5.4	Configuring Accounts for Downloading with ofxclient	25
	5.5.5	Configuring Accounts for Downloading with Selenium	25
5.6		Getting Help	27
	5.6.1	Bugs and Feature Requests	27
5.7		Development	27
	5.7.1	Guidelines	28
	5.7.2	Loading Data	28
	5.7.3	Testing	28
	5.7.3.1	Unit Tests	28
	5.7.3.2	Integration Tests	28
	5.7.3.3	Acceptance Tests	28
	5.7.4	Alembic DB Migrations	29
	5.7.5	Database Debugging	29
	5.7.6	Docker Image Build	29
	5.7.7	Frontend / UI	29
	5.7.8	Release Checklist	29
5.8		Changelog	30
	5.8.1	0.1.2 (2017-05-28)	30
	5.8.2	0.1.1 (2017-05-20)	31
	5.8.3	0.1.0 (2017-05-07)	31
5.9		biweeklybudget	31
	5.9.1	biweeklybudget package	31
	5.9.1.1	Subpackages	31
	5.9.1.2	Submodules	43
5.10		UI JavaScript Docs	55
	5.10.1	Files	55
	5.10.1.1	jsdoc.budget_transfer_modal	55
	5.10.1.2	jsdoc.budgets_modal	56
	5.10.1.3	jsdoc.custom	56
	5.10.1.4	jsdoc.forms	57
	5.10.1.5	jsdoc.ofx	57
	5.10.1.6	jsdoc.payperiod_modal	58
	5.10.1.7	jsdoc.reconcile	58
	5.10.1.8	jsdoc.reconcile_modal	61
	5.10.1.9	jsdoc.scheduled_modal	61
	5.10.1.10	jsdoc.transactions_modal	61

Responsive Flask/SQLAlchemy personal finance app, specifically for biweekly budgeting.

For full documentation, see <http://biweeklybudget.readthedocs.io/en/latest/>

For screenshots, see <http://biweeklybudget.readthedocs.io/en/latest/screenshots.html>

For development activity, see <https://waffle.io/jantman/biweeklybudget>

CHAPTER 1

Overview

biweeklybudget is a responsive (mobile-friendly) Flask/SQLAlchemy personal finance application, specifically targeted at budgeting on a biweekly basis. This is a personal project of mine, and really only intended for my personal use. If you find it helpful, great! But this is provided as-is; I'll happily accept pull requests if they don't mess things up for me, but I don't intend on working any feature requests or bug reports at this time. Sorry.

The main motivation for writing this is that I get paid every other Friday, and have for almost all of my professional life. I also essentially live paycheck-to-paycheck; what savings I have is earmarked for specific purposes, so I budget in periods identical to my pay periods. No existing financial software that I know of handles this, and many of them have thousands of Google results of people asking for it; almost everything existing budgets on calendar months. I spent many years using Google Sheets and a handful of scripts to template out budgets and reconcile transactions, but I decided it's time to just bite the bullet and write something that isn't a pain.

Intended Audience: This is decidedly not an end-user application. You should be familiar with Python/Flask/MySQL. If you're going to use the automatic transaction download functionality, you should be familiar with [Hashicorp Vault](#) and how to run a reasonably secure installation of it. I personally don't recommend running this on anything other than your own computer that you physically control, given the sensitivity of the information. I also don't recommend making the application available to anything other than localhost, but if you do, you need to be aware of the security implications. This application is **not** designed to be accessible in any way to anyone other than authorized users (i.e. if you just serve it over the web, someone *will* get your account numbers, or worse).

Important Warning

This software should be considered *alpha* quality at best. At this point, I can't even say that I'm 100% confident it is mathematically correct, balances are right, all scheduled transactions will show up in the right places, etc. I'm going to be testing it for my own purposes, and comparing it against my manual calculations. Until further notice, if you decide to use this, please double-check *everything* produced by it before relying on its output.

Main Features

- Budgeting on a biweekly (fortnightly; every other week) basis, for those of us who are paid that way.

- Optional automatic downloading of transactions/statements from your financial institutions.

CHAPTER 2

Requirements

Note: Alternatively, biweeklybudget is also distributed as a [Docker container](#). Using the dockerized version will eliminate all of these dependencies aside from MySQL (which you can run in another container) and Vault (if you choose to take advantage of the OFX downloading), which you can also run in another container.

- Python 2.7 or 3.3+ (currently tested with 2.7, 3.3, 3.4, 3.5, 3.6 and developed with 3.6)
- Python [VirtualEnv](#) and `pip` (recommended installation method; your OS/distribution should have packages for these)
- Git, to install certain upstream dependencies.
- MySQL, or a compatible database (e.g. [MariaDB](#)). biweeklybudget uses [SQLAlchemy](#) for database abstraction, but currently specifies some MySQL-specific options, and is only tested with MySQL.
- To use the automated OFX transaction downloading functionality:
 - A running, reachable instance of [Hashicorp Vault](#) with your financial institution web credentials stored in it.
 - [PhantomJS](#) for downloading transaction data from institutions that do not support OFX remote access (“Direct Connect”).

CHAPTER 3

Installation

It's recommended that you install into a virtual environment (virtualenv / venv). See the [virtualenv usage documentation](#) for information on how to create a venv.

This app is developed against Python 3.6, but should work back to 2.7. It does not support Python3 < 3.3.

Please note that, at the moment, two dependencies are installed via git in order to make use of un-merged pull requests that fix bugs; since

```
git clone https://github.com/jantman/biweeklybudget.git && cd biweeklybudget
virtualenv --python=python3.6 .
source bin/activate
pip install -r requirements.txt
python setup.py develop
```

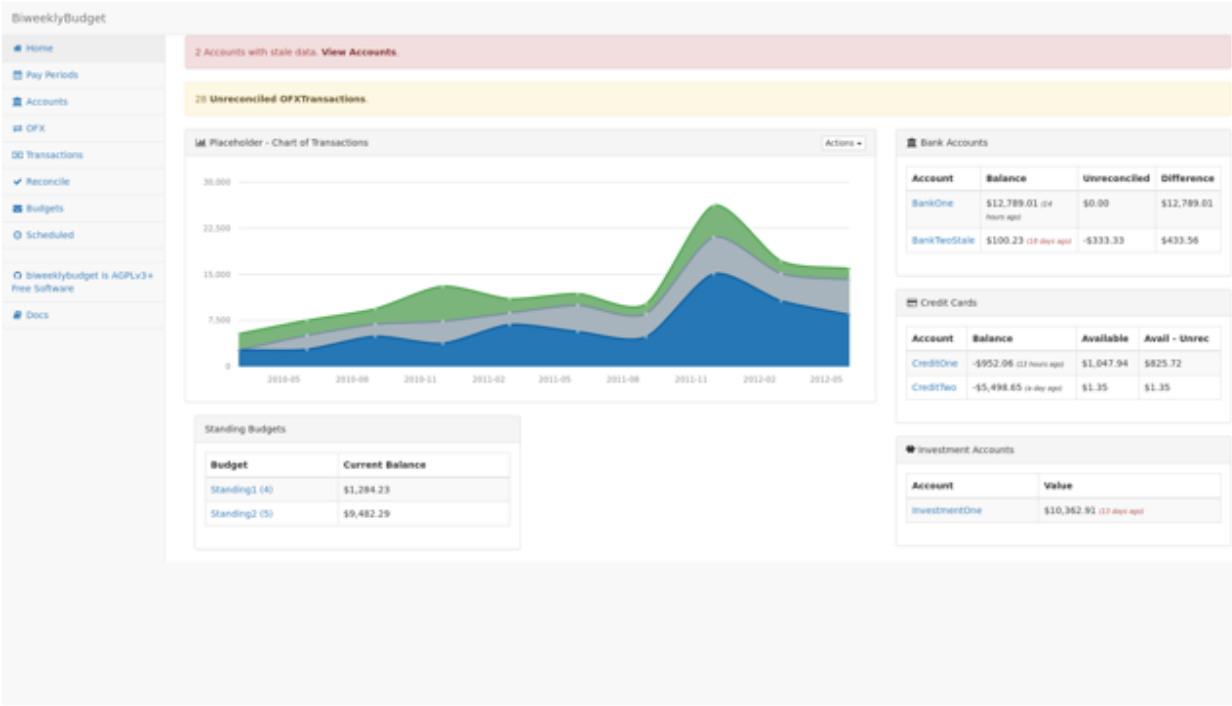

CHAPTER 4

License

biweeklybudget itself is licensed under the [GNU Affero General Public License, version 3](#). This is specifically intended to extend to anyone who uses the software remotely over a network, the same rights as those who download and install it locally. biweeklybudget makes use of various third party software, especially in the UI and frontend, that is distributed under other licenses. Please see `biweeklybudget/flaskapp/static` in the source tree for further information.

Screenshots

Index Page



Pay Periods View

Summary of previous, current and upcoming pay periods, plus date selector to find a pay period.

Pay Periods - BiweeklyBudget

2 Accounts with state data. [View Accounts.](#)

26 Unreconciled OFXTransactions.

\$1,790.12
Remaining this period

[View 2017-04-28 Pay Period](#)

\$2,011.67
Remaining next period

[View 2017-05-12 Pay Period](#)

\$2,011.67
Remaining following period

[View 2017-05-26 Pay Period](#)

Start Date	Allocated	Spent	Remaining
2017-04-14	\$334.00	\$0.00	\$2,011.67
2017-04-28 (current)	\$555.55	\$333.35	\$1,790.12
2017-05-12	\$334.00	\$0.00	\$2,011.67
2017-05-26	\$334.00	\$0.00	\$2,011.67

Find Pay Period

Date: [Go](#)

= March 2017 = = April 2017 = = May 2017 =
 Su Mo Tu We Th Fr Sa Su Mo Tu We Th Fr Sa Su Mo Tu We Th Fr Sa
 26 27 28 1 2 3 4 26 27 28 29 30 31 1 30 1 2 3 4 5 6
 5 6 7 8 9 10 31 2 3 4 5 6 7 8 7 8 9 10 11 12 13
 12 13 14 15 16 17 18 9 10 11 12 13 14 15 14 15 16 17 18 19 20
 19 20 21 22 23 24 25 16 17 18 19 20 21 22 21 22 23 24 25 26 27
 26 27 28 29 30 31 1 23 24 25 26 27 28 29 28 29 30 31 1 2 3
 2 3 4 5 6 7 8 30 1 2 3 4 5 6 4 5 6 7 8 9 10

Single Pay Period View

Shows a pay period (current in this example) balances (income, allocated, spent, remaining), budgets and transactions (previous/manually-entered and scheduled).

2017-04-28 to 2017-05-11 Pay Period - BiweeklyBudget

2 Accounts with state data. [View Accounts.](#)

26 Unreconciled OFXTransactions.

Remaining Balances

2017-04-14 (prev.)	2017-04-28 (curr.)	2017-05-12 (next)	2017-05-26	2017-06-09
\$2,011.67	\$1,790.12	\$2,011.67	\$2,011.67	\$2,011.67

\$2,345.67
Income

\$555.55
allocated

\$333.35
spent

\$1,790.12
remaining

Periodic Budgets

Budget	Amount	Allocated	Spent	Remaining
Periodic1	\$100.00	\$333.33	\$333.33	-\$11.33
Periodic2	\$234.00	\$444.44	\$222.22	-\$210.44
Income ()	\$2,345.67	\$0.00	\$0.00	\$2,345.67

Standing Budgets

Budget	Balance
Standing1	\$1,284.23
Standing2	\$9,482.29

Transactions [Add Transaction](#)

Date	Amount	Description	Account	Budget	Scheduled?	Reconciled?
2017-04-29	\$222.22	T3 (3)	CreditOne	Periodic2		
2017-05-01	-\$333.33	T2 (2)	BankTwoState	Standing1	(from J)	
2017-05-04	\$222.22	(sched) ST2 (2)	BankOne	Periodic2	make trans.	
2017-05-05	\$111.11	T1foo (1)	BankOne	Periodic1	(from J)	Yes (1)

Accounts View

Accounts - BiweeklyBudget

Home

Pay Periods

Accounts

QFX

Transactions

Reconcile

Budgets

Scheduled

Biweeklybudget is AGPLv3+ Free Software

Docs

2 Accounts with stale data. [View Accounts](#)

28 Unreconciled QFXTransactions

Bank Accounts

Account	Balance	Unreconciled	Difference
BankOne	\$12,789.01 (14 hours ago)	\$0.00	\$12,789.01
BankTwoState	\$100.23 (18 days ago)	-\$333.33	\$433.56

Credit Cards

Account	Balance	Credit Limit	Available	Unreconciled	Difference
CreditOne	-\$952.06 (13 hours ago)	\$2,000.00	\$1,047.94	\$222.22	\$825.72
CreditTwo	-\$5,498.65 (18 days ago)	\$5,500.00	\$1.35	\$0.00	\$1.35

Investment Accounts

Account	Value
InvestmentOne	\$10,362.91 (17 days ago)

Account Details

BankOne - BiweeklyBudget

Home

Pay Periods

Accounts

QFX

Transactions

Reconcile

Budgets

Scheduled

Biweeklybudget is AGPLv3+ Free Software

Docs

2 Accounts with stale data. [View Accounts](#)

28 Unreconciled QFXTransactions

BankOne (1)

Description	First Bank Account
Type	Bank
Ledger Balance	\$12,789.01
Available Balance	\$12,563.18
Last QFX Date	2017-04-30 11:09:16 (14 hours ago)
Active?	True
QFXGetter Config	{"fee": "bar"}
reconcile_trans	True
negate_ofx_amounts	False
ofx_cat_memo_to_name	True
Interest Charge regex	^((interest charge purchase finance charge)
Interest Paid regex	^interest paid
Payment regex	^((online payment internet payment online pymt payment)
Fee regex	^((late fee past due fee)

Details of a single account.

OFX Transactions

OFX Transactions - BiweeklyBudget

2 Accounts with stale data. [View Accounts.](#)

20 Unreconciled OFXTransactions.

Show 10 entries Account:

Date	Amount	Account	Type	Name	Memo	Description	FITID
2017-04-30	\$123.81	CreditOne (3)	Purchase	123.81 Credit Purchase T1	38328	CreditOneT1Desc	T1
2017-04-26	\$-50	CreditTwo (4)	Credit	Online Payment - Thank You			002
2017-04-25	\$-20	BankOne (1)	Debit	Late fee			BankOne.0.1
2017-04-25	\$28.53	CreditTwo (4)	Purchase	Interest Charged			001
2017-04-25	\$11	BankOne (1)	Debit	Generated Trans 1			BankOne.1.1
2017-04-24	\$-22	BankOne (1)	Debit	Generated Trans 2			BankOne.1.2
2017-04-24	\$33	BankOne (1)	Debit	Generated Trans 3			BankOne.1.3
2017-04-24	\$-44	BankOne (1)	Debit	Generated Trans 4			BankOne.1.4
2017-04-24	\$55	BankOne (1)	Debit	Generated Trans 5			BankOne.1.5
2017-04-24	\$-66	BankOne (1)	Debit	Generated Trans 6			BankOne.1.6
Date	Amount	Account	Type	Name	Memo	Description	FITID

Showing 1 to 10 of 29 entries

Shows transactions imported from OFX statements.

Transactions View

Transactions - BiweeklyBudget

2 Accounts with stale data. [View Accounts.](#)

20 Unreconciled OFXTransactions.

[Add Transaction](#)

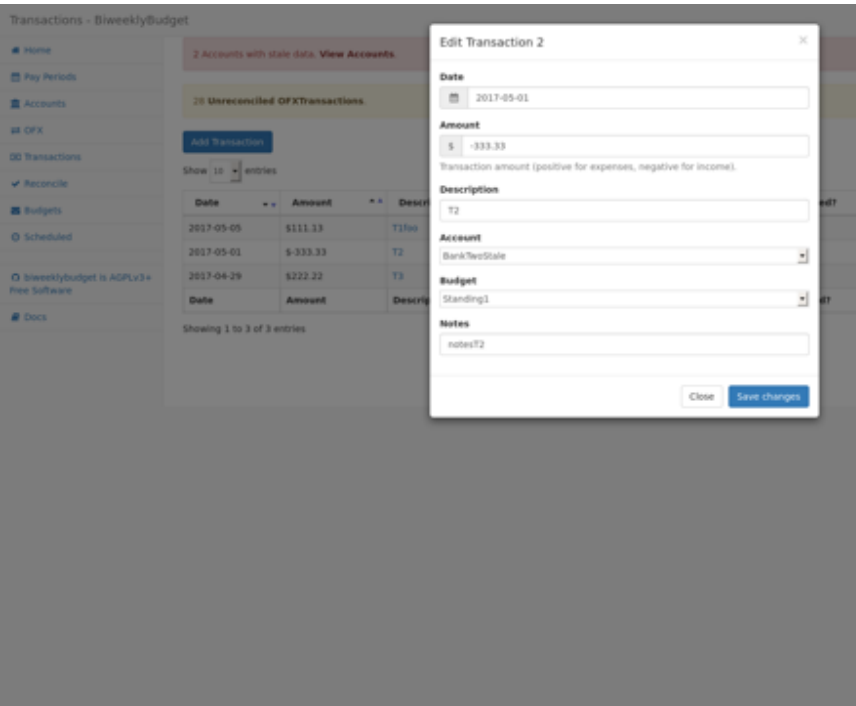
Show 10 entries Account:

Date	Amount	Description	Account	Budget	Scheduled?	Budgeted Amount
2017-05-05	\$111.33	T1foo	BankOne (1)	Periodic1 (1)	Yes (1)	\$111.31
2017-05-01	\$-333.33	T2	BankTwoStale (2)	Standing1 (4)	Yes (3)	
2017-04-29	\$222.22	T3	CreditOne (3)	Periodic2 (2)		
Date	Amount	Description	Account	Budget	Scheduled?	Budgeted Amount

Showing 1 to 3 of 3 entries

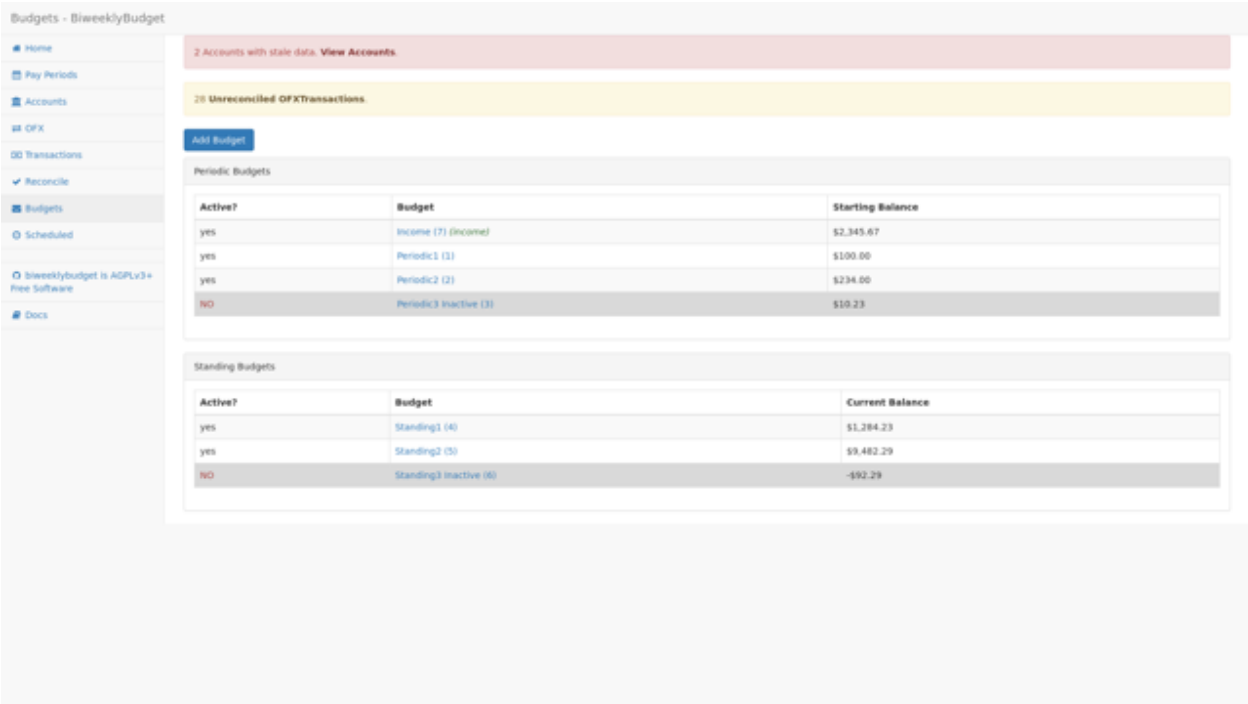
Shows all manually-entered transactions.

Transaction Detail



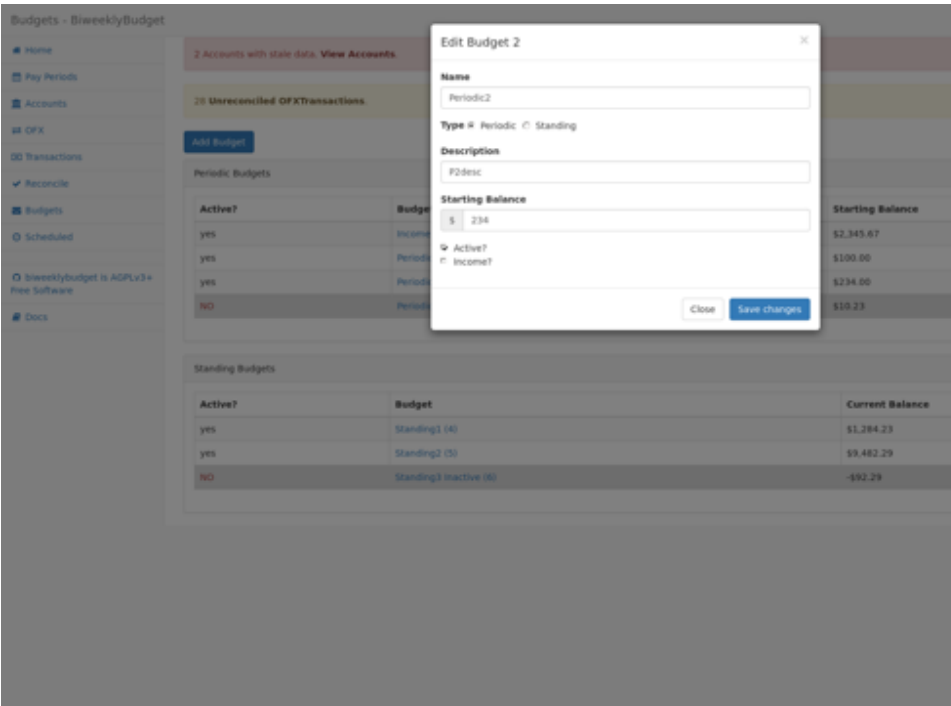
Transaction detail modal to view and edit a transaction.

Budgets



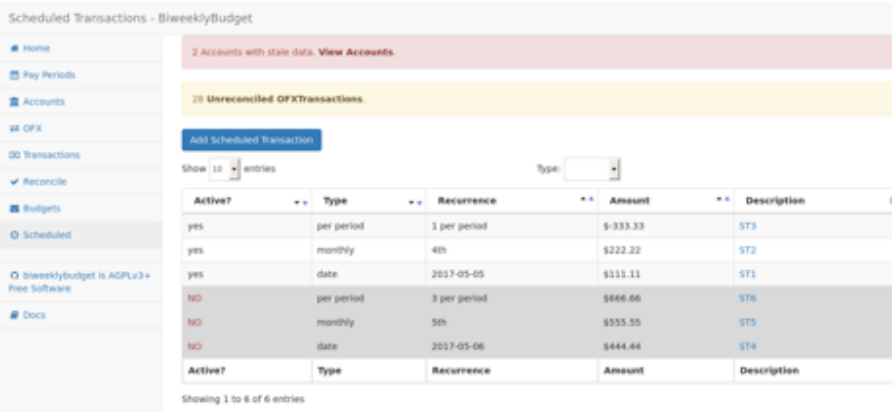
List all budgets

Single Budget View



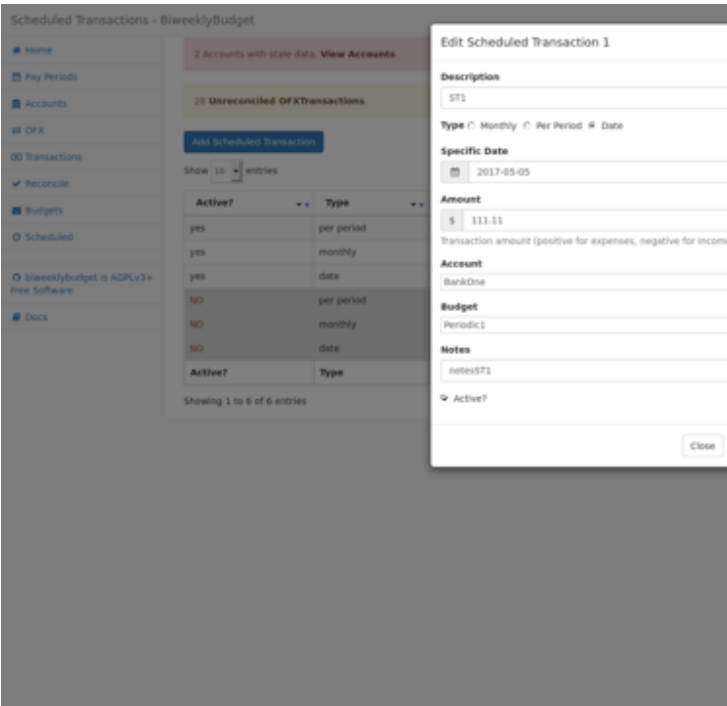
Budget detail modal to view and edit a budget.

Scheduled Transactions



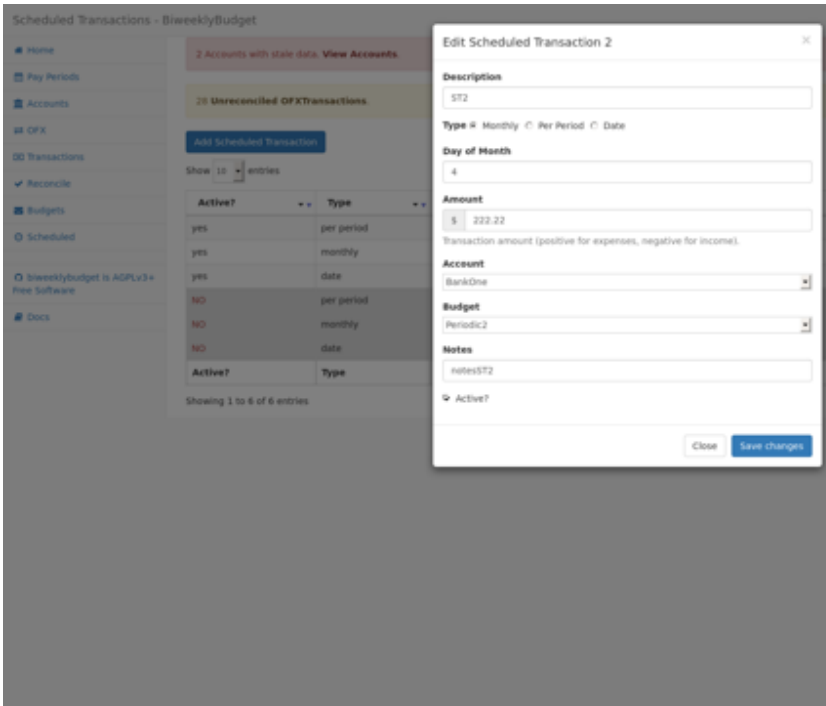
List all scheduled transactions (active and inactive).

Specific Date Scheduled Transaction



Scheduled transactions can occur one-time on a single specific date.

Monthly Scheduled Transaction



Scheduled transactions can occur monthly on a given date.

Number Per-Period Scheduled Transactions

Scheduled transactions can occur a given number of times per pay period.

Edit Scheduled Transaction 3

Description: ST3

Type: ☐ Monthly ☒ Per Period ☐ Date

Number Per Pay Period: 1

Amount: \$ -333.33
Transaction amount (positive for expenses, negative for income).

Account: BankTwoState

Budget: Standing1

Notes: NotesST3

☒ Active?

Close Save changes

Scheduled Transactions - BiweeklyBudget

2 Accounts with state data. [View Accounts.](#)

28 Unreconciled OFX Transactions.

Add Scheduled Transaction

Show 10 entries

Active?	Type
yes	per period
yes	monthly
yes	date
NO	per period
NO	monthly
NO	date

Showing 1 to 6 of 6 entries

Account	Budget
BankTwoState (2)	Standing1 (4)
BankOne (1)	Periodic2 (2)
BankOne (1)	Periodic1 (1)
BankTwoState (2)	Standing1 (4)
BankOne (1)	Periodic2 (2)
BankOne (1)	Periodic1 (1)

Previous Next

Reconcile Transactions with OFX

OFX Transactions reported by financial institutions can be marked as reconciled with a corresponding Transaction.

Reconcile Transactions - BiweeklyBudget

Transactions

Date	Amount	Account	Budget
2017-04-29	\$222.22	Acct: CreditOne (3)	Budget: Periodic2 (2) (no OFX)
2017-05-01	-333.33	Acct: BankTwoState (2)	Budget: Standing1 (4) (no OFX)

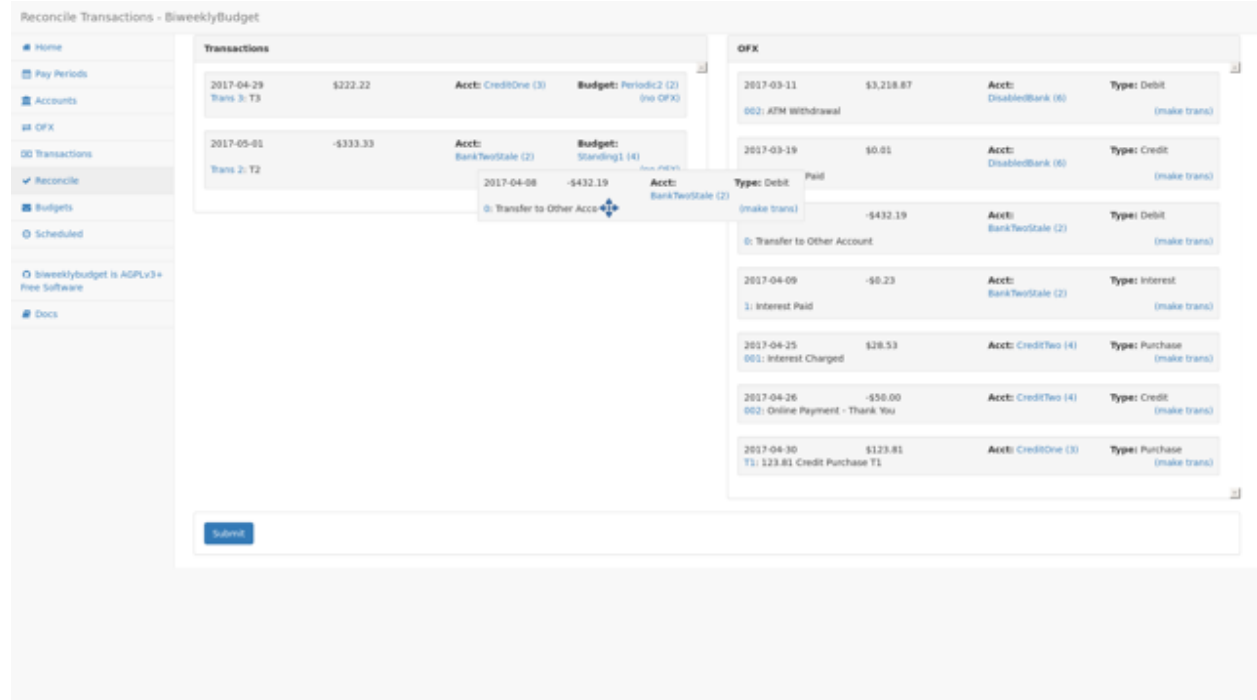
OFX

2017-03-11	\$3,218.87	Acct: DisabledBank (6)	Type: Debit
002: ATM Withdrawal			(make trans)
2017-03-19	\$0.01	Acct: DisabledBank (6)	Type: Credit
001: Interest Paid			(make trans)
2017-04-08	-432.19	Acct: BankTwoState (2)	Type: Debit
0: Transfer to Other Account			(make trans)
2017-04-09	-60.23	Acct: BankTwoState (2)	Type: Interest
1: Interest Paid			(make trans)
2017-04-25	\$28.53	Acct: CreditTwo (4)	Type: Purchase
001: Interest Charged			(make trans)
2017-04-26	-150.00	Acct: CreditTwo (4)	Type: Credit
002: Online Payment - Thank You			(make trans)
2017-04-30	\$123.81	Acct: CreditOne (3)	Type: Purchase
T1: 123.81 Credit Purchase T1			(make trans)

Submit

Drag-and-Drop Reconciling

To reconcile an OFX transaction with a Transaction, just drag and drop.



Getting Started

Requirements

Note: Alternatively, biweeklybudget is also distributed as a *Docker container*. Using the dockerized version will eliminate all of these dependencies aside from MySQL and Vault (the latter only if you choose to take advantage of the OFX downloading), both of which you can also run in containers.

- Python 2.7 or 3.3+ (currently tested with 2.7, 3.3, 3.4, 3.5, 3.6 and developed with 3.6)
- Python [VirtualEnv](#) and `pip` (recommended installation method; your OS/distribution should have packages for these)
- Git, to install certain upstream dependencies.
- MySQL, or a compatible database (e.g. [MariaDB](#)). biweeklybudget uses [SQLAlchemy](#) for database abstraction, but currently specifies some MySQL-specific options, and is only tested with MySQL.
- To use the automated OFX transaction downloading functionality:
 - A running, reachable instance of [Hashicorp Vault](#) with your financial institution web credentials stored in it.
 - [PhantomJS](#) for downloading transaction data from institutions that do not support OFX remote access (“Direct Connect”).

Installation

It's recommended that you install into a virtual environment (virtualenv / venv). See the [virtualenv usage documentation](#) for information on how to create a venv.

This app is developed against Python 3.6, but should work back to 2.7. It does not support Python3 < 3.3.

Please note that, at the moment, one dependency is installed via git in order to make use of an un-merged pull request that fixes a bug; since installation doesn't support specifying git dependencies in `setup.py`, you must install with `requirements.txt` directly:

```
git clone https://github.com/jantman/biweeklybudget.git && cd biweeklybudget
virtualenv --python=python3.6 .
source bin/activate
pip install -r requirements.txt
python setup.py develop
```

Configuration

biweeklybudget can take its configuration settings via either constants defined in a Python module or environment variables. Configuration in environment variables always overrides configuration from the settings module.

Settings Module

`biweeklybudget.settings` imports all globals/constants from a module defined in the `SETTINGS_MODULE` environment variable. The recommended way to configure this is to create your own separate Python package for customization (either in a private git repository, or just in a directory on your computer) and install this package into the same virtualenv as biweeklybudget. You then set the `SETTINGS_MODULE` environment variable to the Python module/import path of this module (i.e. the dotted path, like `packagename.modulename`).

Once you've created the customization package, you can install it in the virtualenv with `pip install -e <git URL>` (if it is kept in a git repository) or `pip install -e <local path>`.

This customization package can also be used for [Loading Data](#) during development, or implementing [Custom OFX Downloading via Selenium](#). It is the recommended configuration method if you need to include more logic than simply defining static configuration settings.

Environment Variables

Every configuration setting can also be specified by setting an environment variable with the same name; these will override any settings defined in a `SETTINGS_MODULE`, if specified. Note that some environment variables require specific formatting of their values; see the [settings module documentation](#) for a list of these variables and the required formats.

Usage

Setup

```
source bin/activate
export SETTINGS_MODULE=<settings module>
```

It's recommended that you create an alias to do this for you. Alternatively, instead of setting `SETTINGS_MODULE`, you can export the required environment variables (see above).

Flask

For information on the Flask application, see *Flask App* <flask_app>.

Command Line Entrypoints and Scripts

biweeklybudget provides the following setuptools entrypoints (command-line script wrappers in `bin/`). First setup your environment according to the instructions above.

- `bin/db_tester.py` - Skeleton of a script that connects to and inits the DB. Edit this to use for one-off DB work.
- `loaddata` - Entrypoint for dropping **all** existing data and loading test fixture data, or your base data. This is an awful, manual hack right now.
- `ofxbackfiller` - Entrypoint to backfill OFX Statements to DB from disk.
- `ofxgetter` - Entrypoint to download OFX Statements for one or all accounts, save to disk, and load to DB. See *OFX*.

Docker

Biweeklybudget is also distributed as a [docker image](#), to make it easier to run without installing as many [Requirements](#).

You can pull the latest version of the image with `docker pull jantman/biweeklybudget:latest`, or a specific release version X.Y.Z with `docker pull jantman/biweeklybudget:X.Y.Z`.

The only dependencies for a Docker installation are:

- MySQL, which can be run via Docker ([MariaDB](#) recommended) or local on the host
- Vault, if you wish to use the OFX downloading feature, which can also be run [via Docker](#)

Important Note: If you run MySQL and/or Vault in containers, please make sure that their data is backed up and will not be removed.

The [image](#) runs with the [tini](#) init wrapper and uses [gunicorn](#) under Python 3.6 to serve the web UI, exposed on port 80. Note that, while it runs with 4 worker threads, there is no HTTP proxy in front of Gunicorn and this image is intended for local network use by a single user/client.

For ease of running, the image defaults the `SETTINGS_MODULE` environment variable to `biweeklybudget.settings_example`. This allows leveraging the environment variable [configuration](#) overrides so that you need only specify configuration options that you want to override from [settings_example.py](#).

For ease of running, it's highly recommended that you put your configuration in a Docker-readable environment variables file.

Environment Variable File

In the following examples, we reference the following environment variable file. It will override settings from [settings_example.py](#) as needed; specifically, we need to override the database connection string, pay period start date and reconcile begin date. In the examples below, we would save this as `biweeklybudget.env`:

```
DB_CONNSTRING=mysql+pymysql://USERNAME:PASSWORD@HOST:PORT/DBNAME?charset=utf8mb4
PAY_PERIOD_START_DATE=2017-03-28
RECONCILE_BEGIN_DATE=2017-02-15
```

Containerized MySQL Example

This assumes that you already have a MySQL database container running with the container name “mysql” and exposing port 3306, and that we want the biweeklybudget web UI served on host port 8080:

In our `biweeklybudget.env`, we would specify the database connection string for the “mysql” container:

```
DB_CONNSTRING=mysql+pymysql://USERNAME:PASSWORD@mysql:3306/DBNAME?charset=utf8mb4
```

And then run biweeklybudget:

```
docker run --name biweeklybudget --env-file biweeklybudget.env \
-p 8080:80 --link mysql jantman/biweeklybudget:latest
```

Host-Local MySQL Example

It is also possible to use a MySQL server on the physical (Docker) host system. To do so, you’ll need to know the host system’s IP address. On Linux when using the default “bridge” Docker networking mode, this will coorespond to a `docker0` interface on the host system. The Docker documentation on [adding entries to the Container’s hosts file](#) provides a helpful snippet for this (on my systems, this results in `172.17.0.1`):

```
ip -4 addr show scope global dev docker0 | grep inet | awk '{print $2}' | cut -d / -f 1
↪1
```

In our `biweeklybudget.env`, we would specify the database connection string that uses the “dockerhost” hosts file entry, created by the `--add-host` option:

```
# "dockerhost" is added to /etc/hosts via the `--add-host` docker run option
DB_CONNSTRING=mysql+pymysql://USERNAME:PASSWORD@dockerhost:3306/DBNAME?charset=utf8mb4
```

So using that, we could run biweeklybudget listening on port 8080 and using our host’s MySQL server (on port 3306):

```
docker run --name biweeklybudget --env-file biweeklybudget.env \
--add-host="dockerhost:$(ip -4 addr show scope global dev docker0 | grep inet | awk '
↪{print $2}' | cut -d / -f 1)" \
-p 8080:80 jantman/biweeklybudget:latest
```

You may need to adjust those commands depending on your operating system, Docker networking mode, and MySQL server.

Settings Module Example

If you need to provide biweeklybudget with more complicated configuration, this is still possible via a Python settings module. The easiest way to inject one into the Docker image is to [mount](#) a python module directly into the biweeklybudget package directory. Assuming you have a custom settings module on your local machine at `/opt/biweeklybudget-settings.py`, you would run the container as shown below to mount the custom settings module into the container and use it. Note that this example assumes using MySQL in another container; adjust as necessary if you are using MySQL running on the Docker host:

```
docker run --name biweeklybudget -e SETTINGS_MODULE=biweeklybudget.mysettings \
-v /opt/biweeklybudget-settings.py:/app/lib/python3.6/site-packages/biweeklybudget/
↪mysettings.py \
-p 8080:80 --link mysql jantman/biweeklybudget:latest
```

Note on Locales

biweeklybudget uses Python's `locale` module to format currency. This requires an appropriate locale installed on the system. The docker image distributed for this package only includes the `en_US.UTF-8` locale. If you need a different one, please cut a pull request against `docker_build.py`.

Running ofxgetter in Docker

If you wish to use the *ofxgetter* script inside the Docker container, some special settings are needed:

1. You must mount the statement save path (`STATEMENTS_SAVE_PATH`) into the container.
2. You must mount the Vault token file path (`TOKEN_PATH`) into the container.
3. You must set either the `VAULT_ADDR` environment variable, or the `VAULT_ADDR` setting.

As an example, for using ofxgetter with `STATEMENTS_SAVE_PATH` in your settings file set to `/statements` and `TOKEN_PATH` set to `/.token` (root paths used here for simplicity in the example), you would add to your docker run command:

```
-v /statements:/statements \
-v /.token:/.token
```

Assuming your container was running with `--name biweeklybudget`, you could run ofxgetter (e.g. via cron) as:

We run explicitly in the `statements` directory so that if ofxgetter encounters an error when using a *ScreenScraper* class, the screenshots and HTML output will be saved to the host filesystem.

Flask Application

Running

1. First, setup your environment per *Getting Started - Setup*.
2. `export FLASK_APP="biweeklybudget.flaskapp.app"`
3. `flask --help` for information on usage:
 - Run App: `flask run`
 - Run with debug/reload: `flask rundev`

To run the app against the acceptance test database, use: `DB_CONNSTRING='mysql+pymysql://budgetTester@127.0.0.1:3306/budgettest?charset=utf8mb4' flask run`

By default, Flask will only bind to localhost. If you want to bind to all interfaces, you can add `--host=0.0.0.0` to the `flask run` commands. Please be aware of the implications of this (see “Security”, below).

If you wish to run the flask app in a multi-process/thread/worker WSGI container, be sure that you run the `initdb` entrypoint before starting the workers. Otherwise, it's likely that all workers will attempt to create the database tables or run migrations at the same time, and fail.

Security

This code hasn't been audited. It might have SQL injection vulnerabilities in it. It might dump your bank account details in HTML comments. Anything is possible!

To put it succinctly, this was written to be used by me, and me only. It was written with the assumption that anyone who can possibly access any of the application at all, whether in a browser or locally, is authorized to view and/or edit anything and everything related to the application (configuration, everything in the database, everything in Vault if it's being used). If you even think about making this accessible to anything other than localhost on a computer you physically own, it's entirely up to you how you secure it, but make sure you do it really well.

OFX Transaction Downloading

biweeklybudget has the ability to download OFX transaction data from your financial institutions, either manually or automatically (via an external command scheduler such as `cron`).

There are two overall methods of downloading transaction data; for banks that support the [OFX protocol](#), statement data can be downloaded using HTTP only, via the [ofxclient](#) project (note our requirements file specifies the upstream of [PR #37](#), which includes a fix for Discover credit cards). For banks that do not support the OFX protocol and require you to use their website to download OFX format statements, biweeklybudget provides a base [ScreenScraper](#) class that can be used to develop a [selenium](#)-based tool to automate logging in to your bank's site and downloading the OFX file.

In order to use either of these methods, you must have an instance of [Hashicorp Vault](#) running and have your login credentials stored in it.

Important Note on Transaction Downloading

biweeklybudget includes support for automatically downloading transaction data from your bank. Credentials are stored in an instance of [Hashicorp Vault](#), as that is a project the author has familiarity with, and was chosen as the most secure way of storing and retrieving secrets non-interactively. Please keep in mind that it is your decision and your decision alone how secure your banking credentials are kept. What is considered acceptable to the author of this program may not be acceptably secure for others; it is your sole responsibility to understand the security and privacy implications of this program as well as Vault, and to understand the risks of storing your banking credentials in this way.

Also note that biweeklybudget includes a base class ([ScreenScraper](#)) intended to simplify developing [selenium](#)-based browser automation to log in to financial institution websites and download your transactions. Many banks and other financial institutions have terms of service that *explicitly forbid automated or programmatic use of their websites*. As such, it is up to you as the user of this software to determine your bank's policy and abide by it. I provide a base class to help in writing automated download tooling if your institution allows it, but I cannot and will not distribute institution-specific download tooling.

ofxgetter endpoint

This package provides an `ofxgetter` command line endpoint that can be used to download OFX statements for one or all Accounts that are appropriately configured. The script used for this provides exit codes and logging suitable for use via `cron` (it exits non-zero if any accounts failed, and unless options are provided to increase verbosity, only outputs the number of accounts successfully downloaded as well as any errors).

Vault Setup

Configuring and running Vault is outside the scope of this document. Once you have a Vault installation running and appropriately secured (you shouldn't be using the dev server unless you want to lose all your data every time you reboot) and have given biweeklybudget access to a valid token stored in a file somewhere, you'll need to ensure that your username and password data is stored in Vault in the proper format (`username` and `password`

keys). If you happen to use [LastPass](#) to store your passwords, you may find my [lastpass2vault.py](#) helpful; run it as `./lastpass2vault.py -vv -f PATH_TO_VAULT_TOKEN LASTPASS_USERNAME` and it will copy all of your credentials from LastPass to Vault, preserving the folder structure.

Configuring Accounts for Downloading with ofxclient

1. Use the `ofxclient` CLI to configure and test your account.
2. Put your creds in Vault.
3. Migrate `~/ofxclient.ini` to JSON, add it to your *Account*.

A working configuration for a Bank account might look something like this:

```
{
  "routing_number": "012345678",
  "account_type": "CHECKING",
  "description": "Checking",
  "number": "111222333",
  "local_id": "f0a14074d33cdf83b4a099bc322dbe2fe19680ca1719425b33de5022",
  "institution": {
    "client_args": {
      "app_version": "2200",
      "app_id": "QWIN",
      "ofx_version": "103",
      "id": "f87217350cc341e2ba7407cf99dcdede"
    },
    "description": "MyBank",
    "url": "https://ofx.MyBank.com",
    "local_id": "e51fb78f88580a1c2e3bb65bd59495384388abda8796c9bf06dcf",
    "broker_id": "",
    "org": "ORG",
    "id": "98765"
  }
}
```

Configuring Accounts for Downloading with Selenium

In your *customization package* `<getting_started.customization>`, subclass *ScreenScraper*. Override the constructor to take whatever keyword arguments are required, and add those to your account's `ofxgetter_config_json` as shown below. `:py:class:~biweeklybudget.ofxgetter.OfxGetter` will instantiate the class passing it the specified keyword arguments in addition to `username`, `password` and `savedir` keyword arguments. `savedir` is the directory under `STATEMENTS_SAVE_PATH` where the account's OFX statements should be saved. After instantiating the class, `ofxgetter` will call the class's `run()` method with no arguments, and expect to receive an OFX statement string back.

If cookies are a concern, be aware that saving and loading cookies is [broken in PhantomJS 2.x](#). If you need to persist cookies across sessions, look into the *ScreenScraper* class' `load_cookies()` and `save_cookies()` methods.

```
{
  "class_name": "MyScraper",
  "module_name": "budget_customization.myscraper",
  "institution": {},
  "kwargs": {
    "acct_num": "1234"
  }
}
```

```
}  
}
```

Here's a simple, contrived example of such a class:

```
import logging  
import time  
import codecs  
from datetime import datetime  
  
from selenium.common.exceptions import NoSuchElementException  
  
from biweeklybudget.screenscraper import ScreenScraper  
  
logger = logging.getLogger(__name__)  
  
# suppress selenium logging  
selenium_log = logging.getLogger("selenium")  
selenium_log.setLevel(logging.WARNING)  
selenium_log.propagate = True  
  
class MyScraper(ScreenScraper):  
  
    def __init__(self, username, password, savedir='./',  
                 acct_num=None, screenshot=False):  
        """  
        :param username: username  
        :type username: str  
        :param password: password  
        :type password: str  
        :param savedir: directory to save OFX in  
        :type savedir: str  
        :param acct_num: last 4 of account number, as shown on homepage  
        :type acct_num: str  
        """  
        super(MyScraper, self).__init__(  
            savedir=savedir, screenshot=screenshot  
        )  
        self.browser = self.get_browser('phantomjs')  
        self.username = username  
        self.password = password  
        self.acct_num = acct_num  
  
    def run(self):  
        """ download the transactions, return file path on disk """  
        logger.debug("running, username={u}".format(u=self.username))  
        logger.info('Logging in...')  
        try:  
            self.do_login(self.username, self.password)  
            logger.info('Logged in; sleeping 2s to stabilize')  
            time.sleep(2)  
            self.do_screenshot()  
            self.select_account()  
            act = self.get_account_activity()  
        except Exception:  
            self.error_screenshot()  
            raise
```



```

    return act

def do_login(self, username, password):
    self.get_page('http://example.com')
    raise NotImplementedError("login to your bank here")

def select_account(self):
    self.get_page('http://example.com')
    logger.debug('Finding account link...')
    link = self.browser.find_element_by_xpath(
        '//a[contains(text(), "%s")]' % self.acct_num
    )
    logger.debug('Clicking account link: %s', link)
    link.click()
    self.wait_for_ajax_load()
    self.do_screenshot()

def get_account_activity(self):
    # some bank-specific stuff here, then we POST to get OFX
    post_list = self.xhr_post_urlencoded(
        post_url, post_data, headers=post_headers
    )
    if not post_list.startswith('OFXHEADER'):
        self.error_screenshot()
        with codecs.open('result', 'w', 'utf-8') as fh:
            fh.write(post_list)
        raise SystemExit("Got non-OFX response")
    return post_list

```

Getting Help

Bugs and Feature Requests

Bug reports and feature requests are happily accepted via the [GitHub Issue Tracker](#). Pull requests are welcome. Issues that don't have an accompanying pull request will be worked on as my time and priority allows.

Development

To install for development:

1. Fork the [biweeklybudget](#) repository on GitHub
2. Create a new branch off of master in your fork.

```

$ virtualenv biweeklybudget
$ cd biweeklybudget && source bin/activate
$ pip install -e git+git@github.com:YOURNAME/biweeklybudget.git@BRANCHNAME
→ #egg=biweeklybudget
$ cd src/biweeklybudget

```

The git clone you're now in will probably be checked out to a specific commit, so you may want to `git checkout BRANCHNAME`.

Guidelines

- pep8 compliant with some exceptions (see `pytest.ini`)
- 100% test coverage with `pytest` (with valid tests)

Loading Data

The sample data used for acceptance tests is defined in `biweeklybudget/tests/fixtures/sampledata.py`. This data can be loaded by *setting up the environment* <getting_started.setup> and then using the `loaddata` entrypoint (the following values for options are actually the defaults, but are shown for clarity):

```
loaddata -m biweeklybudget.tests.fixtures.sampledata -c SampleDataLoader
```

This entrypoint will **drop all tables and data** and then load fresh data from the specified class.

If you wish, you can copy `biweeklybudget/tests/fixtures/sampledata.py` to your *customization package* <getting_started.customization> and edit it to load your own custom data. This should only be required if you plan on dropping and reinitializing the database often.

Testing

Testing is done via `pytest`, driven by `tox`.

- testing is as simple as:
 - `pip install tox`
 - `tox`
- If you want to pass additional arguments to `pytest`, add them to the `tox` command line after “–”. i.e., for verbose `pytest` output on `py27` tests: `tox -e py27 -- -v`

For rapid iteration on tests, you can either use my `toxlt` script to re-run the test commands in an existing `tox` environment, or you can use the `bin/t` and `bin/ta` scripts to run unit or acceptance tests, respectively, on only one module.

Unit Tests

There are minimal unit tests, really only some examples and room to test some potentially fragile code. Run them via the `^py\d+` `tox` environments.

Integration Tests

There’s a `pytest` marker for integration tests, effectively defined as anything that might use either a mocked/in-memory DB or the flask test client, but no HTTP server and no real RDBMS. Run them via the `integration` `tox` environment. But there aren’t any of them yet.

Acceptance Tests

There are acceptance tests, which use a real MySQL DB (see the connection string in `tox.ini` and `conftest.py`) and a real Flask HTTP server, and selenium. Run them via the `acceptance` `tox` environment.

The acceptance tests connect to a local MySQL database using a connection string specified by the `DB_CONNSTRING` environment variable, or defaulting to a DB name and user/password that can be seen in `conf/test.py`. Once connected, the tests will drop all tables in the test DB, re-create all models/tables, and then load sample data. After the DB is initialized, tests will run the local Flask app on a random port, and run Selenium backed by PhantomJS.

If you want to run the acceptance tests without dumping and refreshing the test database, export the `NO_REFRESH_DB` environment variable. Setting the `NO_CLASS_REFRESH_DB` environment variable will prevent refreshing the DB after classes that manipulate data; this will cause subsequent tests to fail but can be useful for debugging.

Alembic DB Migrations

This project uses [Alembic](#) for DB migrations:

- To generate migrations, run `alembic -c biweeklybudget/alembic/alembic.ini revision --autogenerate -m "message"` and examine/edit then commit the resulting file(s).
- To apply migrations, run `alembic -c biweeklybudget/alembic/alembic.ini upgrade head`.
- To see the current DB version, run `alembic -c biweeklybudget/alembic/alembic.ini current`.
- To see migration history, run `alembic -c biweeklybudget/alembic/alembic.ini history`.

Database Debugging

If you set the `SQL_ECHO` environment variable to “true”, all SQL run by SQLAlchemy will be logged at INFO level.

Docker Image Build

Use the `docker` tox environment. See the docstring at the top of `biweeklybudget/tests/docker_build.py` for further information.

Frontend / UI

The UI is based on [BlackrockDigital's startbootstrap-sb-admin-2](#), currently as of the 3.3.7-1 GitHub release. It is currently not modified at all, but should it need to be rebuilt, this can be done with: `pushd biweeklybudget/flaskapp/static/startbootstrap-sb-admin-2 && gulp`

Sphinx also generates documentation for the custom javascript files. This must be done manually on a machine with `jsdoc` installed, via: `tox -e jsdoc`.

Release Checklist

1. Open an issue for the release; cut a branch off master for that issue.
2. Verify whether or not DB migrations are needed. If they are, ensure they've been created, tested and verified.
3. Confirm that there are `CHANGES.rst` entries for all major changes.
4. Rebuild documentation and javascript documentation locally: `tox -e jsdoc, docs`. Commit any changes.
5. Run the Docker image build and tests locally: `tox -e docker`.
6. Ensure that Travis tests passing in all environments.

7. Ensure that test coverage is no less than the last release, and that there are acceptance tests for any non-trivial changes.
8. If there have been any major visual or functional changes to the UI, regenerate screenshots via `tox -e screenshots`.
9. Increment the version number in `biweeklybudget/version.py` and add version and release date to `CHANGES.rst`, then push to GitHub.
10. Confirm that `README.rst` renders correctly on GitHub.
11. Upload package to testpypi:
 - Make sure your `~/.pypirc` file is correct (a repo called `test` for <https://testpypi.python.org/pypi>)
 - `rm -Rf dist`
 - `python setup.py register -r https://testpypi.python.org/pypi`
 - `python setup.py sdist bdist_wheel`
 - `twine upload -r test dist/*`
 - Check that the README renders at <https://testpypi.python.org/pypi/biweeklybudget>
12. Create a pull request for the release to be merged into master. Upon successful Travis build, merge it.
13. Tag the release in Git, push tag to GitHub:
 - tag the release. for now the message is quite simple: `git tag -a X.Y.Z -m 'X.Y.Z released YYYY-MM-DD'`
 - push the tag to GitHub: `git push origin X.Y.Z`
14. Upload package to live pypi:
 - `twine upload dist/*`
15. Build and push the new Docker image:
 - Check out the git tag: `git checkout X.Y.Z`
 - Build the Docker image: `tox -e docker`
 - Follow the instructions from that script to push the image to the Docker Hub and tag a “latest” version.
16. make sure any GH issues fixed in the release were closed.
17. Log in to readthedocs.org and enable building of the release tag. You may need to re-run another build to get the tag to be picked up.

Changelog

0.1.2 (2017-05-28)

- Minor fix to instructions printed after release build in `biweeklybudget/tests/docker_build.py`
- [Issue #61](#) - Document running `ofxgetter` in the Docker container.
- fix `ReconcileRule repr` for uncommitted (id is None)
- [Issue #67](#) - `ofxgetter` logging - suppress DB and Alembic logging at INFO and above; log number of inserted and updated transactions.

- [Issue #71](#) - Fix display text next to prev/curr/next periods on `/payperiod/YYYY-mm-dd` view; add 6 more future pay periods to the `/payperiods` table.
- [Issue #72](#) - Add a built-in method for transferring money from periodic (per-pay-period) to standing budgets; add budget Transfer buttons on Budgets and Pay Period views.
- [Issue #75](#) - Add link on payperiod views to skip a ScheduledTransaction instance this period.
- [Issue #57](#) - Ignore future transactions from unreconciled transactions list.
- Transaction model - fix default for `date` field to actually be just a date; previously, Transactions with `date` left as default would attempt to put a full datetime into a date column, and throw a data truncation warning.
- Transaction model - Fix `__repr__` to not throw exception on un-persisted objects.
- When adding or updating the `actual_amount` of a Transaction against a Standing Budget, update the `current_balance` of the budget.
- Fix ordering of Transactions table on Pay Period view, to properly sort by date and then amount.
- Numerous fixes to date-sensitive acceptance tests.
- [Issue #79](#) - Update `/pay_period_for` view to redirect to current pay period when called with no query parameters; add bookmarkable link to current pay period to Pay Periods view.

0.1.1 (2017-05-20)

- Improve ofxgetter/ofxupdater error handling; catch OFX files with error messages in them.
- [Issue #62](#) - Fix phantomjs in Docker image. * Allow docker image tests to run against an existing image, defined by `DOCKER_TEST_TAG`. * Retry MySQL DB creation during Docker tests until it succeeds, or fails 10 times. * Add testing of PhantomJS in Docker image testing; check version and that it actually works (GET a page). * More reliable stopping and removing of Docker containers during Docker image tests.
- [Issue #63](#) - Enable gunicorn request logging in Docker container.
- Switch to my fork of ofxclient in requirements.txt, to pull in [ofxclient PR #41](#)
- [Issue #64](#) - Fix duplicate/multiple on click event handlers in UI that were causing duplicate transactions.

0.1.0 (2017-05-07)

- Initial Release

biweeklybudget

biweeklybudget package

Subpackages

biweeklybudget.flaskapp package

Subpackages

biweeklybudget.flaskapp.views package

Submodules

`biweeklybudget.flaskapp.views.accounts` module

`biweeklybudget.flaskapp.views.budgets` module

`biweeklybudget.flaskapp.views.example` module

`biweeklybudget.flaskapp.views.formhandlerview` module

`biweeklybudget.flaskapp.views.help` module

`biweeklybudget.flaskapp.views.index` module

`biweeklybudget.flaskapp.views.ofx` module

`biweeklybudget.flaskapp.views.payperiods` module

`biweeklybudget.flaskapp.views.reconcile` module

`biweeklybudget.flaskapp.views.scheduled` module

`biweeklybudget.flaskapp.views.searchableajaxview` module

`biweeklybudget.flaskapp.views.transactions` module

Submodules

`biweeklybudget.flaskapp.app` module

`biweeklybudget.flaskapp.cli_commands` module

`biweeklybudget.flaskapp.cli_commands.template_paths()`
Return a list of all Flask app template paths, to auto-reload on change.

from <http://stackoverflow.com/a/41666467/211734>

Returns list of all template paths

Return type list

`biweeklybudget.flaskapp.context_processors` module

`biweeklybudget.flaskapp.filters` module

`biweeklybudget.flaskapp.jinja_tests` module

biweeklybudget.flaskapp.jsonencoder module

```
class biweeklybudget.flaskapp.jsonencoder.MagicJSONEncoder (skipkeys=False,      en-
                                                             sure_ascii=True,
                                                             check_circular=True,
                                                             allow_nan=True,
                                                             sort_keys=False,
                                                             indent=None,      sep-
                                                             arators=None,
                                                             encoding='utf-8',  de-
                                                             fault=None)
```

Bases: `json.encoder.JSONEncoder`

Customized JSONEncoder class that uses `as_dict` properties on objects to encode them.

default (*o*)

biweeklybudget.flaskapp.notifications module

```
class biweeklybudget.flaskapp.notifications.NotificationsController
```

Bases: `object`

static budget_account_sum ()

Return the sum of current balances for all `is_budget_source` accounts.

Returns Combined balance of all budget source accounts

Return type `float`

static get_notifications ()

Return all notifications that should be displayed at the top of pages, as a list in the order they should appear. Each list item is a dict with keys “classes” and “content”, where classes is the string that should appear in the notification div’s “class” attribute, and content is the string content of the div.

static num_stale_accounts ()

Return the number of accounts with stale data.

@TODO This is a hack because I just cannot figure out how to do this natively in SQLAlchemy.

Returns count of accounts with stale data

Return type `int`

static num_unreconciled_ofx ()

Return the number of unreconciled OFXTransactions.

Returns number of unreconciled OFXTransactions

Return type `int`

static standing_budgets_sum ()

Return the sum of current balances of all standing budgets.

Returns sum of current balances of all standing budgets

Return type `float`

biweeklybudget.models package

Submodules

biweeklybudget.models.account module

```
class biweeklybudget.models.account.Account (**kwargs)
    Bases: sqlalchemy.ext.declarative.api.Base, biweeklybudget.models.base.
    ModelAsDict

    _sa_class_manager = <ClassManager of <class 'biweeklybudget.models.account.Account'> at 7f86886fd2a0>

    acct_type
        Type of account (Enum AcctType)

    all_statements
        Relationship to all OFXStatement for this Account

    balance
        Return the latest AccountBalance object for this Account.

        Returns latest AccountBalance for this Account

        Return type biweeklybudget.models.account_balance.AccountBalance

    credit_limit
        credit limit, for credit accounts

    description
        description

    for_ofxgetter
        Return whether or not this account should be handled by ofxgetter.

        Returns whether or not ofxgetter should run for this account

        Return type bool

    id
        Primary Key

    is_active
        whether or not the account is active and can be used, or historical

    is_budget_source
        Return whether or not this account should be considered a funding source for Budgets.

        Returns whether or not this account is a Budget funding source

        Return type bool

    is_stale
        Return whether or not there is stale data for this account.

        Returns whether or not data for this account is stale

        Return type bool

    name
        name for the account

    negate_ofx_amounts
        For use in reconciling our Transaction entries with the account's OFXTransaction entries, whether
```


or not to negate the OfxTransaction amount. We enter Transactions with income as negative amounts and expenses as positive amounts, but most bank OFX statements will show the opposite.

ofx_cat_memo_to_name

whether or not to concatenate the OFX memo text onto the OFX name text; for banks like Chase that use the memo for run-on from the name

ofx_statement

Return the latest OFXStatement for this Account.

Returns latest OFXStatement for this Account

Return type *biweeklybudget.models.ofx_statement.OFXStatement*

ofxgetter_config

Return the deserialized ofxgetter_config_json dict.

Returns ofxgetter config

Return type *dict*

ofxgetter_config_json

JSON-encoded ofxgetter configuration

re_fee

regex for matching transactions as fees

re_interest_charge

regex for matching transactions as interest charges

re_interest_paid

regex for matching transactions as interest paid

re_payment

regex for matching transactions as payments

reconcile_trans

Include Transactions and OFXTransactions from this account when reconciling. Set to False to exclude accounts that are investment, payment only, or otherwise won't have a matching Transaction for each OFXTransaction.

set_balance (***kwargs*)

Create an AccountBalance object for this account and associate it with the account. Add it to the current session.

set_ofxgetter_config (*config*)

Set ofxgetter configuration.

Parameters *config* (*dict*) – ofxgetter configuration

unreconciled

Return a query to match all unreconciled Transactions for this account.

Parameters *db* (*sqlalchemy.orm.session.Session*) – active database session to use for queries

Returns query to match all unreconciled Transactions

Return type *sqlalchemy.orm.query.Query*

unreconciled_sum

Return the sum of all unreconciled transaction amounts for this account.

Returns sum of amounts of all unreconciled transactions

Return type `float`

vault_creds_path

path in Vault to read the credentials from

class `biweeklybudget.models.account.AcctType`

Bases: `enum.Enum`

Bank = 1

Cash = 4

Credit = 2

Investment = 3

Other = 5

_member_map_ = `OrderedDict([('Bank', <AcctType.Bank: 1>), ('Credit', <AcctType.Credit: 2>), ('Investment', <AcctType.Credit: 2>), ('Cash', <AcctType.Cash: 4>), ('Other', <AcctType.Other: 5>)])`

_member_names_ = ['Bank', 'Credit', 'Investment', 'Cash', 'Other']

_member_type_

alias of `object`

_value2member_map_ = {1: <AcctType.Bank: 1>, 2: <AcctType.Credit: 2>, 3: <AcctType.Credit: 2>, 4: <AcctType.Cash: 4>, 5: <AcctType.Other: 5>}

as_dict

`biweeklybudget.models.account_balance` module

class `biweeklybudget.models.account_balance.AccountBalance` (***kwargs*)

Bases: `sqlalchemy.ext.declarative.api.Base`, `biweeklybudget.models.base.ModelAsDict`

ModelAsDict

_sa_class_manager = <ClassManager of <class 'biweeklybudget.models.account_balance.AccountBalance'> at 7f868d1b1b10>

account

Relationship to *Account* this balance is for

account_id

ID of the account this balance is for

avail

Available balance

avail_date

as-of date for the available balance

id

Primary Key

ledger

Ledger balance, or investment account value, or credit card balance

ledger_date

as-of date for the ledger balance

overall_date

overall balance as of DateTime

biweeklybudget.models.base module

```
class biweeklybudget.models.base.ModelAsDict
    Bases: object

    as_dict
        Return a dict representation of the model.

        Returns model's variables/attributes

        Return type dict
```

biweeklybudget.models.budget_model module

```
class biweeklybudget.models.budget_model.Budget (**kwargs)
    Bases: sqlalchemy.ext.declarative.api.Base, biweeklybudget.models.base.
    ModelAsDict

    _sa_class_manager = <ClassManager of <class 'biweeklybudget.models.budget_model.Budget'> at 7f86886fd618>

    current_balance
        current balance for standing budgets

    description
        description

    id
        Primary Key

    is_active
        whether active or historical

    is_income
        whether this is an Income budget (True) or expense (False).

    is_periodic
        Whether the budget is standing (long-running) or periodic (resets each pay period or budget cycle)

    name
        name of the budget

    starting_balance
        starting balance for periodic budgets
```

biweeklybudget.models.ofx_statement module

```
class biweeklybudget.models.ofx_statement.OFXStatement (**kwargs)
    Bases: sqlalchemy.ext.declarative.api.Base, biweeklybudget.models.base.
    ModelAsDict

    _sa_class_manager = <ClassManager of <class 'biweeklybudget.models.ofx_statement.OFXStatement'> at 7f86886fd618>

    account
        Relationship to the Account this statement is for

    account_id
        Foreign key - Account.id - ID of the account this statement is for

    acct_type
        Textual account type, from the bank (i.e. "Checking")
```

acctid
Institution's account ID

as_of
Last OFX statement datetime

avail_bal
Available balance

avail_bal_as_of
as-of date for the available balance

bankid
FID of the Institution

brokerid
BrokerID, for investment accounts

currency
Currency definition ("USD")

file_mtime
File mtime

filename
Filename parsed from

id
Unique ID

ledger_bal
Ledger balance, or investment account value

ledger_bal_as_of
as-of date for the ledger balance

routing_number
Routing Number

type
Account Type, string corresponding to ofxparser.ofxparser.AccountType

biweeklybudget.models.ofx_transaction module

```
class biweeklybudget.models.ofx_transaction.OFXTransaction(**kwargs)
    Bases: sqlalchemy.ext.declarative.api.Base, biweeklybudget.models.base.
    ModelAsDict
    _sa_class_manager = <ClassManager of <class 'biweeklybudget.models.ofx_transaction.OFXTransaction'> at 7f8688...
    account
        Account this transaction is associated with
    account_amount
        Return the amount of the transaction, appropriately negated if the Account for this transaction has
        negate_ofx_amounts True.
        Returns amount, negated as appropriate
        Return type decimal.Decimal
```

account_id
Account ID this transaction is associated with

amount
OFX - Amount

checknum
OFX - Checknum

date_posted
OFX - Date Posted

description
Description

fitid
OFX - FITID

is_interest_charge
Account's `re_interest_charge` matched

is_interest_payment
Account's `re_interest_paid` matched

is_late_fee
Account's `re_late_fee` matched

is_other_fee
Account's `re_fee` matched

is_payment
Account's `re_payment` matched

mcc
OFX - MCC

memo
OFX - Memo

name
OFX - Name

notes
Notes

static params_from_ofxparser_transaction (*t, acct_id, stmt, cat_memo=False*)
Given an `ofxparser.ofxparser.Transaction` object, generate and return a dict of kwargs to create a new `OFXTransaction`.

Parameters

- **t** (`ofxparser.ofxparser.Transaction`) – ofxparser transaction
- **acct_id** (*int*) – OFXAccount ID
- **stmt** (`biweeklybudget.models.ofx_statement.OFXStatement`) – OFXStatement this transaction was on
- **cat_memo** (*bool*) – whether or not to concatenate OFX Memo to Name

Returns dict of kwargs to create an `OFXTransaction`

Return type `dict`

reconcile_id
The reconcile_id for the OFX Transaction

sic
OFX - SIC

statement
OFXStatement this transaction was last seen in

statement_id
OFXStatement ID this transaction was last seen in

trans_type
OFX - Transaction Type

static unreconciled (*db*)
Return a query to match all unreconciled OFXTransactions.

Parameters *db* (*sqlalchemy.orm.session.Session*) – active database session to use for queries

Returns query to match all unreconciled OFXTransactions

Return type *sqlalchemy.orm.query.Query*

biweeklybudget.models.reconcile_rule module

```
class biweeklybudget.models.reconcile_rule.ReconcileRule(**kwargs)
    Bases: sqlalchemy.ext.declarative.api.Base, biweeklybudget.models.base.ModelAsDict
    _sa_class_manager = <ClassManager of <class 'biweeklybudget.models.reconcile_rule.ReconcileRule'> at 7f868864c...
    id
        Primary Key
    is_active
        whether the rule is enabled or disabled
    name
        Name of the rule
```

biweeklybudget.models.scheduled_transaction module

```
class biweeklybudget.models.scheduled_transaction.ScheduledTransaction(**kwargs)
    Bases: sqlalchemy.ext.declarative.api.Base, biweeklybudget.models.base.ModelAsDict
    _sa_class_manager = <ClassManager of <class 'biweeklybudget.models.scheduled_transaction.ScheduledTransaction'> at 7f868864c...
    account
        Relationship - Account the transaction is against
    account_id
        ID of the account the transaction is against
    amount
        Amount of the transaction
```

budget

Relationship - *Budget* the transaction is against

budget_id

ID of the budget the transaction is against

date

Denotes a scheduled transaction that will happen once on the given date

day_of_month

Denotes a scheduled transaction that happens on the same day of each month

description

description

id

Primary Key

is_active

whether the scheduled transaction is enabled or disabled

notes

notes

num_per_period

Denotes a scheduled transaction that happens N times per pay period

recurrence_str

Return a string describing the recurrence interval. This is a string of the format YYYY-mm-dd, N per period or N(st|nd|rd|th) where N is an integer.

Returns string describing recurrence interval

Return type `str`

schedule_type

Return a string describing the type of schedule; one of `date` (a specific Date), `per period` (a number per pay period) or `monthly` (a given day of the month).

Returns string describing type of schedule

Return type `str`

validate_day_of_month (_, *value*)

validate_num_per_period (_, *value*)

biweeklybudget.models.transaction module

```
class biweeklybudget.models.transaction.Transaction(**kwargs)
```

Bases: `sqlalchemy.ext.declarative.api.Base`, `biweeklybudget.models.base.ModelAsDict`

`_sa_class_manager = <ClassManager of <class 'biweeklybudget.models.transaction.Transaction'> at 7f8688762ab8>`

account

Relationship - *Account* this transaction is against

account_id

ID of the account this transaction is against

actual_amount

Actual amount of the transaction

budget

Relationship - the *Budget* this transaction is against

budget_id

ID of the Budget this transaction is against

budgeted_amount

Budgeted amount of the transaction

date

date of the transaction

description

description

id

Primary Key

notes

free-form notes

scheduled_trans

Relationship - the *ScheduledTransaction* this Transaction was created from; set when a scheduled transaction is converted to a real one

scheduled_trans_id

ID of the ScheduledTransaction this Transaction was created from; set when a scheduled transaction is converted to a real one

static unreconciled (*db*)

Return a query to match all unreconciled Transactions.

Parameters *db* (*sqlalchemy.orm.session.Session*) – active database session to use for queries

Returns query to match all unreconciled Transactions

Return type *sqlalchemy.orm.query.Query*

biweeklybudget.models.txn_reconcile module

```
class biweeklybudget.models.txn_reconcile.TxnReconcile(**kwargs)
```

Bases: *sqlalchemy.ext.declarative.api.Base*, *biweeklybudget.models.base.ModelAsDict*

_sa_class_manager = <ClassManager of <class 'biweeklybudget.models.txn_reconcile.TxnReconcile'> at 7f86886823>

id

Primary Key

note

Notes

ofx_account_id

OFX Transaction Account ID

ofx_fitid

OFX Transaction FITID

ofx_trans

Relationship - *OFXTransaction*

reconciled_at
time when this reconcile was made

rule
Relationship - *ReconcileRule* that created this reconcile, if any.

rule_id
ReconcileRule ID; set if this reconcile was created by a rule

transaction
Relationship - *Transaction*

txn_id
Transaction ID

Submodules

biweeklybudget.backfill_ofx module

class `biweeklybudget.backfill_ofx.OfxBackfiller` (*savedir*)
Bases: `object`

Class to backfill OFX in database from files on disk.

_do_account_dir (*acct_id, acct_name, cat_memo, path*)
Handle all OFX statements in a per-account directory.

Parameters

- **acct_id** (*int*) – account database ID
- **acct_name** (*str*) – account name
- **cat_memo** (*bool*) – whether or not to concatenate OFX Memo to Name
- **path** (*str*) – absolute path to per-account directory

_do_one_file (*updater, path*)
Parse one OFX file and use OFXUpdater to upsert it into the DB.

Parameters

- **updater** (`biweeklybudget.ofxupdater.OFXUpdater`) – OFXUpdater instance for this class
- **path** (*str*) – absolute path to OFX/QFX file

run ()
Main entry point - run the backfill.

`biweeklybudget.backfill_ofx.main` ()
Main entry point - instantiate and run *OfxBackfiller*.

`biweeklybudget.backfill_ofx.parse_args` ()
Parse command-line arguments.

biweeklybudget.biweeklypayperiod module

class `biweeklybudget.biweeklypayperiod.BiweeklyPayPeriod` (*start_date, db_session*)
Bases: `object`

This object contains all logic related to working with pay periods, specifically finding a pay period for a given data, and figuring out the start and end dates of pay periods. Sure, the app is called “biweeklybudget” but there’s no reason to hard-code logic all over the place that’s this simple.

`_data`

Return the object-local data cache dict. Built it if not already present.

Returns object-local data cache

Return type `dict`

`_dict_for_sched_trans (t)`

Return a dict describing the ScheduledTransaction `t`. Called from `_trans_dict()`.

The resulting dict will have the following layout:

- `type (str)` “Transaction” or “ScheduledTransaction”
- `id (int)` the id of the object
- `date (date)` the date of the transaction, or None for per-period ScheduledTransactions
- `sched_type (str)` for ScheduledTransactions, the schedule type (“monthly”, “date”, or “per period”)
- `sched_trans_id` **None**
- `description (str)` the transaction description
- `amount (float)` the transaction amount
- `budgeted_amount` **None**
- `account_id (int)` the id of the Account the transaction is against.
- `account_name (str)` the name of the Account the transaction is against.
- `budget_id (int)` the id of the Budget the transaction is against.
- `budget_name (str)` the name of the Budget the transaction is against.
- `reconcile_id (int)` the ID of the TxnReconcile, or None

Parameters `t (ScheduledTransaction)` – ScheduledTransaction to describe

Returns common-format dict describing `t`

Return type `dict`

`_dict_for_trans (t)`

Return a dict describing the Transaction `t`. Called from `_trans_dict()`.

The resulting dict will have the following layout:

- `type (str)` “Transaction” or “ScheduledTransaction”
- `id (int)` the id of the object
- `date (date)` the date of the transaction, or None for per-period ScheduledTransactions
- `sched_type (str)` for ScheduledTransactions, the schedule type (“monthly”, “date”, or “per period”)
- `sched_trans_id (int)` for Transactions, the ScheduledTransaction `id` that it was created from, or None.
- `description (str)` the transaction description

- `amount` (**float**) the transaction amount
- `budgeted_amount` (**float**) the budgeted amount. This may be `None`.
- `account_id` (**int**) the id of the Account the transaction is against.
- `account_name` (**str**) the name of the Account the transaction is against.
- `budget_id` (**int**) the id of the Budget the transaction is against.
- `budget_name` (**str**) the name of the Budget the transaction is against.
- `reconcile_id` (**int**) the ID of the TxnReconcile, or `None`

Parameters `t` (`Transaction`) – transaction to describe

Returns common-format dict describing `t`

Return type `dict`

`_income_budget_ids`

Return a list of all `Budget` IDs for Income budgets.

Returns list of income budget IDs

Return type `list`

`_make_budget_sums()`

Find the sums of all transactions per periodic budget ID ; return a dict where keys are budget IDs and values are per-budget dicts containing:

- `budget_amount` (*float*) - the periodic budget *starting_balance*.
- `allocated` (*float*) - sum of all *ScheduledTransaction* and *Transaction* amounts against the budget this period. For actual transactions, we use the *budgeted_amount* if present (not `None`).
- `spent` (*float*) - the sum of all actual *Transaction* amounts against the budget this period.
- `trans_total` (*float*) - the sum of spent amounts for Transactions that have them, or allocated amounts for ScheduledTransactions.
- `remaining` (*float*) - the remaining amount in the budget. This is `budget_amount` minus the greater of `allocated` or `trans_total`. For income budgets, this is always positive.

Returns dict of dicts, transaction sums and amounts per budget

Return type `dict`

`_make_combined_transactions()`

Combine all Transactions and ScheduledTransactions from `self._data_cache` into one ordered list of similar dicts, adding dates to the monthly ScheduledTransactions as appropriate and excluding ScheduledTransactions that have been converted to real Transactions. Store the finished list back into `self._data_cache`.

`_make_overall_sums()`

Return a dict describing the overall sums for this pay period, namely:

- `allocated` (*float*) total amount allocated via *ScheduledTransaction*, *Transaction* (counting the *budgeted_amount* for Transactions that have one), or *Budget* (not counting income budgets).
- `spent` (*float*) total amount actually spent via *Transaction*.

- `income` (*float*) total amount of income allocated this pay period. Calculated value (from `_make_budget_sums()` / `self._data_cache['budget_sums']`) should be negative, but is returned as its positive inverse (absolute value).
- `remaining` (*float*) income minus the greater of allocated or spent

Returns dict describing sums for the pay period

Return type dict

`_scheduled_transactions_date()`

Return a Query for all *ScheduledTransaction* defined by date (`schedule_type == "date"`) for this pay period.

Returns Query matching all ScheduledTransactions defined by date, for this pay period.

Return type `sqlalchemy.orm.query.Query`

`_scheduled_transactions_monthly()`

Return a Query for all *ScheduledTransaction* defined by day of month (`schedule_type == "monthly"`) for this pay period.

Returns Query matching all ScheduledTransactions defined by day of month (monthly) for this period.

Return type `sqlalchemy.orm.query.Query`

`_scheduled_transactions_per_period()`

Return a Query for all *ScheduledTransaction* defined by number per period (`schedule_type == "per period"`) for this pay period.

Returns Query matching all ScheduledTransactions defined by number per period, for this pay period.

Return type `sqlalchemy.orm.query.Query`

`_trans_dict(t)`

Given a Transaction or ScheduledTransaction, return a dict of a common format describing the object.

The resulting dict will have the following layout:

- `type` (**str**) "Transaction" or "ScheduledTransaction"
- `id` (**int**) the id of the object
- `date` (**date**) the date of the transaction, or None for per-period ScheduledTransactions
- `sched_type` (**str**) for ScheduledTransactions, the schedule type ("monthly", "date", or "per period")
- `sched_trans_id` (**int**) for Transactions, the ScheduledTransaction id that it was created from, or None.
- `description` (**str**) the transaction description
- `amount` (**float**) the transaction amount
- `budgeted_amount` (**float**) the budgeted amount. This may be None.
- `account_id` (**int**) the id of the Account the transaction is against.
- `account_name` (**str**) the name of the Account the transaction is against.
- `budget_id` (**int**) the id of the Budget the transaction is against.
- `budget_name` (**str**) the name of the Budget the transaction is against.

- `reconcile_id` (**int**) the ID of the TxnReconcile, or None

Parameters `t` (*Transaction* or *ScheduledTransaction*) – the object to return a dict for

Returns dict describing `t`

Return type dict

`_transactions()`

Return a Query for all *Transaction* for this pay period.

Returns Query matching all Transactions for this pay period

Return type sqlalchemy.orm.query.Query

`budget_sums`

Return a dict of budget sums; the return value of `_make_budget_sums()`.

Returns dict of dicts, transaction sums and amounts per budget

Return type dict

`end_date`

Return the date of the last day in this pay period. The pay period is generally considered to end at the last instant (i.e. 23:59:59) of this date.

Returns last date in the pay period

Return type datetime.date

`filter_query(query, date_prop)`

Filter query for `date_prop` in this pay period. Returns a copy of the query.

e.g. to filter an existing query of *OFXTransaction* for the BiweeklyPayPeriod starting on 2017-01-14:

```
q = # some query here
p = BiweeklyPayPeriod(date(2017, 1, 14))
q = p.filter_query(q, OFXTransaction.date_posted)
```

Parameters

- **`query`** (sqlalchemy.orm.query.Query) – The query to filter
- **`date_prop`** – the Model's date property, to filter on.

Returns the filtered query

Return type sqlalchemy.orm.query.Query

`next`

Return the BiweeklyPayPeriod following this one.

Returns next BiweeklyPayPeriod after this one

Return type *BiweeklyPayPeriod*

`overall_sums`

Return a dict of overall sums; the return value of `_make_overall_sums()`.

Returns dict describing sums for the pay period

Return type dict

static `period_for_date(dt, db_session)`

Given a datetime, return the `BiweeklyPayPeriod` instance describing the pay period containing this date.

Todo

This is a very naive, poorly-performing implementation.

Parameters

- **dt** (`datetime` or `date`) – datetime or date to find the pay period for
- **db_session** (`sqlalchemy.orm.session.Session`) – active database session to use for queries

Returns `BiweeklyPayPeriod` containing the specified date

Return type `BiweeklyPayPeriod`

period_interval

Return the interval between `BiweeklyPayPeriods` as a `timedelta`.

Returns interval between `BiweeklyPayPeriods`

Return type `datetime.timedelta`

period_length

Return the length of a `BiweeklyPayPeriod`; this is calculated as `period_interval` minus one second.

Returns length of one `BiweeklyPayPeriod`

Return type `datetime.timedelta`

previous

Return the `BiweeklyPayPeriod` preceding this one.

Returns previous `BiweeklyPayPeriod` before this one

Return type `BiweeklyPayPeriod`

start_date

Return the starting date for this pay period. The period is generally considered to start at midnight (00:00) of this date.

Returns start date for pay period

Return type `datetime.date`

transactions_list

Return an ordered list of dicts, each representing a transaction for this pay period. Dicts have keys and values as described in `_trans_dict()`.

Returns ordered list of transaction dicts

Return type `list`

biweeklybudget.cliutils module

`biweeklybudget.cliutils.set_log_debug(logger)`

set logger level to `DEBUG`, and debug-level output format, via `set_log_level_format()`.

`biweeklybudget.cliutils.set_log_info(logger)`
 set logger level to INFO via `set_log_level_format()`.

`biweeklybudget.cliutils.set_log_level_format(logger, level, format)`
 Set logger level and format.

Parameters

- **logger** (`logging.Logger`) – the logger object to set on
- **level** (`int`) – logging level; see the `logging` constants.
- **format** (`str`) – logging formatter format string

biweeklybudget.db module

`biweeklybudget.db._alembic_get_current_rev(config, script)`
 Works sorta like `alembic.command.current`

Parameters `config` – alembic Config

Returns current revision

Return type `str`

`biweeklybudget.db.cleanup_db()`
 This must be called from all scripts, using
`atexit.register(cleanup_db)`

`biweeklybudget.db.db_session = <sqlalchemy.orm.scoping.scoped_session object>`
`sqlalchemy.orm.scoping.scoped_session` session

`biweeklybudget.db.engine = Engine(sqlite:///memory:)`
 The database engine object; return value of `sqlalchemy.create_engine()`.

`biweeklybudget.db.init_db()`
 Initialize the database; call `sqlalchemy.schema.MetaData.create_all()` on the metadata object.

`biweeklybudget.db.upsert_record(model_class, key_fields, **kwargs)`
 Upsert a record in the database.

`key_fields` is either a string primary key field name (a key in the `kwargs` dict) or a list or tuple of string primary key field names, for compound keys.

If a record can be found matching these keys, it will be updated and committed. If not, a new one will be inserted. Either way, the record is returned.

`sqlalchemy.orm.session.Session.commit()` is **NOT** called.

Parameters

- **model_class** (`biweeklybudget.models.base.ModelAsDict`) – the class of model to insert/update
- **key_fields** – The field name(s) (keys in `kwargs`) that make up the primary key. This can be a single string, or a list or tuple of strings for compound keys. The values for these key fields **MUST** be included in `kwargs`.
- **kwargs** (`dict`) – arguments to provide to the model class constructor, or to update if there is an existing record matching the key.

Returns inserted or updated record; type is an instance of `model_class`

biweeklybudget.db_event_handlers module

`biweeklybudget.db_event_handlers.handle_before_flush(session, flush_context, instances)`

Hook into `before_flush` (`sqlalchemy.orm.events.SessionEvents.before_flush()`) on the DB session, to handle updates that need to be made before persisting data. Currently, this method just calls a number of other methods to handle specific cases:

- `handle_new_transaction()`

Parameters

- **session** (`sqlalchemy.orm.session.Session`) – current database session
- **flush_context** (`sqlalchemy.orm.session.UOWTransaction`) – internal SQLAlchemy object
- **instances** – deprecated

`biweeklybudget.db_event_handlers.handle_new_transaction(session)`

`before_flush` event handler (`sqlalchemy.orm.events.SessionEvents.before_flush()`) on the DB session, to handle creation of *new* Transactions. For updates to existing Transactions, we rely on `handle_trans_amount_change()`.

If the Transaction's *budget* is a *Budget* with *is_periodic* False (i.e. a standing budget), update the Budget's *current_balance* for this transaction.

Parameters **session** (`sqlalchemy.orm.session.Session`) – current database session

`biweeklybudget.db_event_handlers.handle_trans_amount_change(**kwargs)`

Handle change of *Transaction.actual_amount* for existing instances (*id* is not None). For new instances, we rely on `handle_new_transaction()` called via `handle_before_flush()`.

If the Transaction's *budget* is a *Budget* with *is_periodic* False (i.e. a standing budget), update the Budget's *current_balance* for this transaction.

See: `sqlalchemy.orm.events.AttributeEvents.set()`

Parameters **kwargs** (*dict*) – keyword arguments

`biweeklybudget.db_event_handlers.init_event_listeners(db_session)`

Initialize/register all SQLAlchemy event listeners.

See <http://docs.sqlalchemy.org/en/latest/orm/events.html>

Parameters **db_session** (`sqlalchemy.orm.session.Session`) – the Database Session

biweeklybudget.initdb module

`biweeklybudget.initdb.main()`

`biweeklybudget.initdb.parse_args()`

biweeklybudget.load_data module

`biweeklybudget.load_data.main()`

`biweeklybudget.load_data.parse_args()`

biweeklybudget.ofxgetter module

```
class biweeklybudget.ofxgetter.OfxGetter(savedir='./')
```

```
    Bases: object
```

```
    _get_ofx_scraper(account_name, days=30)
```

Get OFX via a ScreenScraper subclass.

Parameters

- **account_name** (*str*) – account name
- **days** (*int*) – number of days of data to download

Returns OFX string

Return type *str*

```
    _ofx_to_db(account_name, fname, ofxdata)
```

Put OFX Data to the DB

Parameters

- **account_name** (*str*) – account name to download
- **ofxdata** (*str*) – raw OFX data
- **fname** (*str*) – filename OFX was written to

```
    _write_ofx_file(account_name, ofxdata)
```

Write OFX data to a file.

Parameters

- **account_name** (*str*) – account name
- **ofxdata** (*str*) – raw OFX data string

Returns name of the file that was written

Return type *str*

```
static accounts()
```

Return a sorted list of all Account objects that are for ofxgetter.

```
get_ofx(account_name, write_to_file=True, days=30)
```

Download OFX from the specified account. Return it as a string.

Parameters

- **account_name** (*str*) – account name to download
- **write_to_file** (*bool*) – if True, also write to a file named “<account_name>_<date stamp>.ofx”
- **days** (*int*) – number of days of data to download

Returns OFX string

Return type *str*

```
biweeklybudget.ofxgetter.main()
```

```
biweeklybudget.ofxgetter.parse_args()
```

biweeklybudget.ofxupdater module**exception** `biweeklybudget.ofxupdater.DuplicateFileException`Bases: `exceptions.Exception`

Exception raised when trying to parse a file that has already been parsed for the Account (going by the OFX signon date).

class `biweeklybudget.ofxupdater.OFXUpdater(acct_id, acct_name, cat_memo=False)`Bases: `object`

Class to wrap updating the database with a parsed OFX file.

_create_statement (*ofx, mtime, filename*)

Create an OFXStatement for this OFX file. If one already exists with the same account and filename, raise DuplicateFileException.

Parameters

- **ofx** (`ofxparse.ofxparse.Ofx`) – Ofx instance for parsed file
- **mtime** (`datetime.datetime`) – OFX file modification time (or current time)
- **filename** (*str*) – OFX file name

Returns the OFXStatement object**Return type** `biweeklybudget.models.ofx_statement.OFXStatement`**Raises** DuplicateFileException**_update_bank_or_credit** (*ofx, stmt*)

Update a single OFX file for this Bank or Credit account.

Parameters

- **ofx** (`ofxparse.ofxparse.Ofx`) – Ofx instance for parsed file
- **stmt** (`biweeklybudget.models.ofx_statement.OFXStatement`) – the OFXStatement for this statement

Returns the OFXStatement object**Return type** `biweeklybudget.models.ofx_statement.OFXStatement`**_update_investment** (*ofx, stmt*)

Update a single OFX file for this Investment account.

Parameters

- **ofx** (`ofxparse.ofxparse.Ofx`) – Ofx instance for parsed file
- **stmt** (`biweeklybudget.models.ofx_statement.OFXStatement`) – the OFXStatement for this statement

Returns the OFXStatement object**Return type** `biweeklybudget.models.ofx_statement.OFXStatement`**update** (*ofx, mtime=None, filename=None*)

Update a single OFX file for this account.

Parameters

- **ofx** (`ofxparse.ofxparse.Ofx`) – Ofx instance for parsed file
- **mtime** (`datetime.datetime`) – OFX file modification time (or current time)

- **filename** (*str*) – OFX file name

Returns the OFXStatement created by this run

Return type *biweeklybudget.models.ofx_statement.OFXStatement*

biweeklybudget.screenscraper module

class `biweeklybudget.screenscraper.ScreenScraper` (*savedir='./', screenshot=False*)

Bases: `object`

Base class for screen-scraping bank/financial websites.

do_screenshot ()

take a debug screenshot

doc_readystate_is_complete (*foo*)

return true if document is ready/complete, false otherwise

error_screenshot (*fname=None*)

get_browser (*browser_name*)

get a webdriver browser instance

jquery_finished (*foo*)

return true if `jQuery.active == 0` else false

load_cookies (*cookie_file*)

Load cookies from a JSON cookie file on disk. This file is not the format used natively by PhantomJS, but rather the JSON-serialized representation of the dict returned by `selenium.webdriver.remote.webdriver.WebDriver.get_cookies()`.

Cookies are loaded via `selenium.webdriver.remote.webdriver.WebDriver.add_cookie()`

Parameters `cookie_file` (*str*) – path to the cookie file on disk

save_cookies (*cookie_file*)

Save cookies to a JSON cookie file on disk. This file is not the format used natively by PhantomJS, but rather the JSON-serialized representation of the dict returned by `selenium.webdriver.remote.webdriver.WebDriver.get_cookies()`.

Parameters `cookie_file` (*str*) – path to the cookie file on disk

wait_for_ajax_load (*timeout=20*)

Function to wait for an ajax event to finish and trigger page load, like the Janrain login form.

Pieced together from <http://stackoverflow.com/a/15791319>

timeout is in seconds

xhr_get_url (*url*)

use JS to download a given URL, return its contents

xhr_post_urlencoded (*url, data, headers={}*)

use JS to download a given URL, return its contents

biweeklybudget.settings module

`biweeklybudget.settings.DB_CONNSTRING = 'sqlite:///memory:'`
string - SQLAlchemy database connection string. See the [SQLAlchemy Database URLs docs](#) for further information.

`biweeklybudget.settings.DEFAULT_ACCOUNT_ID = 1`
int - Account ID to show first in dropdown lists. This must be the database ID of a valid account.

`biweeklybudget.settings.PAY_PERIOD_START_DATE = datetime.date(2017, 3, 17)`
`datetime.date` - The starting date of one pay period (generally the first pay period represented in data in this app). The dates of all pay periods will be determined based on an interval from this date. This must be specified in Y-m-d format (i.e. parsable by `datetime.datetime.strptime()` with `%Y-%m-%d` format).

`biweeklybudget.settings.RECONCILE_BEGIN_DATE = datetime.date(2017, 1, 1)`
`datetime.date` - When listing unreconciled transactions that need to be reconciled, any transaction before this date will be ignored. This must be specified in Y-m-d format (i.e. parsable by `datetime.datetime.strptime()` with `%Y-%m-%d` format).

`biweeklybudget.settings.STALE_DATA_TIMEDELTA = datetime.timedelta(2)`
`datetime.timedelta` - Time interval beyond which OFX data for accounts will be considered old/stale. This must be specified as a number (integer) that will be converted to a number of days.

`biweeklybudget.settings.STATEMENTS_SAVE_PATH = '/home/docs/ofx'`
string - (optional) Filesystem path to download OFX statements to, and for `backfill_ofx` to read them from.

`biweeklybudget.settings.TOKEN_PATH = 'vault_token.txt'`
string - (optional) Filesystem path to read Vault token from, for OFX credentials.

`biweeklybudget.settings.VAULT_ADDR = 'http://127.0.0.1:8200'`
string - (optional) Address to connect to Vault at, for OFX credentials.

biweeklybudget.settings_example module

`biweeklybudget.settings_example.DB_CONNSTRING = 'sqlite:///memory:'`
SQLAlchemy database connection string. Note that the value given in generated documentation is the value used in TravisCI, not the real default.

`biweeklybudget.settings_example.DEFAULT_ACCOUNT_ID = 1`
Account ID to show first in dropdown lists

`biweeklybudget.settings_example.PAY_PERIOD_START_DATE = datetime.date(2017, 3, 17)`
The starting date of one pay period. The dates of all pay periods will be determined based on an interval from this date.

`biweeklybudget.settings_example.RECONCILE_BEGIN_DATE = datetime.date(2017, 1, 1)`
When listing unreconciled transactions that need to be reconciled, any `OFXTransaction` before this date will be ignored.

`biweeklybudget.settings_example.STALE_DATA_TIMEDELTA = datetime.timedelta(2)`
`datetime.timedelta` beyond which OFX data will be considered old

`biweeklybudget.settings_example.STATEMENTS_SAVE_PATH = '/home/docs/ofx'`
Path to download OFX statements to, and for `backfill_ofx` to read them from

`biweeklybudget.settings_example.TOKEN_PATH = 'vault_token.txt'`
Path to read Vault token from, for OFX credentials

`biweeklybudget.settings_example.VAULT_ADDR = 'http://127.0.0.1:8200'`
Address to connect to Vault at, for OFX credentials

biweeklybudget.utils module

exception `biweeklybudget.utils.SecretMissingException` (*path*)

Bases: `exceptions.Exception`

class `biweeklybudget.utils.Vault` (*addr='http://127.0.0.1:8200', token_path='vault_token.txt'*)

Bases: `object`

Provides simpler access to Vault

read (*secret_path*)

Read and return a secret from Vault. Return only the data portion.

Parameters `secret_path` (*str*) – path to read in Vault

Returns secret data

Return type `dict`

`biweeklybudget.utils.date_suffix` (*n*)

Given an integer day of month ($1 \leq n \leq 31$), return that number with the appropriate suffix (stndnrdlth).

From: <http://stackoverflow.com/a/5891598/211734>

Parameters `n` (*int*) – Integer day of month

Returns `n` with the appropriate suffix

Return type `str`

`biweeklybudget.utils.dtnow` ()

Return the current datetime as a timezone-aware DateTime object in UTC.

Returns current datetime

Return type `datetime.datetime`

`biweeklybudget.utils.fix_werkzeug_logger` ()

Remove the werkzeug logger StreamHandler (call from `app.py`).

With Werkzeug at least as of 0.12.1, `werkzeug.internal.log` sets up its own StreamHandler if logging isn't already configured. Because we're using the `flask` command line wrapper, that will ALWAYS be imported (and executed) before we can set up our own logger. As a result, to fix the duplicate log messages, we have to go back and remove that StreamHandler.

`biweeklybudget.utils.in_directory` (**args, **kws*)

biweeklybudget.version module

UI JavaScript Docs

Files

`jsdoc.budget_transfer_modal`

File: `biweeklybudget/flaskapp/static/js/budget_transfer_modal.js`

budgetTransferDivForm ()

Generate the HTML for the form on the Modal

budgetTransferModal()

Show the modal popup for transferring between budgets. Uses *budgetTransferDivForm()* to generate the form.

jsdoc.budgets_modal

File: biweeklybudget/flaskapp/static/js/budgets_modal.js

budgetModal(id, dataTableObj)

Show the modal popup, populated with information for one Budget. Uses *budgetModalDivFillAndShow()* as ajax callback.

Arguments

- **id** (*number*) – the ID of the Budget to show modal for, or null to show a modal to add a new Budget.
- **dataTableObj** (*Object* | *null*) – passed on to *handleForm()*

budgetModalDivFillAndShow(msg)

Ajax callback to fill in the modalDiv with data on a budget. Callback for ajax call in *budgetModal()*.

budgetModalDivForm()

Generate the HTML for the form on the Modal

budgetModalDivHandleType()

Handle change of the “Type” radio buttons on the modal

jsdoc.custom

File: biweeklybudget/flaskapp/static/js/custom.js

fmt_currency(value)

Format a float as currency

Arguments

- **value** (*number*) – the number to format

Returns **string** – The number formatted as currency

fmt_null(o)

Format a null object as “ ”

Arguments

- **o** (*Object* | *null*) – input value

Returns **Object|string** – o if not null, if null

isoformat(d)

Format a javascript Date as ISO8601 YYYY-MM-DD

Arguments

- **d** (*Date*) – the date to format

Returns **string** – YYYY-MM-DD

jsdoc.forms

File: biweeklybudget/flaskapp/static/js/forms.js

handleForm (*container_id*, *form_id*, *post_url*, *dataTableObj*)

Generic function to handle form submission with server-side validation.

See the Python server-side code for further information.

Arguments

- **container_id** (*string*) – The ID of the container element (div) that is the visual parent of the form. On successful submission, this element will be emptied and replaced with a success message.
- **form_id** (*string*) – The ID of the form itself.
- **post_url** (*string*) – Relative URL to post form data to.
- **dataTableObj** (*Object*) – passed on to `handleFormSubmitted()`

handleFormError (*jqXHR*, *textStatus*, *errorThrown*, *container_id*, *form_id*)

Handle an error in the HTTP request to submit the form.

handleFormSubmitted (*data*, *container_id*, *form_id*, *dataTableObj*)

Handle the response from the API URL that the form data is POSTed to.

This should either display a success message, or one or more error messages.

Arguments

- **data** (*Object*) – response data
- **container_id** (*string*) – the ID of the modal container on the page
- **form_id** (*string*) – the ID of the form on the page
- **dataTableObj** (*Object*) – A reference to the DataTable on the page, that needs to be refreshed. If null, reload the whole page. If a function, call that function. If false, do nothing.

isFunction (*functionToCheck*)

Return True if *functionToCheck* is a function, False otherwise.

From: <http://stackoverflow.com/a/7356528/211734>

Arguments

- **functionToCheck** (*Object*) – The object to test.

serializeForm (*form_id*)

Given the ID of a form, return an Object (hash/dict) of all data from it, to POST to the server.

Arguments

- **form_id** (*string*) – The ID of the form itself.

jsdoc.ofx

File: biweeklybudget/flaskapp/static/js/ofx.js

ofxTransModal (*acct_id*, *fitid*)

Show the modal popup, populated with information for one OFX Transaction.

jsdoc.payperiod_modal

File: biweeklybudget/flaskapp/static/js/payperiod_modal.js

schedToTransModal (*id*, *payperiod_start_date*)

Show the Scheduled Transaction to Transaction modal popup. This function calls `schedToTransModalDivForm()` to generate the form HTML, `schedToTransModalDivFillAndShow()` to populate the form for editing, and `handleForm()` to handle the Submit action.

Arguments

- **id** (*number*) – the ID of the ScheduledTransaction to show a modal for.
- **payperiod_start_date** (*string*) – The Y-m-d starting date of the pay period.

schedToTransModalDivFillAndShow (*msg*)

Ajax callback to fill in the modalDiv with data on a budget.

schedToTransModalDivForm ()

Generate the HTML for the form on the Modal

skipSchedTransModal (*id*, *payperiod_start_date*)

Show the Skip Scheduled Transaction modal popup. This function calls `skipSchedTransModalDivForm()` to generate the form HTML, `skipSchedTransModalDivFillAndShow()` to populate the form for editing, and `handleForm()` to handle the Submit action.

Arguments

- **id** (*number*) – the ID of the ScheduledTransaction to show a modal for.
- **payperiod_start_date** (*string*) – The Y-m-d starting date of the pay period.

skipSchedTransModalDivFillAndShow (*msg*)

Ajax callback to fill in the modalDiv with data on a budget.

skipSchedTransModalDivForm ()

Generate the HTML for the form on the Modal

jsdoc.reconcile

File: biweeklybudget/flaskapp/static/js/reconcile.js

clean_fitid (*fitid*)

Given an OFXTransaction fitid, return a “clean” (alphanumeric) version of it, suitable for use as an HTML element id.

Arguments

- **fitid** (*String*) – original, unmodified OFXTransaction fitid.

makeTransFromOfx (*acct_id*, *fitid*)

Link function to create a Transaction from a specified OFXTransaction, and then reconcile them.

Arguments

- **acct_id** (*Integer*) – the OFXTransaction account ID
- **fitid** (*String*) – the OFXTransaction fitid

makeTransSaveCallback (*data*, *acct_id*, *fitid*)

Callback for the “Save” button on the Transaction modal created by `makeTransFromOfx()`. Displays the new Transaction at the bottom of the Transactions list, then reconciles it with the original OFXTransaction

Arguments

- **data** (*Object*) – response data from POST to /forms/transaction
- **acct_id** (*Integer*) – the OFXTransaction account ID
- **fitid** (*String*) – the OFXTransaction fitid

reconcileDoUnreconcile (*trans_id*, *acct_id*, *fitid*)

Unreconcile a reconciled OFXTransaction/Transaction. This removes `trans_id` from the `reconciled` variable, empties the Transaction div’s reconciled div, and shows the OFX div.

Arguments

- **trans_id** (*Integer*) – the transaction id
- **acct_id** (*Integer*) – the account id
- **fitid** (*String*) – the FITID

reconcileDoUnreconcileNoOfx (*trans_id*)

Unreconcile a reconciled NoOFX Transaction. This removes `trans_id` from the `reconciled` variable and empties the Transaction div’s reconciled div.

Arguments

- **trans_id** (*Integer*) – the transaction id

reconcileGetOFX ()

Show unreconciled OFX transactions in the proper div. Empty the div, then load transactions via ajax. Uses `reconcileShowOFX()` as the ajax callback.

reconcileGetTransactions ()

Show unreconciled transactions in the proper div. Empty the div, then load transactions via ajax. Uses `reconcileShowTransactions()` as the ajax callback.

reconcileHandleSubmit ()

Handle click of the Submit button on the reconcile view. This POSTs to /ajax/reconcile via ajax. Feedback is provided by appending a div with id `reconcile-msg` to `div#notifications-row/div.col-lg-12`.

reconcileOfxDiv (*trans*)

Generate a div for an individual OFXTransaction, to display on the reconcile view.

Arguments

- **ofxtrans** (*Object*) – ajax JSON object representing one OFXTransaction

reconcileShowOFX (*data*)

Ajax callback handler for `reconcileGetOFX()`. Display the returned data in the proper div.

Arguments

- **data** (*Object*) – ajax response (JSON array of OFXTransaction Objects)

reconcileShowTransactions (*data*)

Ajax callback handler for `reconcileGetTransactions()`. Display the returned data in the proper div.

Sets each Transaction div as droppable, using `reconcileTransHandleDropEvent()` as the drop event handler and `reconcileTransDroppableAccept()` to test if a draggable is droppable on the element.

Arguments

- **data** (*Object*) – ajax response (JSON array of Transaction Objects)

reconcileTransDiv (*trans*)

Generate a div for an individual Transaction, to display on the reconcile view.

Arguments

- **trans** (*Object*) – ajax JSON object representing one Transaction

reconcileTransDroppableAccept (*drag*)

Accept function for droppables, to determine if a given draggable can be dropped on it.

Arguments

- **drag** (*Object*) – the draggable element being dropped.

reconcileTransHandleDropEvent (*event, ui*)

Handler for Drop events on reconcile Transaction divs. Setup as handler via `reconcileShowTransactions()`. This just gets the draggable and the target from the event and ui, and then passes them on to `reconcileTransactions()`.

Arguments

- **event** (*Object*) – the drop event
- **ui** (*Object*) – the UI element, containing the draggable

reconcileTransNoOfx (*trans_id, note*)

Reconcile a Transaction without a matching OFXTransaction. Called from the Save button handler in `transNoOfx()`.

reconcileTransactions (*ofx_div, target*)

Reconcile a transaction; move the divs and other elements as necessary, and updated the `reconciled` variable.

Arguments

- **ofx_div** (*Object*) – the OFXTransaction div element (draggable)
- **target** (*Object*) – the Transaction div (drop target)

transModalOfxFillAndShow (*data*)

Callback for the GET /ajax/ofx/<acct_id>/<ftid> from `makeTransFromOfx()`. Receives the OFXTransaction data and populates it into the Transaction modal form.

Arguments

- **data** (*Object*) – OFXTransaction response data

transNoOfx (*trans_id*)

Show the modal for reconciling a Transaction without a matching OFXTransaction. Calls `transNoOfxDivForm()` to generate the modal form div content. Uses an inline function to handle the save action, which calls `reconcileTransNoOfx()` to perform the reconcile action.

Arguments

- **trans_id** (*number*) – the ID of the Transaction

transNoOfxDivForm (*trans_id*)

Generate the modal form div content for the modal to reconcile a Transaction without a matching OFXTransaction. Called by `transNoOfx()`.

Arguments

- **trans_id** (*number*) – the ID of the Transaction

updateReconcileTrans (*trans_id*)

Trigger update of a single Transaction on the reconcile page.

Arguments

- **trans_id** (*Integer*) – the Transaction ID to update.

jsdoc.reconcile_modal

File: biweeklybudget/flaskapp/static/js/reconcile_modal.js

txnReconcileModal (*id*)

Show the TxnReconcile modal popup. This function calls *txnReconcileModalDiv()* to generate the HTML.

Arguments

- **id** (*number*) – the ID of the TxnReconcile to show a modal for.

txnReconcileModalDiv (*msg*)

Ajax callback to generate the modal HTML with reconcile information.

jsdoc.scheduled_modal

File: biweeklybudget/flaskapp/static/js/scheduled_modal.js

schedModal (*id*, *dataTableObj*)

Show the ScheduledTransaction modal popup, optionally populated with information for one ScheduledTransaction. This function calls *schedModalDivForm()* to generate the form HTML, *schedModalDivFillAndShow()* to populate the form for editing, and *handleForm()* to handle the Submit action.

Arguments

- **id** (*number*) – the ID of the ScheduledTransaction to show a modal for, or null to show modal to add a new ScheduledTransaction.
- **dataTableObj** (*Object* | *null*) – passed on to *handleForm()*

schedModalDivFillAndShow (*msg*)

Ajax callback to fill in the modalDiv with data on a budget.

schedModalDivForm ()

Generate the HTML for the form on the Modal

schedModalDivHandleType ()

Handle change of the “Type” radio buttons on the modal

jsdoc.transactions_modal

File: biweeklybudget/flaskapp/static/js/transactions_modal.js

transModal (*id*, *dataTableObj*)

Show the ScheduledTransaction modal popup, optionally populated with information for one ScheduledTransaction. This function calls *schedModalDivForm()* to generate the form HTML, *schedModalDivFillAndShow()* to populate the form for editing, and *handleForm()* to handle the Submit action.

Arguments

- **id** (*number*) – the ID of the ScheduledTransaction to show a modal for, or null to show modal to add a new ScheduledTransaction.
- **dataTableObj** (*Object / null*) – passed on to *handleForm()*

transModalDivFillAndShow (*msg*)

Ajax callback to fill in the modalDiv with data on a budget.

transModalDivForm ()

Generate the HTML for the form on the Modal

CHAPTER 6

Indices and tables

- `genindex`
- `modindex`
- `search`

b

- `biweeklybudget`, 31
- `biweeklybudget.backfill_ofx`, 43
- `biweeklybudget.biweeklypayperiod`, 43
- `biweeklybudget.cliutils`, 48
- `biweeklybudget.db`, 49
- `biweeklybudget.db_event_handlers`, 50
- `biweeklybudget.flaskapp`, 31
- `biweeklybudget.flaskapp.cli_commands`, 32
- `biweeklybudget.flaskapp.jsonencoder`, 33
- `biweeklybudget.flaskapp.notifications`, 33
- `biweeklybudget.initdb`, 50
- `biweeklybudget.load_data`, 50
- `biweeklybudget.models`, 34
- `biweeklybudget.models.account`, 34
- `biweeklybudget.models.account_balance`, 36
- `biweeklybudget.models.base`, 37
- `biweeklybudget.models.budget_model`, 37
- `biweeklybudget.models.ofx_statement`, 37
- `biweeklybudget.models.ofx_transaction`, 38
- `biweeklybudget.models.reconcile_rule`, 40
- `biweeklybudget.models.scheduled_transaction`, 40
- `biweeklybudget.models.transaction`, 41
- `biweeklybudget.models.txn_reconcile`, 42
- `biweeklybudget.ofxgetter`, 51
- `biweeklybudget.ofxupdater`, 52
- `biweeklybudget.screenscraper`, 53
- `biweeklybudget.settings`, 54
- `biweeklybudget.settings_example`, 54
- `biweeklybudget.utils`, 55
- `biweeklybudget.version`, 55

Symbols

<code>_alembic_get_current_rev()</code> (in module <code>biweeklybudget.db</code>), 49	<code>_member_type_</code> (biweeklybudget.models.account.AcctType attribute), 36
<code>_create_statement()</code> (biweeklybudget.ofxupdater.OFXUpdater method), 52	<code>_ofx_to_db()</code> (biweeklybudget.ofxgetter.OfxGetter method), 51
<code>_data</code> (biweeklybudget.biweeklypayperiod.BiweeklyPayPeriod attribute), 44	<code>_sa_class_manager</code> (biweeklybudget.models.account.Account attribute), 34
<code>_dict_for_sched_trans()</code> (biweeklybudget.biweeklypayperiod.BiweeklyPayPeriod method), 44	<code>_sa_class_manager</code> (biweeklybudget.models.account_balance.AccountBalance attribute), 36
<code>_dict_for_trans()</code> (biweeklybudget.biweeklypayperiod.BiweeklyPayPeriod method), 44	<code>_sa_class_manager</code> (biweeklybudget.models.budget_model.Budget attribute), 37
<code>_do_account_dir()</code> (biweeklybudget.backfill_ofx.OfxBackfiller method), 43	<code>_sa_class_manager</code> (biweeklybudget.models.ofx_statement.OFXStatement attribute), 37
<code>_do_one_file()</code> (biweeklybudget.backfill_ofx.OfxBackfiller method), 43	<code>_sa_class_manager</code> (biweeklybudget.models.ofx_transaction.OFXTransaction attribute), 38
<code>_get_ofx_scraper()</code> (biweeklybudget.ofxgetter.OfxGetter method), 51	<code>_sa_class_manager</code> (biweeklybudget.models.reconcile_rule.ReconcileRule attribute), 40
<code>_income_budget_ids</code> (biweeklybudget.biweeklypayperiod.BiweeklyPayPeriod attribute), 45	<code>_sa_class_manager</code> (biweeklybudget.models.scheduled_transaction.ScheduledTransaction attribute), 40
<code>_make_budget_sums()</code> (biweeklybudget.biweeklypayperiod.BiweeklyPayPeriod method), 45	<code>_sa_class_manager</code> (biweeklybudget.models.transaction.Transaction attribute), 41
<code>_make_combined_transactions()</code> (biweeklybudget.biweeklypayperiod.BiweeklyPayPeriod method), 45	<code>_sa_class_manager</code> (biweeklybudget.models.txn_reconcile.TxnReconcile attribute), 42
<code>_make_overall_sums()</code> (biweeklybudget.biweeklypayperiod.BiweeklyPayPeriod method), 45	<code>_scheduled_transactions_date()</code> (biweeklybudget.biweeklypayperiod.BiweeklyPayPeriod method), 46
<code>_member_map_</code> (biweeklybudget.models.account.AcctType attribute), 36	<code>_scheduled_transactions_monthly()</code> (biweeklybudget.biweeklypayperiod.BiweeklyPayPeriod method), 46
<code>_member_names_</code> (biweeklybudget.models.account.AcctType attribute), 36	<code>_scheduled_transactions_per_period()</code> (biweeklybudget.biweeklypayperiod.BiweeklyPayPeriod method), 46

`_trans_dict()` (biweeklybudget.get.biweeklypayperiod.BiweeklyPayPeriod method), 46

`_transactions()` (biweeklybudget.get.biweeklypayperiod.BiweeklyPayPeriod method), 47

`_update_bank_or_credit()` (biweeklybudget.get.ofxupdater.OFXUpdater method), 52

`_update_investment()` (biweeklybudget.get.ofxupdater.OFXUpdater method), 52

`_value2member_map_` (biweeklybudget.get.models.account.AcctType attribute), 36

`_write_ofx_file()` (biweeklybudget.ofxgetter.OfxGetter method), 51

A

`account` (biweeklybudget.models.account_balance.AccountBalance attribute), 36

`account` (biweeklybudget.models.ofx_statement.OFXStatement attribute), 37

`account` (biweeklybudget.models.ofx_transaction.OFXTransaction attribute), 38

`account` (biweeklybudget.models.scheduled_transaction.ScheduledTransaction attribute), 40

`account` (biweeklybudget.models.transaction.Transaction attribute), 41

`Account` (class in biweeklybudget.models.account), 34

`account_amount` (biweeklybudget.get.models.ofx_transaction.OFXTransaction attribute), 38

`account_id` (biweeklybudget.get.models.account_balance.AccountBalance attribute), 36

`account_id` (biweeklybudget.get.models.ofx_statement.OFXStatement attribute), 37

`account_id` (biweeklybudget.get.models.ofx_transaction.OFXTransaction attribute), 38

`account_id` (biweeklybudget.get.models.scheduled_transaction.ScheduledTransaction attribute), 40

`account_id` (biweeklybudget.get.models.transaction.Transaction attribute), 41

`AccountBalance` (class in biweeklybudget.get.models.account_balance), 36

`accounts()` (biweeklybudget.ofxgetter.OfxGetter static method), 51

`acct_type` (biweeklybudget.models.account.Account attribute), 34

`acct_type` (biweeklybudget.get.models.ofx_statement.OFXStatement attribute), 37

`acctid` (biweeklybudget.models.ofx_statement.OFXStatement attribute), 38

`AcctType` (class in biweeklybudget.models.account), 36

`actual_amount` (biweeklybudget.get.models.transaction.Transaction attribute), 41

`all_statements` (biweeklybudget.models.account.Account attribute), 34

`amount` (biweeklybudget.models.ofx_transaction.OFXTransaction attribute), 39

`amount` (biweeklybudget.models.scheduled_transaction.ScheduledTransaction attribute), 40

`as_dict` (biweeklybudget.models.account.AcctType attribute), 36

`as_dict` (biweeklybudget.models.base.ModelAsDict attribute), 37

`as_of` (biweeklybudget.models.ofx_statement.OFXStatement attribute), 38

`avail` (biweeklybudget.models.account_balance.AccountBalance attribute), 36

`avail_bal` (biweeklybudget.get.models.ofx_statement.OFXStatement attribute), 38

`avail_bal_as_of` (biweeklybudget.get.models.ofx_statement.OFXStatement attribute), 38

`avail_date` (biweeklybudget.get.models.account_balance.AccountBalance attribute), 36

B

`balance` (biweeklybudget.models.account.Account attribute), 34

`Bank` (biweeklybudget.models.account.AcctType attribute), 36

`bankid` (biweeklybudget.models.ofx_statement.OFXStatement attribute), 38

`biweeklybudget` (module), 31

`biweeklybudget.backfill_ofx` (module), 43

`biweeklybudget.biweeklypayperiod` (module), 43

`biweeklybudget.cliutils` (module), 48

`biweeklybudget.db` (module), 49

`biweeklybudget.db_event_handlers` (module), 50

`biweeklybudget.flaskapp` (module), 31

`biweeklybudget.flaskapp.cli_commands` (module), 32

`biweeklybudget.flaskapp.jsonencoder` (module), 33

`biweeklybudget.flaskapp.notifications` (module), 33

`biweeklybudget.initdb` (module), 50

`biweeklybudget.load_data` (module), 50

`biweeklybudget.models` (module), 34

`biweeklybudget.models.account` (module), 34

`biweeklybudget.models.account_balance` (module), 36

`biweeklybudget.models.base` (module), 37

- biweeklybudget.models.budget_model (module), 37
- biweeklybudget.models.ofx_statement (module), 37
- biweeklybudget.models.ofx_transaction (module), 38
- biweeklybudget.models.reconcile_rule (module), 40
- biweeklybudget.models.scheduled_transaction (module), 40
- biweeklybudget.models.transaction (module), 41
- biweeklybudget.models.txn_reconcile (module), 42
- biweeklybudget.ofxgetter (module), 51
- biweeklybudget.ofxupdater (module), 52
- biweeklybudget.screenscraper (module), 53
- biweeklybudget.settings (module), 54
- biweeklybudget.settings_example (module), 54
- biweeklybudget.utils (module), 55
- biweeklybudget.version (module), 55
- BiweeklyPayPeriod (class in biweeklybudget.models.biweeklypayperiod), 43
- brokerid (biweeklybudget.models.ofx_statement.OFXStatement attribute), 38
- budget (biweeklybudget.models.scheduled_transaction.ScheduledTransaction attribute), 40
- budget (biweeklybudget.models.transaction.Transaction attribute), 41
- Budget (class in biweeklybudget.models.budget_model), 37
- budget_account_sum() (biweeklybudget.flaskapp.notifications.NotificationsController static method), 33
- budget_id (biweeklybudget.models.scheduled_transaction.ScheduledTransaction attribute), 41
- budget_id (biweeklybudget.models.transaction.Transaction attribute), 42
- budget_sums (biweeklybudget.models.biweeklypayperiod.BiweeklyPayPeriod attribute), 47
- budgeted_amount (biweeklybudget.models.transaction.Transaction attribute), 42
- budgetModal() (built-in function), 56
- budgetModalDivFillAndShow() (built-in function), 56
- budgetModalDivForm() (built-in function), 56
- budgetModalDivHandleType() (built-in function), 56
- budgetTransferDivForm() (built-in function), 55
- budgetTransferModal() (built-in function), 55
- C**
- Cash (biweeklybudget.models.account.AcctType attribute), 36
- checknum (biweeklybudget.models.ofx_transaction.OFXTransaction attribute), 39
- clean_fitid() (built-in function), 58
- cleanup_db() (in module biweeklybudget.db), 49
- Credit (biweeklybudget.models.account.AcctType attribute), 36
- credit_limit (biweeklybudget.models.account.Account attribute), 34
- currency (biweeklybudget.models.ofx_statement.OFXStatement attribute), 38
- current_balance (biweeklybudget.models.budget_model.Budget attribute), 37
- D**
- date (biweeklybudget.models.scheduled_transaction.ScheduledTransaction attribute), 41
- date (biweeklybudget.models.transaction.Transaction attribute), 42
- date_posted (biweeklybudget.models.ofx_transaction.OFXTransaction attribute), 39
- date_suffix() (in module biweeklybudget.utils), 55
- day_of_month (biweeklybudget.models.scheduled_transaction.ScheduledTransaction attribute), 41
- DB_CONNSTRING (in module biweeklybudget.settings), 54
- DB_CONNSTRING (in module biweeklybudget.settings_example), 54
- db_session (in module biweeklybudget.db), 49
- default() (biweeklybudget.flaskapp.jsonencoder.MagicJSONEncoder method), 33
- DEFAULT_ACCOUNT_ID (in module biweeklybudget.settings), 54
- DEFAULT_ACCOUNT_ID (in module biweeklybudget.settings_example), 54
- description (biweeklybudget.models.account.Account attribute), 34
- description (biweeklybudget.models.budget_model.Budget attribute), 37
- description (biweeklybudget.models.ofx_transaction.OFXTransaction attribute), 39
- description (biweeklybudget.models.scheduled_transaction.ScheduledTransaction attribute), 41
- description (biweeklybudget.models.transaction.Transaction attribute), 42
- do_screenshot() (biweeklybudget.screenscraper.ScreenScraper method), 53

`doc_readystate_is_complete()` (biweeklybudget.get.screenscraper.ScreenScraper method), 53

`dtnow()` (in module biweeklybudget.utils), 55

`DuplicateFileException`, 52

E

`end_date` (biweeklybudget.get.biweeklypayperiod.BiweeklyPayPeriod attribute), 47

`engine` (in module biweeklybudget.db), 49

`error_screenshot()` (biweeklybudget.get.screenscraper.ScreenScraper method), 53

F

`file_mtime` (biweeklybudget.get.models.ofx_statement.OFXStatement attribute), 38

`filename` (biweeklybudget.get.models.ofx_statement.OFXStatement attribute), 38

`filter_query()` (biweeklybudget.get.biweeklypayperiod.BiweeklyPayPeriod method), 47

`fitid` (biweeklybudget.models.ofx_transaction.OFXTransaction attribute), 39

`fix_werkzeug_logger()` (in module biweeklybudget.utils), 55

`fmt_currency()` (built-in function), 56

`fmt_null()` (built-in function), 56

`for_ofxgetter` (biweeklybudget.models.account.Account attribute), 34

G

`get_browser()` (biweeklybudget.get.screenscraper.ScreenScraper method), 53

`get_notifications()` (biweeklybudget.flaskapp.notifications.NotificationsController static method), 33

`get_ofx()` (biweeklybudget.ofxgetter.OfxGetter method), 51

H

`handle_before_flush()` (in module biweeklybudget.db_event_handlers), 50

`handle_new_transaction()` (in module biweeklybudget.db_event_handlers), 50

`handle_trans_amount_change()` (in module biweeklybudget.db_event_handlers), 50

`handleForm()` (built-in function), 57

`handleFormError()` (built-in function), 57

`handleFormSubmitted()` (built-in function), 57

I

`id` (biweeklybudget.models.account.Account attribute), 34

`id` (biweeklybudget.models.account_balance.AccountBalance attribute), 36

`id` (biweeklybudget.models.budget_model.Budget attribute), 37

`id` (biweeklybudget.models.ofx_statement.OFXStatement attribute), 38

`id` (biweeklybudget.models.reconcile_rule.ReconcileRule attribute), 40

`id` (biweeklybudget.models.scheduled_transaction.ScheduledTransaction attribute), 41

`id` (biweeklybudget.models.transaction.Transaction attribute), 42

`id` (biweeklybudget.models.txn_reconcile.TxnReconcile attribute), 42

`in_directory()` (in module biweeklybudget.utils), 55

`init_db()` (in module biweeklybudget.db), 49

`init_event_listeners()` (in module biweeklybudget.db_event_handlers), 50

`Investment` (biweeklybudget.models.account.AcctType attribute), 36

`is_active` (biweeklybudget.models.account.Account attribute), 34

`is_active` (biweeklybudget.models.budget_model.Budget attribute), 37

`is_active` (biweeklybudget.get.models.reconcile_rule.ReconcileRule attribute), 40

`is_active` (biweeklybudget.get.models.scheduled_transaction.ScheduledTransaction attribute), 41

`is_budget_source` (biweeklybudget.get.models.account.Account attribute), 34

`is_income` (biweeklybudget.get.models.budget_model.Budget attribute), 37

`is_interest_charge` (biweeklybudget.get.models.ofx_transaction.OFXTransaction attribute), 39

`is_interest_payment` (biweeklybudget.get.models.ofx_transaction.OFXTransaction attribute), 39

`is_late_fee` (biweeklybudget.get.models.ofx_transaction.OFXTransaction attribute), 39

`is_other_fee` (biweeklybudget.get.models.ofx_transaction.OFXTransaction attribute), 39

`is_payment` (biweeklybudget.get.models.ofx_transaction.OFXTransaction attribute), 39

`is_periodic` (biweeklybud-

- get.models.budget_model.Budget attribute), 37
- is_stale (biweeklybudget.models.account.Account attribute), 34
- isFunction() (built-in function), 57
- isoformat() (built-in function), 56
- ## J
- jquery_finished() (biweeklybudget.screenscraper.ScreenScraper method), 53
- ## L
- ledger (biweeklybudget.models.account_balance.AccountBalance attribute), 36
- ledger_bal (biweeklybudget.models.ofx_statement.OFXStatement attribute), 38
- ledger_bal_as_of (biweeklybudget.models.ofx_statement.OFXStatement attribute), 38
- ledger_date (biweeklybudget.models.account_balance.AccountBalance attribute), 36
- load_cookies() (biweeklybudget.screenscraper.ScreenScraper method), 53
- ## M
- MagicJSONEncoder (class in biweeklybudget.flaskapp.jsonencoder), 33
- main() (in module biweeklybudget.backfill_ofx), 43
- main() (in module biweeklybudget.initdb), 50
- main() (in module biweeklybudget.load_data), 50
- main() (in module biweeklybudget.ofxgetter), 51
- makeTransFromOfx() (built-in function), 58
- makeTransSaveCallback() (built-in function), 58
- mcc (biweeklybudget.models.ofx_transaction.OFXTransaction attribute), 39
- memo (biweeklybudget.models.ofx_transaction.OFXTransaction attribute), 39
- ModelAsDict (class in biweeklybudget.models.base), 37
- ## N
- name (biweeklybudget.models.account.Account attribute), 34
- name (biweeklybudget.models.budget_model.Budget attribute), 37
- name (biweeklybudget.models.ofx_transaction.OFXTransaction attribute), 39
- name (biweeklybudget.models.reconcile_rule.ReconcileRule attribute), 40
- negate_ofx_amounts (biweeklybudget.models.account.Account attribute), 34
- next (biweeklybudget.biweeklypayperiod.BiweeklyPayPeriod attribute), 47
- note (biweeklybudget.models.txn_reconcile.TxnReconcile attribute), 42
- notes (biweeklybudget.models.ofx_transaction.OFXTransaction attribute), 39
- notes (biweeklybudget.models.scheduled_transaction.ScheduledTransaction attribute), 41
- notes (biweeklybudget.models.transaction.Transaction attribute), 42
- NotificationsController (class in biweeklybudget.flaskapp.notifications), 33
- num_per_period (biweeklybudget.models.scheduled_transaction.ScheduledTransaction attribute), 41
- num_stale_accounts() (biweeklybudget.flaskapp.notifications.NotificationsController static method), 33
- num_unreconciled_ofx() (biweeklybudget.flaskapp.notifications.NotificationsController static method), 33
- ## O
- ofx_account_id (biweeklybudget.models.txn_reconcile.TxnReconcile attribute), 42
- ofx_cat_memo_to_name (biweeklybudget.models.account.Account attribute), 35
- ofx_fitid (biweeklybudget.models.txn_reconcile.TxnReconcile attribute), 42
- ofx_statement (biweeklybudget.models.account.Account attribute), 35
- ofx_trans (biweeklybudget.models.txn_reconcile.TxnReconcile attribute), 42
- OfxBackfiller (class in biweeklybudget.backfill_ofx), 43
- OfxGetter (class in biweeklybudget.ofxgetter), 51
- ofxgetter_config (biweeklybudget.models.account.Account attribute), 35
- ofxgetter_config_json (biweeklybudget.models.account.Account attribute), 35
- OFXStatement (class in biweeklybudget.models.ofx_statement), 37
- OFXTransaction (class in biweeklybudget.models.ofx_transaction), 38
- ofxTransModal() (built-in function), 57
- OFXUpdater (class in biweeklybudget.ofxupdater), 52
- Other (biweeklybudget.models.account.AcctType attribute), 36
- overall_date (biweeklybudget.models.account_balance.AccountBalance attribute), 36

overall_sums (biweeklybudget.get.biweeklypayperiod.BiweeklyPayPeriod attribute), 47

P

params_from_ofxparser_transaction() (biweeklybudget.get.models.ofx_transaction.OFXTransaction static method), 39

parse_args() (in module biweeklybudget.backfill_ofx), 43

parse_args() (in module biweeklybudget.initdb), 50

parse_args() (in module biweeklybudget.load_data), 50

parse_args() (in module biweeklybudget.ofxgetter), 51

PAY_PERIOD_START_DATE (in module biweeklybudget.get.settings), 54

PAY_PERIOD_START_DATE (in module biweeklybudget.get.settings_example), 54

period_for_date() (biweeklybudget.get.biweeklypayperiod.BiweeklyPayPeriod static method), 47

period_interval (biweeklybudget.get.biweeklypayperiod.BiweeklyPayPeriod attribute), 48

period_length (biweeklybudget.get.biweeklypayperiod.BiweeklyPayPeriod attribute), 48

previous (biweeklybudget.get.biweeklypayperiod.BiweeklyPayPeriod attribute), 48

R

re_fee (biweeklybudget.models.account.Account attribute), 35

re_interest_charge (biweeklybudget.models.account.Account attribute), 35

re_interest_paid (biweeklybudget.models.account.Account attribute), 35

re_payment (biweeklybudget.models.account.Account attribute), 35

read() (biweeklybudget.utils.Vault method), 55

RECONCILE_BEGIN_DATE (in module biweeklybudget.get.settings), 54

RECONCILE_BEGIN_DATE (in module biweeklybudget.get.settings_example), 54

reconcile_id (biweeklybudget.get.models.ofx_transaction.OFXTransaction attribute), 39

reconcile_trans (biweeklybudget.get.models.account.Account attribute), 35

reconciled_at (biweeklybudget.get.models.txn_reconcile.TxnReconcile attribute), 42

reconcileDoUnreconcile() (built-in function), 59

reconcileDoUnreconcileNoOfx() (built-in function), 59

reconcileGetOFX() (built-in function), 59

reconcileGetTransactions() (built-in function), 59

reconcileHandleSubmit() (built-in function), 59

reconcileOfxDiv() (built-in function), 59

ReconcileRule (class in biweeklybudget.get.models.reconcile_rule), 40

reconcileShowOFX() (built-in function), 59

reconcileShowTransactions() (built-in function), 59

reconcileTransactions() (built-in function), 60

reconcileTransDiv() (built-in function), 60

reconcileTransDroppableAccept() (built-in function), 60

reconcileTransHandleDropEvent() (built-in function), 60

reconcileTransNoOfx() (built-in function), 60

recurrence_str (biweeklybudget.get.models.scheduled_transaction.ScheduledTransaction attribute), 41

routing_number (biweeklybudget.get.models.ofx_statement.OFXStatement attribute), 38

rule (biweeklybudget.models.txn_reconcile.TxnReconcile attribute), 43

rule_id (biweeklybudget.models.txn_reconcile.TxnReconcile attribute), 43

run() (biweeklybudget.backfill_ofx.OfxBackfiller method), 43

S

save_cookies() (biweeklybudget.get.screenscraper.ScreenScraper method), 53

schedModal() (built-in function), 61

schedModalDivFillAndShow() (built-in function), 61

schedModalDivForm() (built-in function), 61

schedModalDivHandleType() (built-in function), 61

schedToTransModal() (built-in function), 58

schedToTransModalDivFillAndShow() (built-in function), 58

schedToTransModalDivForm() (built-in function), 58

schedule_type (biweeklybudget.get.models.scheduled_transaction.ScheduledTransaction attribute), 41

scheduled_trans (biweeklybudget.get.models.transaction.Transaction attribute), 42

scheduled_trans_id (biweeklybudget.get.models.transaction.Transaction attribute), 42

ScheduledTransaction (class in biweeklybudget.get.models.scheduled_transaction), 40

ScreenScraper (class in biweeklybudget.screenscraper), 53

SecretMissingException, 55

serializeForm() (built-in function), 57

set_balance() (biweeklybudget.models.account.Account method), 35

- [set_log_debug\(\)](#) (in module `biweeklybudget.cliutils`), [48](#)
[set_log_info\(\)](#) (in module `biweeklybudget.cliutils`), [48](#)
[set_log_level_format\(\)](#) (in module `biweeklybudget.cliutils`), [49](#)
[set_ofxgetter_config\(\)](#) (`biweeklybudget.models.account.Account` method), [35](#)
[sic](#) (`biweeklybudget.models.ofx_transaction.OFXTransaction` attribute), [40](#)
[skipSchedTransModal\(\)](#) (built-in function), [58](#)
[skipSchedTransModalDivFillAndShow\(\)](#) (built-in function), [58](#)
[skipSchedTransModalDivForm\(\)](#) (built-in function), [58](#)
[STALE_DATA_TIMEDELTA](#) (in module `biweeklybudget.settings`), [54](#)
[STALE_DATA_TIMEDELTA](#) (in module `biweeklybudget.settings_example`), [54](#)
[standing_budgets_sum\(\)](#) (`biweeklybudget.flaskapp.notifications.NotificationsController` static method), [33](#)
[start_date](#) (`biweeklybudget.biweeklypayperiod.BiweeklyPayPeriod` attribute), [48](#)
[starting_balance](#) (`biweeklybudget.models.budget_model.Budget` attribute), [37](#)
[statement](#) (`biweeklybudget.models.ofx_transaction.OFXTransaction` attribute), [40](#)
[statement_id](#) (`biweeklybudget.models.ofx_transaction.OFXTransaction` attribute), [40](#)
[STATEMENTS_SAVE_PATH](#) (in module `biweeklybudget.settings`), [54](#)
[STATEMENTS_SAVE_PATH](#) (in module `biweeklybudget.settings_example`), [54](#)
- ## T
- [template_paths\(\)](#) (in module `biweeklybudget.flaskapp.cli_commands`), [32](#)
[TOKEN_PATH](#) (in module `biweeklybudget.settings`), [54](#)
[TOKEN_PATH](#) (in module `biweeklybudget.settings_example`), [54](#)
[trans_type](#) (`biweeklybudget.models.ofx_transaction.OFXTransaction` attribute), [40](#)
[transaction](#) (`biweeklybudget.models.txn_reconcile.TxnReconcile` attribute), [43](#)
[Transaction](#) (class in `biweeklybudget.models.transaction`), [41](#)
[transactions_list](#) (`biweeklybudget.biweeklypayperiod.BiweeklyPayPeriod` attribute), [48](#)
[transModal\(\)](#) (built-in function), [61](#)
[transModalDivFillAndShow\(\)](#) (built-in function), [62](#)
[transModalDivForm\(\)](#) (built-in function), [62](#)
[transModalOfxFillAndShow\(\)](#) (built-in function), [60](#)
[transNoOfx\(\)](#) (built-in function), [60](#)
[transNoOfxDivForm\(\)](#) (built-in function), [60](#)
[txn_id](#) (`biweeklybudget.models.txn_reconcile.TxnReconcile` attribute), [43](#)
[TxnReconcile](#) (class in `biweeklybudget.models.txn_reconcile`), [42](#)
[txnReconcileModal\(\)](#) (built-in function), [61](#)
[txnReconcileModalDiv\(\)](#) (built-in function), [61](#)
[type](#) (`biweeklybudget.models.ofx_statement.OFXStatement` attribute), [38](#)
- ## U
- [unreconciled](#) (`biweeklybudget.models.account.Account` attribute), [35](#)
[unreconciled\(\)](#) (`biweeklybudget.models.ofx_transaction.OFXTransaction` static method), [40](#)
[unreconciled\(\)](#) (`biweeklybudget.models.transaction.Transaction` static method), [42](#)
[unreconciled_sum](#) (`biweeklybudget.models.account.Account` attribute), [35](#)
[update\(\)](#) (`biweeklybudget.ofxupdater.OFXUpdater` method), [52](#)
[updateReconcileTrans\(\)](#) (built-in function), [60](#)
[upsert_record\(\)](#) (in module `biweeklybudget.db`), [49](#)
- ## V
- [validate_day_of_month\(\)](#) (`biweeklybudget.models.scheduled_transaction.ScheduledTransaction` method), [41](#)
[validate_num_per_period\(\)](#) (`biweeklybudget.models.scheduled_transaction.ScheduledTransaction` method), [41](#)
[Vault](#) (class in `biweeklybudget.utils`), [55](#)
[VAULT_ADDR](#) (in module `biweeklybudget.settings`), [54](#)
[VAULT_ADDR](#) (in module `biweeklybudget.settings_example`), [54](#)
[vault_creds_path](#) (`biweeklybudget.models.account.Account` attribute), [36](#)
- ## W
- [wait_for_ajax_load\(\)](#) (`biweeklybudget.screenscraper.ScreenScraper` method), [53](#)
- ## X
- [xhr_get_url\(\)](#) (`biweeklybudget.screenscraper.ScreenScraper` method), [53](#)

xhr_post_urlencoded() (biweeklybud-
get.screenscraper.ScreenScraper method),
[53](#)