

---

# **biweeklybudget Documentation**

*Release 1.1.1*

**Jason Antman**

**Dec 30, 2022**



---

## Contents

---

|          |                                       |            |
|----------|---------------------------------------|------------|
| <b>1</b> | <b>Overview</b>                       | <b>3</b>   |
| 1.1      | Important Warning . . . . .           | 3          |
| 1.2      | Main Features . . . . .               | 4          |
| <b>2</b> | <b>Requirements</b>                   | <b>5</b>   |
| <b>3</b> | <b>Installation</b>                   | <b>7</b>   |
| <b>4</b> | <b>License</b>                        | <b>9</b>   |
| <b>5</b> | <b>Attributions</b>                   | <b>11</b>  |
| <b>6</b> | <b>Contents</b>                       | <b>13</b>  |
| 6.1      | Screenshots . . . . .                 | 13         |
| 6.2      | Getting Started . . . . .             | 29         |
| 6.3      | Application Usage . . . . .           | 33         |
| 6.4      | Flask Application . . . . .           | 34         |
| 6.5      | Plaid . . . . .                       | 35         |
| 6.6      | OFX Transaction Downloading . . . . . | 38         |
| 6.7      | Getting Help . . . . .                | 42         |
| 6.8      | Development . . . . .                 | 42         |
| 6.9      | Changelog . . . . .                   | 46         |
| 6.10     | biweeklybudget . . . . .              | 54         |
| 6.11     | UI JavaScript Docs . . . . .          | 121        |
| <b>7</b> | <b>Indices and tables</b>             | <b>137</b> |
|          | <b>Python Module Index</b>            | <b>139</b> |
|          | <b>Index</b>                          | <b>141</b> |



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>



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 OFX-based automatic transaction download functionality (as opposed to Plaid), 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).

*Note:* Any potential users outside of the US should see the documentation section on [Currency Formatting and Localization](#); the short version is that I've done my best to make this configurable, but as far as I know I'm the only person using this software. If anyone else wants to use it and it doesn't work for your currency or locale, let me know and I'll fix it.

## 1.1 Important Warning

This software should be considered *beta* quality at best. I've been using it for about a year and it seems to be working correctly, but I'm very much a creature of habit; it's entirely possible that there are major bugs I haven't found because I always do the same action in the same way, the same order, the same steps, etc. In short, this application works for *me* and the *exact particular way I use it*, but it hasn't seen enough use from other people to say that it's stable

and correct in the general case. As such, please **DO NOT RELY ON** the mathematical/financial calculations without double-checking them.

## 1.2 Main Features

- Budgeting on a biweekly (fortnightly; every other week) basis, for those of us who are paid that way.
- Periodic (per-pay-period) or standing budgets.
- Optional automatic downloading of transactions/statements from your financial institutions via OFX Direct Connect, screen scraping, or [Plaid](#) and reconciling transactions (bank, credit, and investment accounts).
- Scheduled transactions - specific date or recurring (date-of-month, or number of times per pay period).
- Tracking of vehicle fuel fills (fuel log) and graphing of fuel economy.
- Cost tracking for multiple projects, including bills-of-materials for them. Optional synchronization from Amazon Wishlists to projects.
- Calculation of estimated credit card payoff amount and time, with configurable payment methods, payment increases on specific dates, and additional payments on specific dates.
- Ability to split a Transaction across multiple Budgets.

---

### 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 3.7+ (currently tested and developed with 3.10).
- Python [VirtualEnv](#) and `pip` (recommended installation method; your OS/distribution should have packages for these)
- 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 Plaid transaction downloading functionality, a valid [Plaid](#) account.
- To use the automated OFX Direct Connect transaction downloading functionality:
  - A running, reachable instance of [Hashicorp Vault](#) with your financial institution web credentials stored in it.
  - If your bank does not support OFX remote access (“Direct Connect”), you will need to write a custom screen-scraper class using Selenium and a browser.



## CHAPTER 3

---

### Installation

---

It's recommended that you run from the Docker image, as that's what I do. If you don't want to do that, you can also install in a virtualenv using Python 3.10:

```
mkdir biweeklybudget
python3.10 -mvenv venv
source venv/bin/activate
pip install biweeklybudget
```



## 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.

biweeklybudget includes a number of dependencies distributed alongside it, which are licensed and distributed under their respective licenses. See the `biweeklybudget/vendored` directory in the source distribution for further information.



## CHAPTER 5

---

### Attributions

---

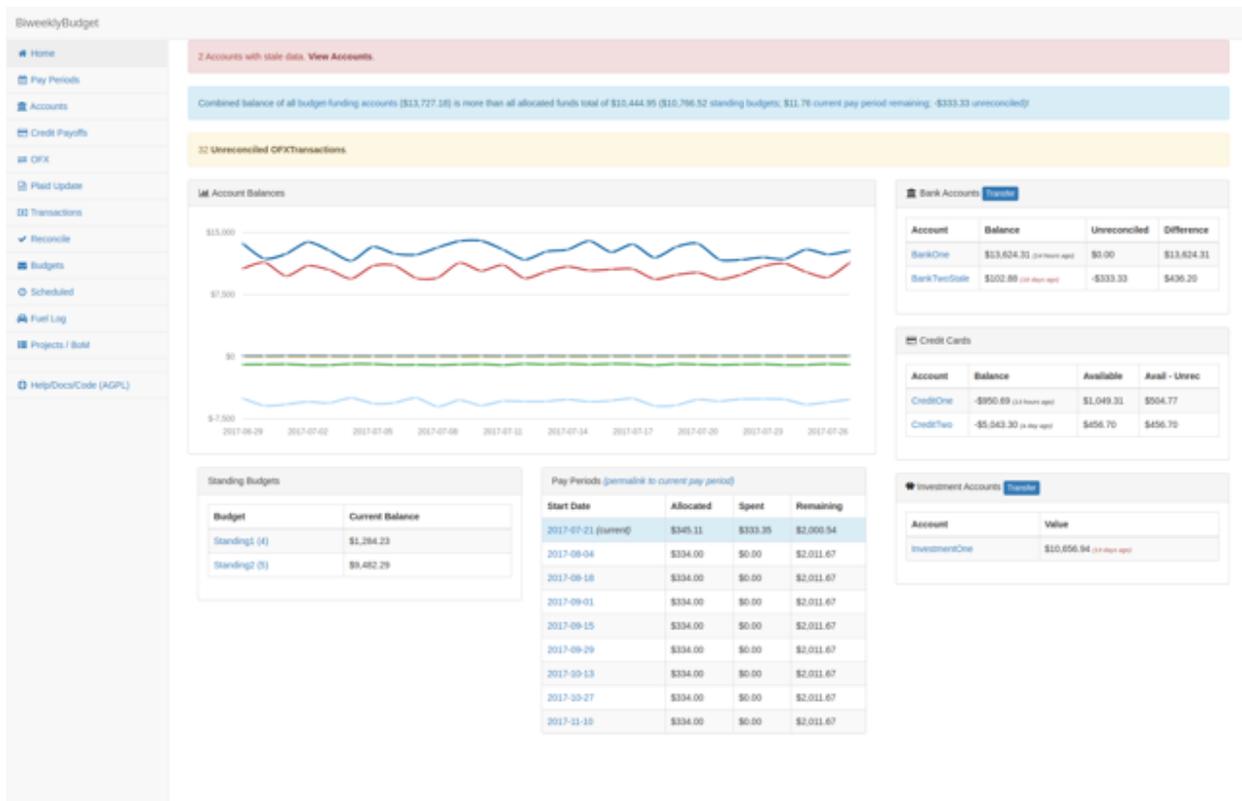
The logo used for biweeklybudget makes use of the wonderful, free Ledger icon by Eucalyp on FlatIcons: [Ledger icons created by Eucalyp - Flaticon](#).



### 6.1 Screenshots

#### 6.1.1 Index Page

Main landing page.



## 6.1.2 Transactions View

Shows all manually-entered transactions.

Transactions - BiweeklyBudget

Combined balance of all budget-funding accounts (\$13,727.18) is more than all allocated funds total of \$10,444.95 (\$10,796.52 standing budgets; \$11.76 current pay period remaining; \$333.33 unreconciled)

32 Unreconciled OPXTransactions.

Add Transaction

Show: 10 entries Budget: Account: Search:

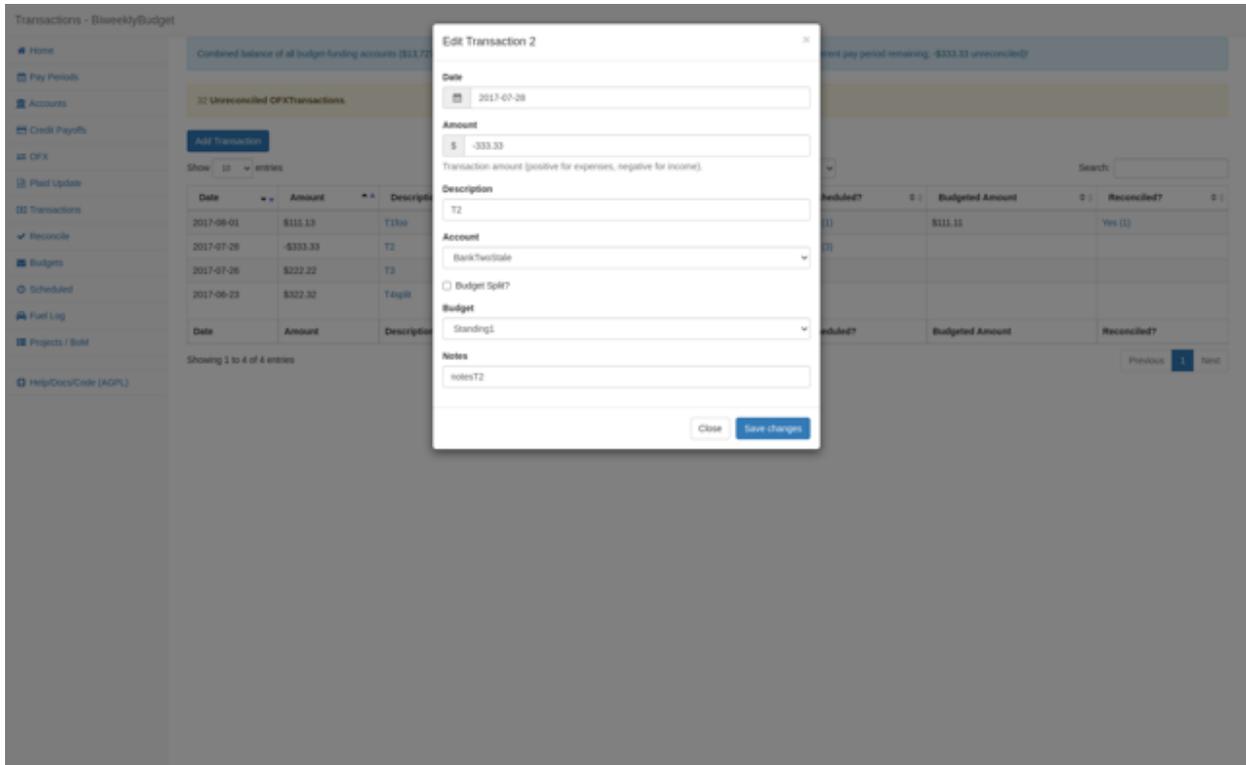
| Date       | Amount    | Description | Account          | Budget   | Scheduled? | Budgeted Amount | Reconciled? |
|------------|-----------|-------------|------------------|--|------------|-----------------|-------------|
| 2017-06-01 | \$111.53  | T3ba        | BankOne (1)      | Period1 (1)                                      | Yes (1)    | \$111.53        | Yes (1)     |
| 2017-07-28 | -\$333.33 | T2          | BankTwoState (2) | Standings (4)                                    | Yes (1)    |                 |             |
| 2017-07-26 | \$222.22  | T3          | CreditOne (3)    | Period2 (2)                                      |            |                 |             |
| 2017-06-23 | \$322.32  | T4qilt      | CreditOne (3)    | Period2 (2) (\$222.22)<br>Period1 (1) (\$100.10) |            |                 |             |
| Date       | Amount    | Description | Account          | Budget   | Scheduled? | Budgeted Amount | Reconciled? |

Showing 1 to 4 of 4 entries

Previous Next

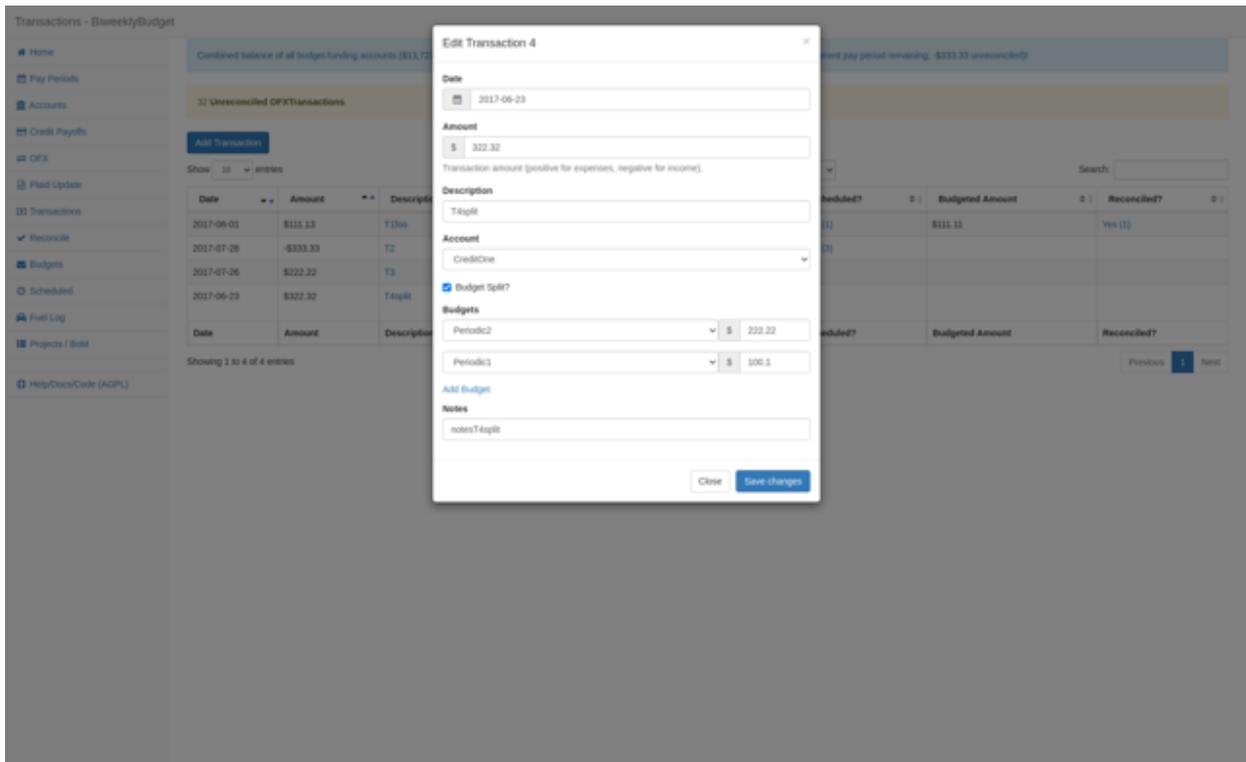
## 6.1.3 Transaction Detail

Transaction detail modal to view and edit a transaction.



## 6.1.4 Transactions with Budget Splits

A single Transaction can be split across multiple budgets.



## 6.1.5 Credit Card Payoff Calculations

Credit card payoff calculations based on a variety of payment methods, with configurable payment increases over time or one-time additional payment amounts.

Credit Card Payoffs - BiweeklyBudget

Combined balance of all budget holding accounts (\$61,707.00) is more than all allocated funds total of \$50,444.00 (\$61,700.00 holding budgets, \$12,734 current pay period remaining, \$365.00 unallocated)

Unallocated OFX Transactions

Notice - These calculations are rough estimates only. Do not rely on them.

- These are based on the interest rate you entered in the Account settings. Interest rates will change over time.
- You have chosen to be within 5% of my own statements for certain credit cards, but there is no guarantee that the exact formula used will match those used by your credit card company.
- These assume no late or punitive charges apply at the accounts, i.e. the only balance changes will be one payment per billing period and interest for that period.
- The interest calculation method may not exactly match your financial institution.
- Tables are tabular so we can calculate multiple payments together, we use the last balance for each account as if it were the beginning of a new billing cycle, and assume that all billing cycles are based on calendar months. All payments are made half way through the month.
- Some tables are rounded.

Settings

Start-of-Month Monthly Payment(s) \$ 125.00

Starting on [ ] increase each of monthly payments to \$ [ ] [\(edit\)](#)

[\(add another increase\)](#)

On the first payment date on or after [ ] add \$ [ ] to the payment amount. [\(edit\)](#)

[\(add another one-time additional payment\)](#)

Next 6 Transactions

MinPaymentMethod - Minimum Payment Only  
Pay only the minimum on each statement.

| Account                             | Next Payment    | Time To Pay Off   | Total Payments    | Total Interest    |
|-------------------------------------|-----------------|-------------------|-------------------|-------------------|
| Creditive (C) (\$500.00 @ 1.00%)    | \$25.00         | 2.7 years         | \$693.45          | \$13.50           |
| CreditFee (F) (\$5,000.00 @ 10.00%) | \$208.07        | 32.8 years        | \$7,288.73        | \$2,045.43        |
| <b>Totals</b>                       | <b>\$233.07</b> | <b>32.8 years</b> | <b>\$7,982.18</b> | <b>\$2,058.93</b> |

LowestInterestRateFirstMethod - Lowest to Highest Interest Rate  
Pay statements of those based to highest interest rate.

| Account                             | Next Payment    | Time To Pay Off  | Total Payments    | Total Interest    |
|-------------------------------------|-----------------|------------------|-------------------|-------------------|
| Creditive (C) (\$500.00 @ 1.00%)    | \$25.00         | 2.7 years        | \$693.45          | \$13.50           |
| CreditFee (F) (\$5,000.00 @ 10.00%) | \$208.07        | 4.8 years        | \$8,385.65        | \$1,952.25        |
| <b>Totals</b>                       | <b>\$233.07</b> | <b>4.8 years</b> | <b>\$9,079.10</b> | <b>\$1,965.75</b> |

LowestBalanceFirstMethod - Lowest to Highest Balance (i.e. a "Snowball" method)  
Pay statements of those based to highest balance, i.e. the "snowball" method.

| Account                             | Next Payment    | Time To Pay Off  | Total Payments    | Total Interest    |
|-------------------------------------|-----------------|------------------|-------------------|-------------------|
| Creditive (C) (\$500.00 @ 1.00%)    | \$25.00         | 2.7 years        | \$693.45          | \$13.50           |
| CreditFee (F) (\$5,000.00 @ 10.00%) | \$208.07        | 4.8 years        | \$8,385.65        | \$1,952.25        |
| <b>Totals</b>                       | <b>\$233.07</b> | <b>4.8 years</b> | <b>\$9,079.10</b> | <b>\$1,965.75</b> |

HighestInterestRateFirstMethod - Highest to Lowest Interest Rate  
Pay statements of those highest to lowest interest rate.

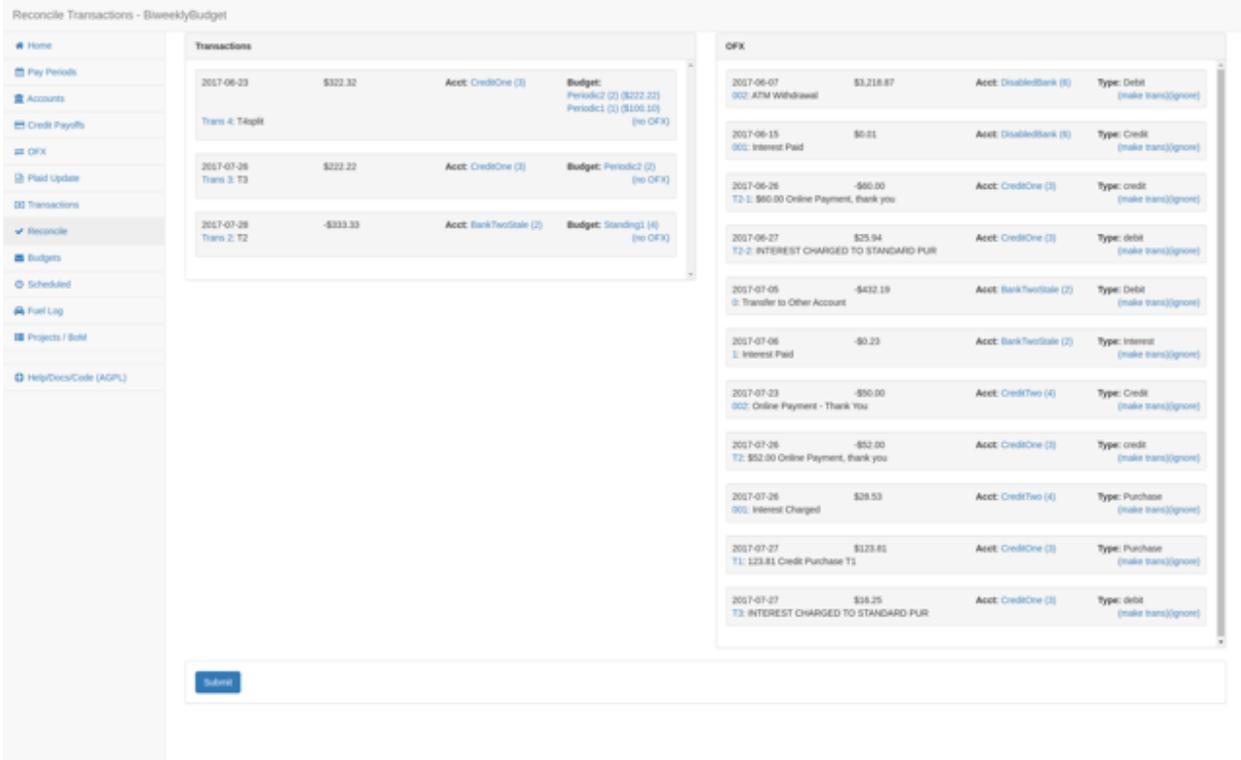
| Account                             | Next Payment    | Time To Pay Off  | Total Payments    | Total Interest    |
|-------------------------------------|-----------------|------------------|-------------------|-------------------|
| Creditive (C) (\$500.00 @ 1.00%)    | \$25.00         | 2.7 years        | \$693.45          | \$13.50           |
| CreditFee (F) (\$5,000.00 @ 10.00%) | \$208.07        | 4.8 years        | \$8,372.43        | \$1,928.75        |
| <b>Totals</b>                       | <b>\$233.07</b> | <b>4.8 years</b> | <b>\$9,065.88</b> | <b>\$1,942.25</b> |

HighestBalanceFirstMethod - Highest to Lowest Balance  
Pay statements of those highest to lowest balance.

| Account                             | Next Payment    | Time To Pay Off  | Total Payments    | Total Interest    |
|-------------------------------------|-----------------|------------------|-------------------|-------------------|
| Creditive (C) (\$500.00 @ 1.00%)    | \$25.00         | 2.7 years        | \$693.45          | \$13.50           |
| CreditFee (F) (\$5,000.00 @ 10.00%) | \$208.07        | 4.8 years        | \$8,372.43        | \$1,928.75        |
| <b>Totals</b>                       | <b>\$233.07</b> | <b>4.8 years</b> | <b>\$9,065.88</b> | <b>\$1,942.25</b> |

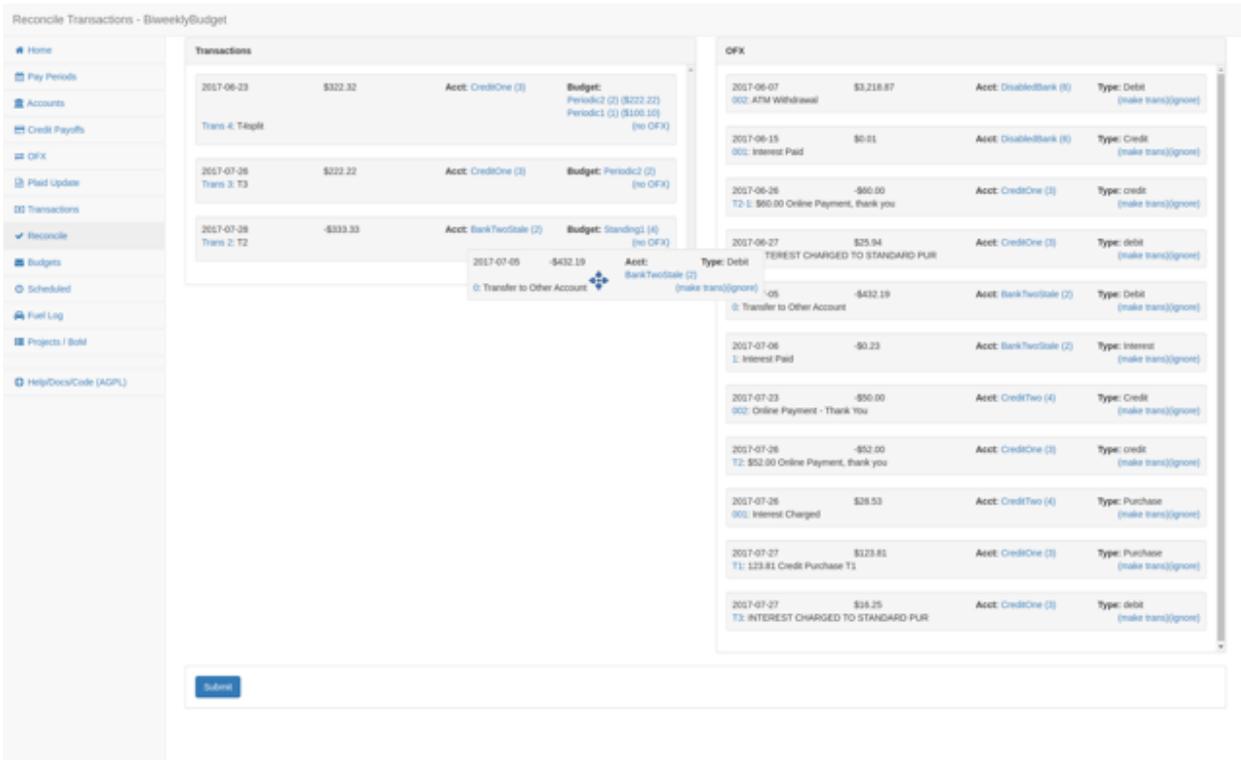
## 6.1.6 Reconcile Transactions with OFX

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



### 6.1.7 Drag-and-Drop Reconciling

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



## 6.1.8 Pay Periods View

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

Pay Periods - BiweeklyBudget

Combined balance of all budget-funding accounts (\$33,727.18) is less than all allocated funds total of \$33,742.88 (\$33,768.52 standing budgets, \$0.02 current pay period remaining, \$2,876.38 unreconciled)

**\$498.47**  
Remaining this period

View 2017-07-23 Pay Period

**\$541.50**  
Remaining next period

View 2017-08-04 Pay Period

**\$351.44**  
Remaining following period

View 2017-08-18 Pay Period

| Start Date           | Allocated  | Spent      | Remaining |
|----------------------|------------|------------|-----------|
| 2017-07-07           | \$1,795.85 | \$1,795.85 | \$049.82  |
| 2017-07-21 (current) | \$1,847.18 | \$1,847.20 | \$498.47  |
| 2017-08-04           | \$1,804.17 | \$1,981.95 | \$541.50  |
| 2017-08-18           | \$1,994.23 | \$1,994.23 | \$351.44  |
| 2017-09-01           | \$1,871.26 | \$1,849.04 | \$474.41  |
| 2017-09-15           | \$2,214.24 | \$2,126.88 | \$131.43  |
| 2017-09-29           | \$2,095.70 | \$1,733.48 | \$289.97  |
| 2017-10-13           | \$2,338.85 | \$2,338.85 | \$6.82    |
| 2017-10-27           | \$2,285.16 | \$2,062.94 | \$460.51  |
| 2017-11-10           | \$1,932.82 | \$1,932.82 | \$412.85  |

Find Pay Period

Date:

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

## 6.1.9 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).

2017-07-21 to 2017-06-03 Pay Period - BiweeklyBudget

Combined balance of all budget funding accounts (\$13,727.18) is less than all allocated funds total of \$13,742.88 (\$13,706.52 standing budgets, \$0.02 current pay period remaining, \$2.9% unrecconciled)

| Remaining Balances |  | 2017-07-21 (curr.) | 2017-06-04 (next) | 2017-06-18 | 2017-06-01 |
|--------------------|--|--------------------|-------------------|------------|------------|
| 2017-07-07 (prev.) |  | \$549.82           | \$498.47          | \$541.50   | \$351.44   |

**\$2,345.67**

Income

**\$1,847.18**

allocated

**\$1,847.20**

spent

**\$498.47**

remaining

**Periodic Budgets - Transfer**

| Budget    | Amount     | Allocated | Spent    | Remaining           |
|-----------|------------|-----------|----------|---------------------|
| Periodic1 | \$300.00   | \$172.33  | \$172.35 | <del>\$127.65</del> |
| Periodic2 | \$234.00   | \$862.85  | \$862.85 | <del>\$628.85</del> |
| Income () | \$2,345.67 | \$0.00    | \$0.00   | \$2,345.67          |
| Budget3   | \$0.00     | \$221.22  | \$221.22 | <del>\$221.22</del> |
| Budget    | \$0.00     | \$590.78  | \$590.78 | <del>\$590.78</del> |

**Standing Budgets - Transfer**

| Budget    | Balance    |
|-----------|------------|
| Standing1 | \$1,204.23 |
| Standing2 | \$8,482.29 |

**Transactions - Add Transaction**

| Date       | Amount    | Description           | Account      | Budget    | Scheduled? | Reconciled? |
|------------|-----------|-----------------------|--------------|-----------|------------|-------------|
| 2017-07-22 | \$6.55    | Transaction 1.1 (34)  | BankOne      | Budget4   |            |             |
| 2017-07-22 | \$32.30   | Transaction 1.7 (22)  | BankOne      | Periodic2 |            |             |
| 2017-07-22 | \$58.20   | Transaction 1.4 (29)  | BankOne      | Budget3   |            |             |
| 2017-07-22 | \$61.22   | Transaction 1.6 (31)  | BankOne      | Periodic1 |            |             |
| 2017-07-22 | \$146.42  | Transaction 1.3 (38)  | BankOne      | Budget5   |            |             |
| 2017-07-22 | \$153.13  | Transaction 1.9 (24)  | BankOne      | Periodic2 |            |             |
| 2017-07-22 | \$163.02  | Transaction 1.11 (24) | BankOne      | Budget3   |            |             |
| 2017-07-22 | \$202.29  | Transaction 1.2 (17)  | BankOne      | Budget4   |            |             |
| 2017-07-22 | \$203.67  | Transaction 1.8 (23)  | BankOne      | Periodic2 |            |             |
| 2017-07-22 | \$235.52  | Transaction 1.10 (25) | BankOne      | Budget4   |            |             |
| 2017-07-22 | \$251.53  | Transaction 1.5 (20)  | BankOne      | Periodic2 |            |             |
| 2017-07-26 | \$222.22  | T3 (3)                | CreditOne    | Periodic2 |            |             |
| 2017-07-26 | -\$333.33 | T2 (2)                | BankTwoState | Standing1 | (from 3)   |             |
| 2017-06-01 | \$111.13  | T10a (2)              | BankOne      | Periodic1 | (from 2)   | Yes (1)     |

## 6.1.10 Budgets

List all budgets, along with graphs of spending per budget, per payperiod and per month.

Budgets - BiweeklyBudget

- Home
- Pay Periods
- Accounts
- Credit Payoffs
- OFX
- Paid Update
- Transactions
- Reconcile
- Budgets**
- Scheduled
- Fuel Log
- Projects / Bolt
- Help/Docs/Code (AGPL)

Combined balance of all budget funding accounts (\$13,727.18) is less than all allocated funds total of \$30,194.95 (\$16,766.52 standing budgets, \$261.76 current pay period remaining, \$13,166.67 unreconciled)

Spending By Budget, Per Pay Period

Spending By Budget, Per Calendar Month

[Add Budget](#) [Transfer](#)

Periodic Budgets

| Active? | Budget                 | Starting Balance |
|---------|------------------------|------------------|
| yes     | Budget3 (10)           | \$0.00           |
| yes     | Budget4 (11)           | \$0.00           |
| yes     | Budget5 (12)           | \$250.00         |
| yes     | Income (7) (income)    | \$2,345.67       |
| yes     | Periodic1 (1)          | \$100.00         |
| yes     | Periodic2 (2)          | \$234.00         |
| NO      | Periodic3 inactive (3) | \$19.23          |

Standing Budgets

| Active? | Budget                 | Current Balance |
|---------|------------------------|-----------------|
| yes     | Standing1 (4)          | \$1,284.23      |
| yes     | Standing2 (5)          | \$8,482.29      |
| NO      | Standing3 inactive (6) | -\$82.29        |

### 6.1.11 Single Budget View

Budget detail modal to view and edit a budget.

The screenshot displays the 'biweeklybudget' application interface. A central dialog box titled 'Edit Budget 2' is open, allowing for the configuration of a budget. The dialog includes fields for Name, Description, Starting Balance, and checkboxes for Active, Income, and Omit from graphs. The background shows a dashboard with a sidebar menu, a top status bar, and two line charts. Below the charts are two tables: 'Periodic Budgets' and 'Standing Budgets'.

**Periodic Budgets Table:**

| Active? | Budget                 | Starting Balance |
|---------|------------------------|------------------|
| yes     | Budget3 (10)           | \$0.00           |
| yes     | Budget4 (11)           | \$0.00           |
| yes     | Budget5 (12)           | \$250.00         |
| yes     | Income (7) (Income)    | \$2,345.67       |
| yes     | Periodic1 (1)          | \$100.00         |
| yes     | Periodic2 (2)          | \$234.00         |
| NO      | Periodic3 inactive (3) | \$10.29          |

**Standing Budgets Table:**

| Active? | Budget                 | Current Balance |
|---------|------------------------|-----------------|
| yes     | Standing1 (4)          | \$1,294.23      |
| yes     | Standing2 (5)          | \$8,462.29      |
| NO      | Standing3 inactive (6) | -\$82.29        |

## 6.1.12 Accounts View

Accounts - BiweeklyBudget

- Home
- Pay Periods
- Accounts
- Credit Payoffs
- OFX
- Filed Update
- Transactions
- Reconcile
- Budgets
- Scheduled
- Fuel Log
- Projects / Bolt
- Help/Docs/Code (AGPL)

Combined balance of all budget-funding accounts (\$13,727.18) is more than all allocated funds total of \$10,444.95 (\$10,796.52 standing budgets, \$11.76 current pay period remaining, -\$333.33 unreconciled)

**Bank Accounts** [Add Account](#) [Transfer](#)

| Account      | Balance                                    | Unreconciled | Difference  |
|--------------|--|--------------|-------------|
| BankOne      | \$13,824.31 <small>(not from appt)</small> | \$0.00       | \$13,824.31 |
| BankTwoState | \$302.89 <small>(over)</small>             | -\$333.33    | \$436.20    |

**Credit Cards** [Add Account](#)

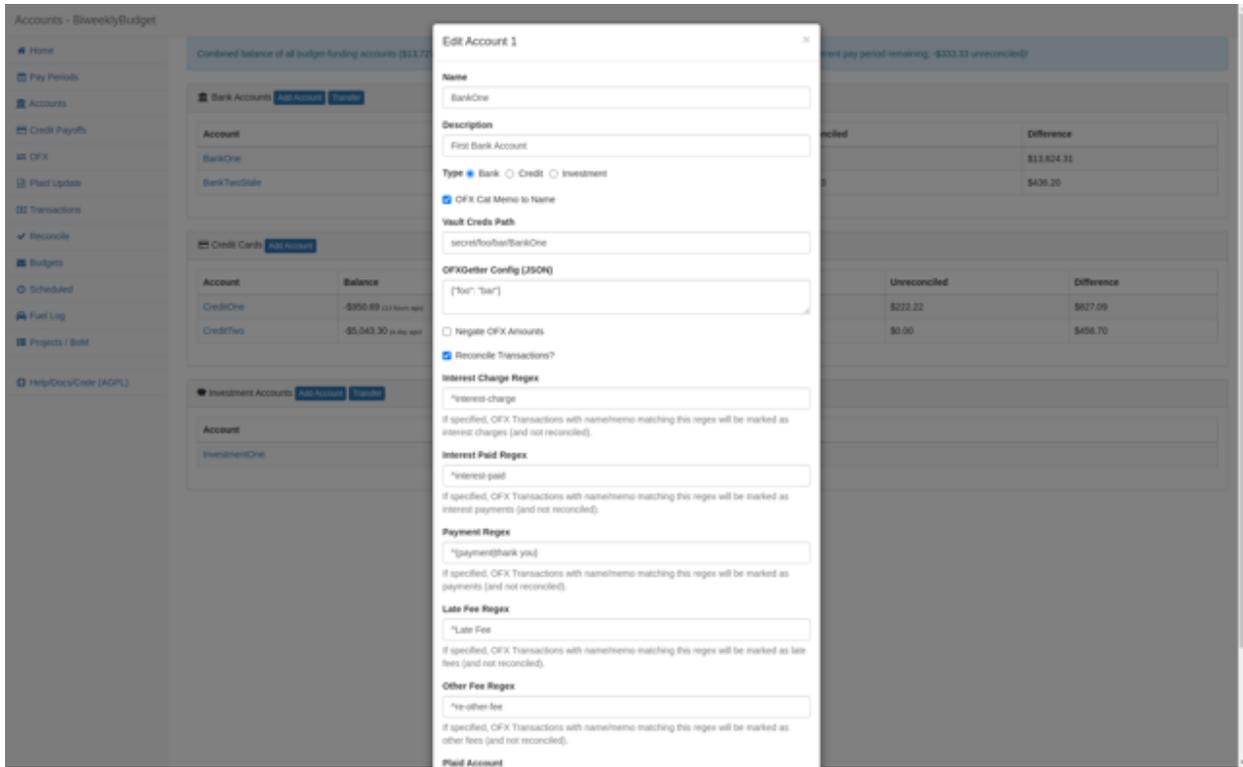
| Account   | Balance                                  | Credit Limit | Available  | Unreconciled | Difference |
|-----------|--|--------------|------------|--------------|------------|
| CreditOne | -\$950.89 <small>(not from appt)</small> | \$2,000.00   | \$1,049.31 | \$222.22     | \$827.09   |
| CreditTwo | -\$5,043.30 <small>(in the appt)</small> | \$5,500.00   | \$456.70   | \$0.00       | \$456.70   |

**Investment Accounts** [Add Account](#) [Transfer](#)

| Account       | Value                             |
|---------------|-----------------------------------|
| InvestmentOne | \$10,856.94 <small>(over)</small> |

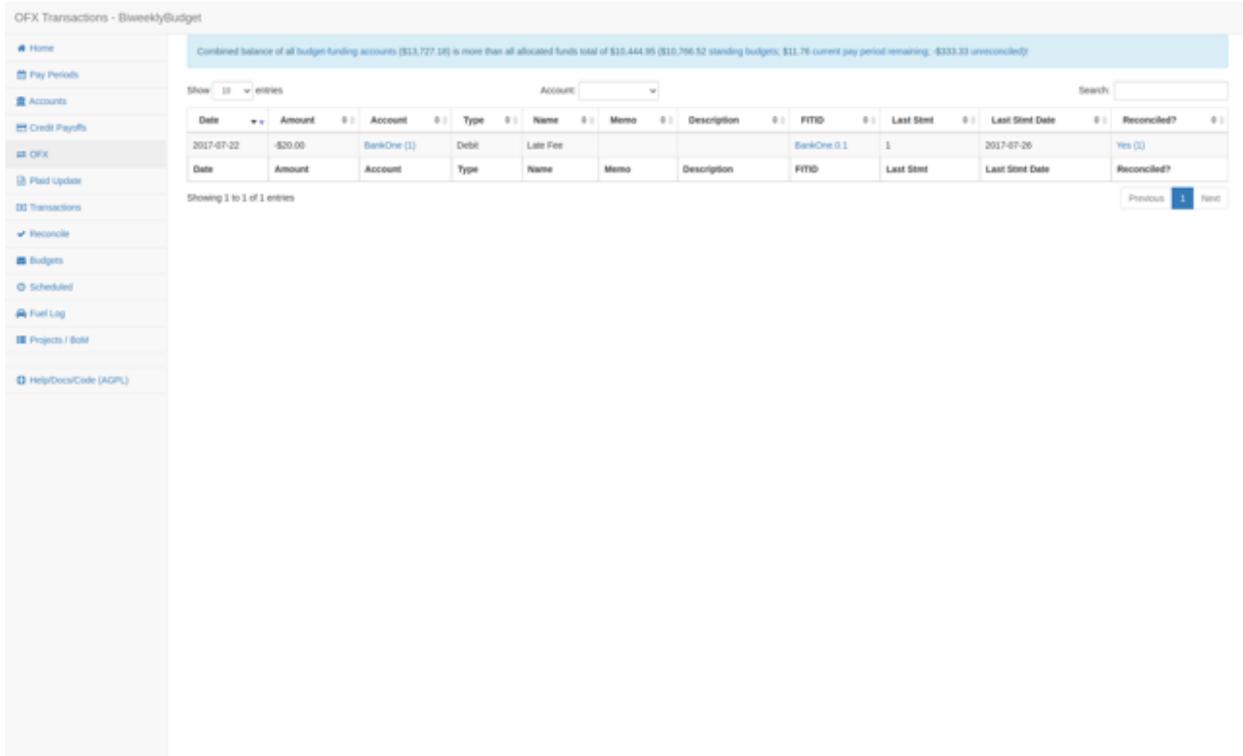
## 6.1.13 Account Details

Details of a single account.



## 6.1.14 OFX Transactions

Shows transactions imported from OFX statements.



## 6.1.15 Scheduled Transactions

List all scheduled transactions (active and inactive).

Scheduled Transactions - BiweeklyBudget

Combined balance of all budget-funding accounts (\$13,727.18) is more than all allocated funds total of \$10,444.95 (\$10,796.52 standing budgets; \$11.76 current pay period remaining; \$333.33 unreconciled)

[Add Scheduled Transaction](#)

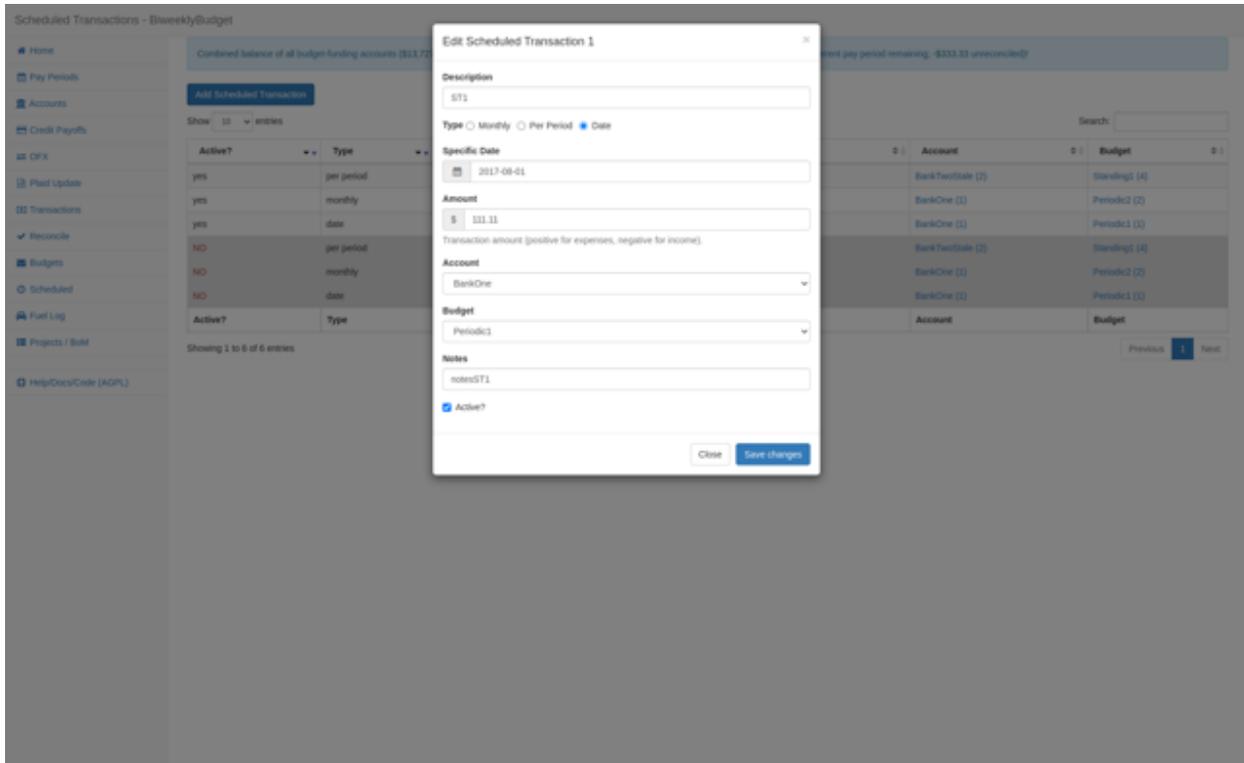
Show: 10 entries      Type:       Search:

| Active? | Type       | Recurrence   | Amount    | Description | Account          | Budget        |
|---------|------------|--------------|-----------|-------------|------------------|---------------|
| yes     | per period | 1 per period | -\$333.33 | ST3         | BankTwoState (2) | Standing1 (4) |
| yes     | monthly    | 4th          | \$222.22  | ST2         | BankOne (1)      | Periodic2 (2) |
| yes     | date       | 2017-08-01   | \$115.15  | ST1         | BankOne (1)      | Periodic1 (3) |
| NO      | per period | 3 per period | \$666.66  | ST6         | BankTwoState (2) | Standing1 (4) |
| NO      | monthly    | 5th          | \$555.55  | ST5         | BankOne (1)      | Periodic2 (2) |
| NO      | date       | 2017-09-02   | \$444.44  | ST4         | BankOne (1)      | Periodic1 (3) |

Showing 1 to 6 of 6 entries      [Previous](#) [Next](#)

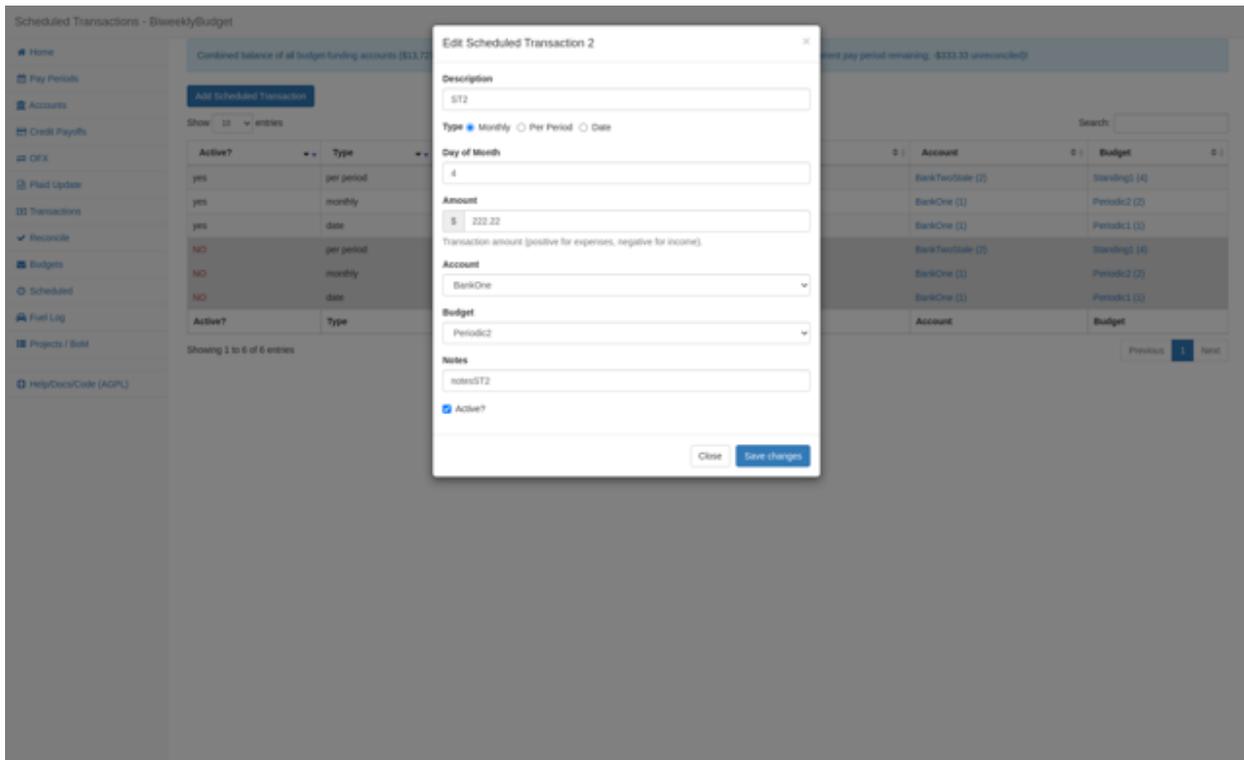
## 6.1.16 Specific Date Scheduled Transaction

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



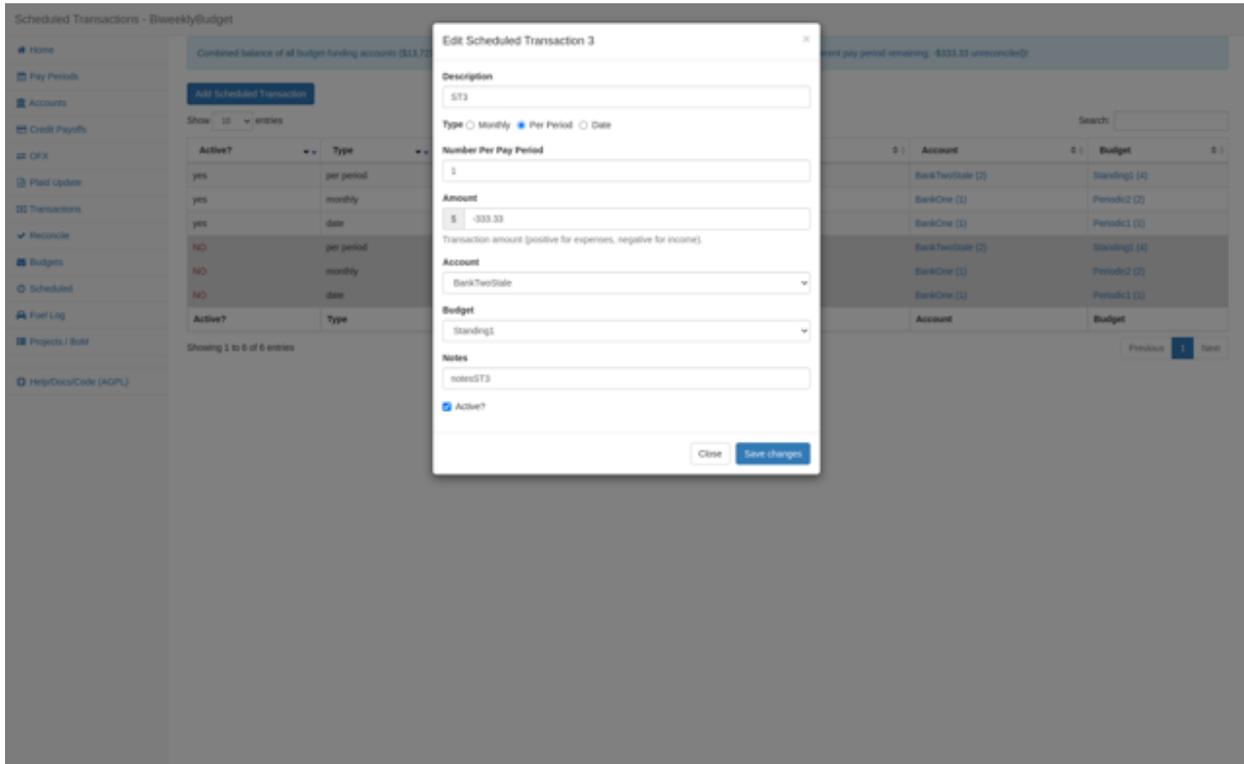
## 6.1.17 Monthly Scheduled Transaction

Scheduled transactions can occur monthly on a given date.



## 6.1.18 Number Per-Period Scheduled Transactions

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



## 6.1.19 Fuel Log

Vehicle fuel log and fuel economy tracking.

Fuel Log - BiweeklyBudget

- Home
- Pay Periods
- Accounts
- Credit Payments
- OFX
- Feed Update
- Transactions
- Records
- Budgets
- Scheduled
- Fuel Log
- Projects / Bulk
- Help/Doc/Code (NDPL)

Combined balance of all budget funding accounts (\$13,727.28) is more than all allocated funds total of \$13,444.95 (\$28,796.52 standing budgets, \$11.76 current pay period remaining, \$633.33 unrecorded)

Fuel Fills [Add Entry](#)

Show 12 entries Vehicle:  Search:

| Date       | Vehicle | Odometer Mi. | Mi. Traveled (from veh.) | Mi. Traveled (adj.) | Start Fuel Level | End Fuel Level | Location | \$/Gal. | Total Cost | Total Gallons | MPG (from veh.) | MPG (adj.) | Notes |
|------------|---------|--------------|--------------------------|---------------------|------------------|----------------|----------|---------|------------|---------------|-----------------|------------|-------|
| 2017-08-29 | WRX (2) | 7,624        | 7624                     | 250                 | 10%              | 100%           | fee      | \$2.87  | \$2.71     | 7.625         |                 | 21.399     |       |
| 2017-08-29 | WRX (2) | 7,065        | 7065                     | 244                 | 30%              | 100%           | fee      | \$2.06  | \$7.77     | 7.330         |                 | 22.373     |       |
| 2017-08-28 | WRX (2) | 6,874        | 6874                     | 236                 | 10%              | 100%           | fee      | \$2.29  | \$2.80     | 6.900         |                 | 26.473     |       |
| 2017-08-29 | WRX (2) | 6,901        | 6901                     | 179                 | 0%               | 100%           | fee      | \$2.75  | \$29.40    | 10.690        |                 | 18.744     |       |
| 2017-08-27 | WRX (2) | 6,636        | 6636                     | 246                 | 10%              | 100%           | fee      | \$2.74  | \$29.48    | 7.110         |                 | 34.88      |       |
| 2017-08-27 | WRX (2) | 6,722        | 4722                     | 236                 | 10%              | 100%           | fee      | \$2.70  | \$14.13    | 6.320         |                 | 19.923     |       |
| 2017-08-26 | WRX (2) | 6,388        | 6388                     | 287                 | 10%              | 100%           | fee      | \$2.25  | \$22.72    | 11.660        |                 | 17.822     |       |
| 2017-08-26 | WRX (2) | 6,964        | 6964                     | 215                 | 20%              | 100%           | fee      | \$2.89  | \$32.80    | 11.350        |                 | 18.942     |       |
| 2017-08-26 | WRX (2) | 6,281        | 4281                     | 236                 | 10%              | 100%           | fee      | \$2.28  | \$26.11    | 7.200         |                 | 21.109     |       |
| 2017-08-25 | WRX (2) | 6,349        | 6349                     | 177                 | 30%              | 100%           | fee      | \$2.25  | \$21.90    | 10.660        |                 | 17.7       |       |

Showing 1 to 12 of 64 entries Previous 1 2 3 4 5 6 7 Next

Vehicles [Add Vehicle](#)

| ID | Name         | Active? |
|----|--------------|---------|
| 1  | WRX1         | True    |
| 2  | WRX2         | True    |
| 3  | WRX3Inactive | False   |

## 6.1.20 Project Tracking

Track projects and their cost.

Projects / Bill of Materials - BiweeklyBudget

Combined balance of all budget funding accounts (\$13,727.18) is more than all allocated funds total of \$10,444.95 (\$10,796.52 standing budgets, \$11.76 current pay period remaining, -\$333.33 unreconciled)

**\$77.77**  
Remaining Cost - Active Projects

**\$2,546.89**  
Total Cost - Active Projects

Projects / Bill of Materials

Show 10 entries Search

| Project        | Total Cost        | Remaining Cost        | Active?        | Notes                |
|----------------|-------------------|-----------------------|----------------|----------------------|
| P1             | \$2,546.89        | \$77.77               | yes (inactive) | ProjectOne           |
| P2             | \$0.00            | \$0.00                | yes (inactive) | ProjectTwo           |
| P3inactive     | \$5.34            | \$3.00                | NO (active)    | ProjectThreeinactive |
| <b>Project</b> | <b>Total Cost</b> | <b>Remaining Cost</b> | <b>Active?</b> | <b>Notes</b>         |

Showing 1 to 3 of 3 entries Previous 1 Next

Name  Notes  Add Project

## 6.1.21 Projects - Bill of Materials

Track individual items/materials for projects.

Project 1 - P1 - BiweeklyBudget

Combined balance of all budget funding accounts (\$13,727.18) is more than all allocated funds total of \$10,444.95 (\$10,796.52 standing budgets, \$11.76 current pay period remaining, -\$333.33 unreconciled)

**\$77.77**  
Remaining

**\$2,546.89**  
Total

Notes  
 ProjectOne

Show 10 entries Search

| Name          | Quantity        | Unit Cost        | Line Cost        | Notes        | Active?        |
|---------------|-----------------|------------------|------------------|--------------|----------------|
| P1Item1 (ind) | 1               | \$11.11          | \$11.11          | P1Item1Notes | yes            |
| P1Item2 (ind) | 3               | \$22.22          | \$66.66          | P1Item2Notes | yes            |
| P1Item3 (ind) | 2               | \$1,234.56       | \$2,469.12       | P1Item3Notes | NO             |
| <b>Name</b>   | <b>Quantity</b> | <b>Unit Cost</b> | <b>Line Cost</b> | <b>Notes</b> | <b>Active?</b> |

Showing 1 to 3 of 3 entries Previous 1 Next

## 6.2 Getting Started

### 6.2.1 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 3.6+ (currently tested with 3.6, 3.7, 3.8 and developed with 3.8). **Python 2 is not supported.**
- Python [VirtualEnv](#) and `pip` (recommended installation method; your OS/distribution should have packages for these)
- 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 new *Plaid* automated transaction downloading functionality, a valid Plaid account.
- To use the (old) automated OFX transaction downloading functionality:
  - A running, reachable instance of [Hashicorp Vault](#) with your financial institution web credentials stored in it.
  - If your bank does not support OFX remote access (“Direct Connect”), you will need to write a custom screen-scraper class using Selenium and a browser.

### 6.2.2 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 3.8. It does not support Python3 < 3.4.

```
mkdir biweeklybudget
virtualenv --python=python3.8 .
source bin/activate
pip install biweeklybudget
```

**Important Note:** Anyone who’s using this project for actual data should install from the package on PyPI. While the `master` branch of the git repository is always in a runnable state, there is no guarantee that data will be not be harmed by upgrading directly to `master`. Specifically, database migrations are only compatible between released versions; `master` is considered a pre-release/development version, and can have migrations removed or altered in breaking ways between official releases.

### 6.2.3 Upgrading

Documentation for upgrades depends on how you’ve installed and run biweeklybudget:

- For non-docker installations, see [Flask Application - Database Migrations](#)
- For Docker installations, no special action is needed.
- For development installations, see [Development - Alembic DB Migrations](#)

In all cases, you should always perform a full backup of your database before an upgrade.

## 6.2.4 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.

There are also some additional environment variables available:

- `BIWEEKLYBUDGET_LOG_FILE` - By default, the Flask application's logs go to `STDOUT`. The `BIWEEKLYBUDGET_LOG_FILE` environment variable can be set to the absolute path of a file, to cause Flask application logs to go to the file *in addition to* `STDOUT`.

## 6.2.5 Running Locally

### 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 and on running the Flask development server, see *Flask App*.

## 6.2.6 Running In 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`. It is recommended that you run a specific version number, and that you make sure to perform a database backup before upgrading.

The only dependencies for a Docker installation are:

- MySQL, which can be run via Docker ([MariaDB official image](#) 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. The image also automatically runs database migrations in a safe manner at start, before starting the Flask application.

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 correspond 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
```

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.

### MySQL Connection Errors

On resource-constrained systems or with MySQL servers tuned for minimal resource utilization, you may see the Flask application returning HTTP 500 errors after periods of inactivity, with the Flask application log reporting something like “Lost connection to MySQL server during query” and MySQL reporting “Aborted connection” errors. This is due to connections in the SQLAlchemy connection pool timing out, but the application not being aware of that. If this happens, you can set the `SQL_POOL_PRE_PING` environment variable (to any value). This will enable SQLAlchemy’s `pool_pre_ping` feature (see [Disconnect Handling - Pessimistic](#)) which tests that connections are still working before executing queries with them.

### 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

**Note:** `ofxgetter` support is tentatively being deprecated. Please see [Plaid](#) for the tentative replacement.

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* in Docker with your statements saved to `/home/myuser/statements` on your host computer and your Vault token stored in `/home/myuser/.vault-token` on your host computer, you would set *STATEMENTS\_SAVE\_PATH* in your settings file to `/statements` and *TOKEN\_PATH* to `/.token`, and add to your `docker run` command:

```
-v /home/myuser/statements:/statements \
-v /home/myuser/.vault-token:/.token
```

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

```
docker exec biweeklybudget /bin/sh -c 'cd /statements && /app/bin/ofxgetter'
```

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.

## 6.2.7 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. To get an interactive session, use `python -i bin/db_tester.py`.
- `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*.
- `wishlist2project` - For any projects with “Notes” fields matching an Amazon wishlist URL of a public wishlist (`^https://www.amazon.com/gp/registry/wishlist/`), synchronize the wishlist items to the project. Requires `wishlist==0.1.2`.

## 6.3 Application Usage

This documentation is a work in progress. I suppose if anyone other than me ever tries to use this, I’ll document it a bit more.

### 6.3.1 Currency Formatting and Localization

biweeklybudget supports configurable currency symbols and display/formatting, controlled by the *LOCALE\_NAME* and *CURRENCY\_CODE* settings. The former must specify a [RFC 5646 / BCP 47](#) language tag with a region identifier (i.e. “en\_US”, “en\_GB”, “de\_DE”, etc.). If it is not set in the settings module or via a *LOCALE\_NAME* environment variable, it will be looked up from the *LC\_ALL*, *LC\_MONETARY*, or *LANG* environment variables, in that order. It cannot be a “C” or “C.” locale, as these do not specify currency formatting. The latter, *CURRENCY\_CODE*, must be a

valid [ISO 4217](#) Currency Code (i.e. “USD”, “EUR”, etc.) and can also be set via a `CURRENCY_CODE` environment variable.

In addition, the Fuel Log functionality supports customization of the volume, distance and fuel economy units via a set of settings (which can also be set via environment variables):

- `biweeklybudget.settings.FUEL_VOLUME_UNIT` and `biweeklybudget.settings.FUEL_VOLUME_ABBREVIATION`
- `biweeklybudget.settings.DISTANCE_UNIT` and `biweeklybudget.settings.DISTANCE_UNIT_ABBREVIATION`
- `biweeklybudget.settings.FUEL_ECO_ABBREVIATION`

These settings only effect the display of monetary units in the user interface and in log files. I haven’t made any attempt at actual internationalization of the text, mainly because as far as I know I’m the only person in the world using this software. If anyone else uses it, I’ll be happy to work to accomodate users of other languages or localities.

Right now, regarding localization and currency formatting, please keep in mind the following caveats (which I’d be happy to fix if anyone needs it):

- The currency specified in downloaded OFX files is ignored. Since currency conversion and exchange rates are far outside the scope of this application, it’s assumed that all accounts will be in the currency defined in settings.
- The `wishlist2project` console script that parses Amazon Wishlists and updates Projects / BoMs with their contents currently only supports items priced in USD, and currently only supports wishlists on the `amazon.com` site; these are limitations of the upstream project used for wishlist parsing.
- The database storage of monetary values assumes that they will all be a decimal number, and currently only allows for six digits to the left of the decimal and four digits to the right; this applies to all monetary units from transaction amounts to account balances. As such, if you have any transactions, budgets or accounts (including bank and investment accounts imported via OFX) with values outside of 999999.9999 to -999999.9999 (inclusive), the application will not function. If anyone needs support for larger numbers (or, at the rate I’m going, I’m still working and paying into my pension in about 300 years), the change shouldn’t be terribly difficult.

## 6.4 Flask Application

### 6.4.1 Running Flask Development Server

Flask comes bundled with a builtin development server for fast local development and testing. This is an easy way to take `biweeklybudget` for a spin, but carries some important and critical warnings if you use it with real data. For upstream documentation, see the [Flask Development Server docs](#). Please note that the development server is a single process and defaults to single-threaded, and is only realistically usable by one user.

1. First, setup your environment per [Getting Started - Setup](#).
2. `export FLASK_APP="biweeklybudget.flaskapp.app"`
3. If you’re running against an existing database, see important information in the “Database Migrations” section, below.
4. `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.

## 6.4.2 Database Migrations

If you run the Flask application (whether in the flask development server or a separate WSGI container) against an existing database and there are unapplied Alembic database migrations, it’s very likely that multiple threads or processes will attempt to perform the same migrations at the same time, and leave the database in an inconsistent and unusable state. As such, there are two important warnings:

1. Always be sure that you have a recent database backup before upgrading.
2. You must manually trigger database migrations before starting Flask. This can be done by running the `initdb` console script provided by the `biweeklybudget` package (`bin/initdb` in your virtualenv).

## 6.4.3 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.

## 6.4.4 Logging

By default, the Flask application’s logs go to `STDOUT`. The `BIWEEKLYBUDGET_LOG_FILE` environment variable can be set to the absolute path of a file, to cause Flask application logs to go to the file *in addition to* `STDOUT`.

## 6.4.5 MySQL Connection Errors

See *Getting Started - MySQL Connection Errors* for some information on handling MySQL connection errors.

## 6.5 Plaid

Plaid is a company that provides API-based tools and solutions for interfacing with financial institutions. Among their products is a transaction API that handles authentication with financial institutions and provides ReST access to transaction and balance information. As of March 2020, they provide free developer accounts with access to up to 100 “items” (financial institution accounts), and production accounts with a pay-as-you-go pricing model that is quite affordable.

With many of my banks and credit card companies discontinuing support for OFX Direct Connect citing security concerns (mainly that most of them use OFX 1.x implementations that require your normal credentials in cleartext), or breaking the OFX protocol in ways that only Intuit seems to support, I needed an alternative to the pain of writing error-prone screen scrapers. Plaid seems to be this solution, is reasonably priced (or free for initial testing with a

developer account), and supports all of my accounts. As such, I'm going to be discontinuing development on the *OFX Transaction Downloading* component and focusing on Plaid for the future. Plaid provides a simple, easy, secure way to retrieve balance and transaction information for accounts.

The Plaid component uses the same “OFX” models in biweeklybudget that the older OFX Direct Connect component used. Transactions retrieved via Plaid will still show up on the “OFX Transactions” view, and reconciling works the same way. Both Plaid and OFX write their data into the same models and database tables.

**IMPORTANT** As of late 2022, a handful of financial institutions, Chase being the most notable, require [OAuth2 integration via Plaid](#). This requires that your Plaid account request and be approved for Production environment access, which is a non-trivial process that requires additional legal agreements and a security and privacy review. In addition, Chase has their own partner review process that includes a security review. While it is *possible* for an individual to pass these reviews for a personal project, it's far from trivial. I would only recommend this for folks who have professional experience developing and operating applications that handle financial data, and who intend to operate biweeklybudget in a similar fashion.

### 6.5.1 Configuration

The first step to using the Plaid transaction support is registering for an account with [Plaid](#). As that process may change, I recommend using their site to find out how. As of this writing, there is a large “Get API keys” button on the homepage to get your started. You will initially be given testing / sandbox keys, which can only retrieve information about fake accounts that Plaid maintains (though it is realistic data, and can be used to see how biweeklybudget integrates with Plaid). In order to retrieve your real data, you will need to request Development access. Once you have that, you can proceed.

biweeklybudget needs to be configured with your Plaid credentials. I highly recommend setting these as environment variables rather than hard-coding them in your settings file (if you use one). You will need to run biweeklybudget with the following Plaid-related environment variables / settings:

- `PLAID_CLIENT_ID` - Your Client ID credential, provided by Plaid.
- `PLAID_SECRET` - Your Secret credential, provided by Plaid.
- `PLAID_ENV` - The Plaid environment name to connect to. This must be one of “Sandbox”, “Development”, or “Production”, and must match the environment that your credentials are for.
- `PLAID_PRODUCTS` - The Plaid products you are requesting access to. Right now, for biweeklybudget, this should be `transactions`.
- `PLAID_COUNTRY_CODES` - A comma-separated list of country codes that you want to be able to select institutions from. Only `US` has been tested.

### 6.5.2 Usage

#### Linking Accounts to Plaid

1. Click the “Plaid Update” link in the left navigation menu.
2. Click the “Link (Add Plaid Item)” button under the “Plaid Items” table. This will connect to one of your financial institution logins via Plaid (an “Item” in Plaid parlance). Note that one Item (institution login) may have multiple accounts.
3. When the Plaid Link process is complete the new item will be shown in the Plaid Items table.
4. Click the “Accounts” link in the left navigation menu.
5. Click the name of the Account that you want to link to one of the new Plaid Accounts.

6. In the Edit Account modal, scroll to the bottom and select the appropriate Item and Account in the “Plaid Account” dropdown.
7. Click “Save changes”.

## Updating Transactions via UI

Updating through the UI will retrieve transactions for the last 30 days. If you want to retrieve more than that, you must do so *via the API*.

1. Click the “Plaid Update” link in the left navigation menu.
2. In the “Plaid Update Transactions” table, select the Plaid Items that you want to update transactions for.
3. Click the “Update Transactions” button at the bottom of the table.
4. When the update is complete, you will be redirected to a page showing results in a table.

## Updating Transactions via API

Transactions can be updated via a simple API at the same `/plaid-update` endpoint. This API can return either a JSON or human-readable plain-text output depending on the `Accept` header. For full documentation, see the documentation of `PlaidUpdate` and `post()`.

In short, the endpoint takes a POST or GET request that specifies an `item_ids` parameter as a string comma-separated list of `PlaidItem` IDs to update, or the special string `ALL` to update all Items. Optionally, you can specify a `num_days` parameter to retrieve transactions for something other than the last 30 days. The response is either JSON if the `Accept` header is set to `application/json` or human-readable plain text if set to `text/plain` (if set to any other value, it will return the full HTML that would be sent to the browser).

The following examples assume that biweeklybudget is available at `http://127.0.0.1:8080`

To update transactions for all Plaid Items via a GET request and return human-readable text:

```
$ curl -H 'Accept: text/plain' 'http://127.0.0.1:8080/plaid-update?item_ids=ALL'
% Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
           Dload  Upload   Total      Spent    Left     Speed

  0     0     0     0     0     0     0     0  0:00:00  0:00:00  0:00:00     0
  0     0     0     0     0     0     0     0  0:00:00  0:00:00  0:00:00     0
  0     0     0     0     0     0     0     0  0:00:00  0:00:01  0:00:00     0
  0     0     0     0     0     0     0     0  0:00:00  0:00:02  0:00:00     0
  0     0     0     0     0     0     0     0  0:00:00  0:00:03  0:00:00     0
100   874   100   874     0     0    231     0  0:00:03  0:00:03  0:00:00   232
AcctOne (plaidItemId1): 23 updated, 0 added (stmts: [21728])
AcctTwo (plaidItemId2): 31 updated, 0 added (stmts: [21729])
AcctThree (plaidItemId3): 35 updated, 3 added (stmts: [21730])
TOTAL: 89 updated, 3 added, 0 account(s) failed
```

To update transactions for Plaid Item with IDs `plaidItemId1` for the last 60 days via a POST, and return JSON:

```
$ curl -XPOST -H 'Accept: application/json' -d 'item_ids=plaidItemId1&num_days=60' \
  http://127.0.0.1:8080/plaid-update
[{"added":0,"exception":"None","item_id":"plaidItemId1","statement_ids":[21747],
  "success":true,"updated":35}]
```

### 6.5.3 Troubleshooting

API responses from Plaid are logged at debug-level. The UI process of linking an account via Plaid happens mostly in client-side JavaScript, which logs pertinent information to the browser's console log. The [Plaid Dashboard](#) also provides some useful debug information, especially when correlated with the `link_token` and/or `item_id` that should be logged by biweeklybudget.

### 6.5.4 Changing Plaid Environments

It may be necessary to change Plaid environments, such as if you started using the Development environment and then switched to Production for OAuth2 integrations. This process will require setting up Plaid again.

Also **note** that Plaid `transaction_id` (our `fitid`) `_will_` change between environments. As such, you should update transactions in the old environment immediately before switching environments, then update transactions in the new environment, and you will need to manually ignore any transactions that are duplicates.

1. Un-associate all of your Accounts from Plaid Accounts. This can be done manually via the Account edit modal or by running the following SQL query directly against the database: `UPDATE accounts SET plaid_item_id=NULL, plaid_account_id=NULL;`
2. Delete all of your Plaid Accounts and Plaid Items from the database: `DELETE FROM plaid_accounts;`  
`DELETE FROM plaid_items;`
3. Update your configuration / environment variables for the new `PLAID_ENV` that you want to use and your `PLAID_SECRET` for that environment.
4. Re-link all of your Plaid items, and then re-associate them with your Accounts.

## 6.6 OFX Transaction Downloading

---

**Important:** OFX support is tentatively being deprecated. Please see [Plaid](#) for the tentative replacement.

---

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 we vendor-in a fork with some bug fixes). 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.

### 6.6.1 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.

## 6.6.2 ofxgetter entrypoint

This package provides an `ofxgetter` command line entrypoint 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).

## 6.6.3 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.

## 6.6.4 Configuring Accounts for Downloading with ofxclient

1. Use the `ofxclient` CLI to configure and test your account, according to the [upstream documentation](#).
2. Store the username and password for your account in Vault, as `username` and `password` keys, respectively, of the same secret (path).
3. Convert `~/ofxclient.ini` to JSON (this will look something like the example below), removing the `institution.username` and `institution.password` keys (these will be read from Vault at runtime).
4. If there is no sensitive information in the resulting JSON, store the JSON string in the `ofxgetter_config_json` attribute of the appropriate `Account` object. This can be done via the `/accounts` view in the Web UI. If there *is* sensitive information in the `ofxclient` configuration JSON, you can store the entire JSON configuration in an additional key on the Vault secret, and then set the `ofxgetter_config_json` attribute to `{"key": "NameOfVaultKeyWithJSON"}`.

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",
```

(continues on next page)

(continued from previous page)

```

        "id": "f87217350cc341e2ba7407cf99dcdede"
    },
    "description": "MyBank",
    "url": "https://ofx.MyBank.com",
    "local_id": "e51fb78f88580a1c2e3bb65bd59495384388abda8796c9bf06dcf",
    "broker_id": "",
    "org": "ORG",
    "id": "98765"
}
}

```

## 6.6.5 Configuring Accounts for Downloading with Selenium

In your *customization package* `<getting_started.customization>`, subclass `ScreenScrapper`. Override the constructor to take whatever keyword arguments are required, and add those to your account's `ofxgetter_config_json` as shown below. `OfxGetter` will instantiate the class passing it the specified keyword arguments in addition to `username`, `password` and `savendir` keyword arguments. `savendir` 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 you need to persist cookies across sessions, look into the `ScreenScrapper` class' `load_cookies()` and `save_cookies()` methods.

```

{
    "class_name": "MyScrapper",
    "module_name": "budget_customization.myscrapper",
    "institution": {},
    "kwargs": {
        "acct_num": "1234"
    }
}

```

This JSON configuration will have the username and password from Vault interpolated as keyword arguments, similar to how they will be added to `institution` for `ofxclient` accounts. As described in `ofxclient` accounts #4, above, you can also store the entire JSON configuration in Vault if desired.

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 ScreenScrapper

logger = logging.getLogger(__name__)

# suppress selenium logging
selenium_log = logging.getLogger("selenium")
selenium_log.setLevel(logging.WARNING)
selenium_log.propagate = True

```

(continues on next page)

(continued from previous page)

```

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('chrome-headless')
        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(

```

(continues on next page)

(continued from previous page)

```

        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

```

## 6.6.6 OFX Related Account Settings

The following attributes on the *Account* model effect OFX downloads and how OFX statements are handled:

- *ofxgetter\_config\_json* - Stores the configuration required for ofxclient- or Selenium-based OFX downloads. See above. This is exposed as the “OFXGetter Config (JSON)” form field when adding or editing accounts through the UI.
- *ofx\_cat\_memo\_to\_name* - This is exposed as the “OFX Cat Memo to Name” checkbox when adding or editing accounts through the UI.
- *negate\_ofx\_amounts* - This is exposed as the “Negate OFX Amounts” checkbox when adding or editing accounts through the UI.

## 6.7 Getting Help

### 6.7.1 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.

## 6.8 Development

To install for development:

1. Fork the [biweeklybudget](#) repository on GitHub
2. Create a new branch off of master in your fork, and clone it locally then:

```

$ cd biweeklybudget
$ python3 -mvenv venv
$ source venv/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`.

### 6.8.1 Guidelines

- pep8 compliant with some exceptions (see `pytest.ini`)

- 100% test coverage with pytest (with valid tests)

## 6.8.2 Docker Database Container

To run a Dockerized database for your test environment:

```
$ docker run -d --name budgettest -p 13306:3306 --env MYSQL_ROOT_PASSWORD=dbroot --
→env MYSQL_ROOT_HOST='% ' mariadb:10.4.7
$ export DB_CONNSTRING='mysql+pymysql://root:dbroot@127.0.0.1:13306/budgettest?
→charset=utf8mb4'; export MYSQL_HOST=127.0.0.1; export MYSQL_PORT=13306; export
→MYSQL_USER=root; export MYSQL_PASS=dbroot; export MYSQL_DBNAME=budgettest; export
→MYSQL_DBNAME_LEFT=alembicLeft; export MYSQL_DBNAME_RIGHT=alembicRight
$ python dev/setup_test_db.py
# run your tests
$ docker stop budgettest && docker rm budgettest
```

## 6.8.3 Test Database Setup

After starting your test database (i.e. *Docker Database Container* above or by running a local MySQL / MariaDB server) and exporting your connection string and MySQL-related variables, i.e.:

```
export DB_CONNSTRING='mysql+pymysql://root:dbroot@127.0.0.1:13306/budgettest?
charset=utf8mb4'; export MYSQL_HOST=127.0.0.1; export MYSQL_PORT=13306; export
MYSQL_USER=root; export MYSQL_PASS=dbroot; export MYSQL_DBNAME=budgettest;
export MYSQL_DBNAME_LEFT=alembicLeft; export MYSQL_DBNAME_RIGHT=alembicRight
```

you can set up the test databases by running `dev/setup_test_db.py`

## 6.8.4 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.

## 6.8.5 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 run only one module at a time like:

```
tox -e plaid -- biweeklybudget/tests/acceptance/test_plaidlink.py
```

Or even just one class by specifying its name with `-k` like:

```
tox -e plaid -- -k TestClassName biweeklybudget/tests/acceptance/
test_plaidlink.py
```

### 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. Note that they're currently configured to use Headless Chrome; running them locally will require a modern Chrome version that supports the `--headless` flag (Chrome 59+) and a matching version of `chromedriver`.

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 `conftest.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 headless Chrome.

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.

### Running Acceptance Tests Against Docker

The acceptance tests have a “hidden” hook to run against an already-running Flask application, run during the `docker` tox environment build. **Be warned** that the acceptance tests modify data, so they should never be run against a real database. This hook is controlled via the `BIWEEKLYBUDGET_TEST_BASE_URL` environment variable. If this variable is set, the acceptance tests will not start a Flask server, but will instead use the specified URL. The URL must not end with a trailing slash.

### Database Migration Tests

There is a `migrations` tox environment that runs `alembic-verify` tests on migrations. This tests running through all upgrade migrations in order and then all downgrade migrations in order, and also tests that the latest (head) migration revision matches the current state of the models.

The environment also runs manually-curated acceptance tests for any migrations that involve data manipulation.

This tox environment is configured via environment variables. Please note that it requires *two* test databases.

- **MYSQL\_HOST** - MySQL DB hostname/IP. Defaults to `127.0.0.1`
- **MYSQL\_PORT** - MySQL DB Port. Defaults to `3306`.
- **MYSQL\_USER** - MySQL DB username. Defaults to `root`.
- **MYSQL\_PASS** - MySQL DB password. Defaults to no password.
- **MYSQL\_DBNAME\_LEFT** - MySQL Database name for the first (“left”) test database.
- **MYSQL\_DBNAME\_RIGHT** - MySQL Database name for the second (“right”) test database.

## 6.8.6 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). This must be run *before* the model changes are applied to the DB. If adding new models, make sure to import the model class in `models/__init__.py`.
- 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`.

## 6.8.7 Database Debugging

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

To get an interactive Python shell with the database initialized, use `python -i bin/db_tester.py`.

## 6.8.8 Performance Profiling and Logging

### Database

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

If you set the `SQL_QUERY_PROFILE` environment variable to “true”, event handlers will be inserted into the SQLAlchemy subsystem that log (at DEBUG level) each query that’s run and the time in seconds that the query took to execute. This will also result in logging each query as it is executed.

### Flask Application

When running the application in development mode using `flask rundev`, the `werkzeug` WSGI handler will append the time taken to serve each request to the request log, in the format `[Nms]` where `N` is an integer number of milliseconds.

When running the application in Docker, the time taken to serve the request in decimal seconds will be appended to the end of the Gunicorn access logs, in the format `[N.Ns]` where `N.N` is the decimal number of seconds.

## 6.8.9 Docker Image Build

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

## 6.8.10 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`.

## 6.8.11 Vendored Requirements

A number of this project's dependencies are or were seemingly abandoned, and weren't responding to bugfix pull requests or weren't pushing new releases to PyPI. This made the installation process painful, as it required `pip install -r requirements.txt` to pull in git requirements.

In an attempt to make installation easier, we've vendored any git requirements in to this repository under `biweeklybudget/vendored/`. The intent is to move these back to `setup.py` requirements when each project includes the fixes we need in its official release on PyPI.

To updated the vendored projects:

1. Update `biweeklybudget/vendored/install_vendored.sh`
2. Run `cd biweeklybudget/vendored && install_vendored.sh`
3. Ensure that our main `setup.py` includes all dependencies of the vendored projects.

## 6.8.12 Release Checklist

1. Ensure that `CHANGES.rst` has entries for all changes.
2. Ensure that the version in `version.py` has been incremented.
3. Update the header in `CHANGES.rst` to have the new version number and release date.
4. Regenerate all docs with `tox -e docs -e jsdoc -e screenshots` and commit the results.
5. Merge all of the above to master.
6. To cut release, tag master.

## 6.9 Changelog

### 6.9.1 1.1.1 (2022-12-30)

- Docker build - don't include `-dirty` in version/tag when building in GHA
- Document how to change Plaid environments
- GHA - Push built Docker images to Docker Hub, for builds of master branch

- Document triggering a Plaid update via the `/plaid-update` endpoint.
- Change `/plaid-update` endpoint argument name from `account_ids` to `item_ids`.
- Add `num_days` parameter support to `/plaid-update` endpoint.

## 6.9.2 1.1.0 (2022-12-29)

### Breaking Changes

- Support for Python versions prior to 3.8 have been dropped; Docker image and testing is now done against Python 3.10.
- Valid values for the `PLAID_ENV` setting / environment variable are now the strings “Production”, “Development”, or “Sandbox” to match the attribute names of `plaid.configuration.Environment`. Previously these were lower-case instead of capitalized.
- The `PLAID_PUBLIC_KEY` setting / environment variable has been removed.
- OFX support is now **deprecated**; going forward, only Plaid will be supported.

### All Changes

- **Drop Python 2 Support and Python 3.5 Support** - biweeklybudget no longer supports Python 2 (2.7) or Python 3.5. Python versions 3.6-3.8 are tested, and development is now done on 3.8.
- [Issue #201](#) - Fix **major** bug in calculation of “Remaining” amount for pay periods, when one or more periodic budgets have a greater amount spent than allocated and a \$0 starting balance. In that case, we were using the allocated amount instead of the spent amount (i.e. if we had a periodic budget with a \$0 starting balance and a \$2 ScheduledTransaction, and converted that ScheduledTransaction to a \$1000 Transaction, the overall PayPeriod remaining amount would be based on the \$2 not the \$1000).
- Add testing for Python 3.7 and 3.8, and make 3.8 the default for tests and tox environments.
- TravisCI updates for Python 3.7 and 3.8.
- Switch base image for Docker from `python:3.6.4-alpine3.7` to `python:3.8.1-alpine3.11`.
- [Issue #198](#) - Fix broken method of retrieving current US Prime Rate. Previously we used `marketwatch.com` for this but they’ve introduced javascript-based bot protection on their site (which is ironic since we were reading a value from the page’s `meta` tags, which are specifically intended to be read by machines). Switch to using `wsj.com` instead and (ugh) parsing a HTML table. This *will* break when the format of the table changes. As previously, we cache this value in the DB for 48 hours in order to be a good citizen.
- [Issue #197](#) - Add notification for case where balance of all budget-funding accounts is *more* than sum of standing budgets, current payperiod remaining, and unreconciled. This is the opposite of the similar notification that already exists, intended to detect if there is money in accounts not accounted for in the budgets.
- [Issue #196](#) - Don’t include inactive budgets in Budget select elements on Transaction Modal form, unless it’s an existing Transaction using that budget.
- [Issue #204](#) - Add support for account transfer between non-Credit accounts.
- Many dependency updates:
  - Upgrade SQLAlchemy from 1.2.0 to 1.2.11 for [python 3 bug fix \(4291\)](#).
  - Upgrade SQLAlchemy from 1.2.11 to 1.3.13 for [CVE-2019-7548](#) and [CVE-2019-7164](#).
  - Upgrade Flask from 0.12.2 to 1.0.2 for [CVE-2018-1000656](#).

- Upgrade cryptography from 2.1.4 to 2.3.1 for [CVE-2018-10903](#).
- Upgrade Jinja2 from 2.10 to 2.10.3 for [CVE-2019-10906](#).
- Upgrade to latest version for all dependencies.
- Remove `convert_unicode` argument from SQLAlchemy DB engine arguments per SQLAlchemy 1.3 upgrade guide / [SQLAlchemy #4393](#).
- Numerous updates to fix `tox` tests.
- Implement transaction downloading via [Plaid](#).
- Switch tests from deprecated `pep8` / `pytest-pep8` packages to `pycodestyle` / `pytest-pycodestyle`.
- Add optional `VERSIONFINDER_DEBUG` env var; set to `true` to enable logging for `versionfinder` / `pip` / `git`.
- Drop testing for Python 3.6; move default test environment to 3.9.
- Add `git` to Docker image.
- Move testing and runtime to Python 3.10, and get all test environments running successfully.
- Move CI from TravisCI to GitHub Actions and remove all traces of TravisCI.
- Add acceptance test coverage of the Plaid Link process.
- Updates for `tox` 4.0.6.
- Update Plaid API client to latest version
  - Valid values for the `PLAID_ENV` setting / environment variable are now the strings “Production”, “Development”, or “Sandbox” to match the attribute names of `plaid.configuration.Environment`.
  - The `PLAID_PUBLIC_KEY` setting / environment variable has been removed.

### 6.9.3 1.0.0 (2018-07-07)

- Fix major logic error in Credit Card Payoff calculations; interest fees were ignored for the current month/statement, resulting in “Next Payment” values significantly lower than they should be. Fixed to use the last Interest Charge retrieved via OFX (or, if no interest charges present in OFX statements, prompt users to manually enter the last Interest Charge via a new modal that will create an `OFXTransaction` for it) as the interest amount on the first month/statement when calculating payoffs. This fix now returns Next Payment values that aren’t identical to sample cards, but are significantly closer (within 1-2%).
- [Issue #105](#) - Major refactor to the Transaction database model. This is transparent to users, but causes massive database and code changes. This is the first step in supporting Transaction splits between multiple budgets:
  - A new `BudgetTransaction` model has been added, which will support a one-to-many association between Transactions and Budgets. This model associates a Transaction with a Budget, and a currency amount counted against that Budget. This first step only supports a one-to-one relationship, but a forthcoming change will implement the one-to-many budget split for Transactions.
  - The database migration for this creates `BudgetTransactions` for every current Transaction, migrating data to the new format.
  - The `budget_id` attribute and `budget` relationship of the Transaction model has been removed, as that information is now in the related `BudgetTransactions`.
  - A new `planned_budget_id` attribute (and `planned_budget` relationship) has been added to the Transaction model. For Transactions that were created from `ScheduledTransactions`, this attribute/relationship stores the original planned budget (distinct from the actual budget now stored in `BudgetTransactions`).

- The Transaction model now has a `budget_transactions` back-populated property, containing a list of all associated BudgetTransactions.
- The Transaction model now has a `set_budget_amounts()` method which takes a single dict mapping either integer Budget IDs or Budget objects, to the Decimal amount of the Transaction allocated to that Budget. While the underlying API supports an arbitrary number of budgets, the UI and codebase currently only supports one.
- The Transaction constructor now accepts a `budget_amounts` keyword argument that passes its value through to `set_budget_amounts()`, for ease of creating Transactions in one call.
- `Transaction.actual_amount` is no longer an attribute stored in the database, but now a hybrid property (read-only) generated from the sum of amounts of all related BudgetTransactions.
- Add support to serialize property values of models, in addition to attributes.
- Relatively major and sweeping code refactors to support the above.
- Switch tests from using deprecated `pytest-capturelog` to using `pytest` built-in log capturing.
- Miscellaneous fixes to unit and acceptance tests, and docs build.
- Finish converting *all* code, including tests and sample data, from using floats to Decimals.
- Acceptance test fix so that `pytest-selenium` can take full page screenshots with Chromedriver.
- [PR #180](#) - Acceptance test fix so that the `testflask LiveServer` fixture captures server logs, and includes them in test HTML reports (this generates a temporary file per-test-run outside of `pytest`'s control).
- Fix bug found where simultaneously editing the Amount and Budget of an existing Transaction against a Standing Budget would result in incorrect changes to the balances of the Budgets.
- Add a new `migrations` tox environment that automatically tests all database migrations (forward and reverse) and also validates that the database schema created from the migrations matches the one created from the models.
- Add support for writing tests of data manipulation during database migrations, and write tests for the migration in for [Issue 105](#), above.
- Add support for `BIWEEKLYBUDGET_LOG_FILE` environment variable to cause Flask application logs to go to a file *in addition to* `STDOUT`.
- Add support for `SQL_POOL_PRE_PING` environment variable to enable SQLAlchemy `pool_pre_ping` feature (see [Disconnect Handling - Pessimistic](#)) for resource-constrained systems.
- Modify acceptance tests to retry up to 3 times, 3 seconds apart, if a `ConnectionError` (or subclass thereof) is raised when constructing the Selenium driver instance. This is a workaround for intermittent `ConnectionResetErrors` in TravisCI.
- [Issue #177](#)
  - Add SQL query timing support via `SQL_QUERY_PROFILE` environment variable.
  - When running under `flask rundev`, append the number of milliseconds taken to serve the request to the `werkzeug` access log.
  - When running under Docker/Gunicorn, append the decimal number of seconds taken to serve the request to the Gunicorn access log.
- [Issue #184](#) - Redact database password from `/help` view, and change `/help` view to show Version containing git commit hash for pre-release/development Docker builds.
- [Issue #183](#)
  - Add UI link to ignore reconciling an OFXTransaction if there will not be a matching Transaction.
  - Remove default values for the Account model's `re_` fields in preparation for actually using them.

- Replace the Account model's `re_fee` field with separate `re_late_fee` and `re_other_fee` fields.
  - Add UI support for specifying Interest Charge, Interest Paid, Payment, Late Fee, and Other Fee regexes on each account.
  - Add DB event handler on new or changed OFXTransaction, to set `is_*` fields according to Account `re_*` fields.
  - Add DB event handler on change to Account model `re_*` fields, that triggers `OFXTransaction.update_is_fields()` to recalculate using the new regex.
  - Change `OFXTransaction.unreconciled` to filter out OFXTransactions with any of the `is_*` set to True.
- Upgrade chromedriver in TravisCI builds from 2.33 to 2.36, to fix failing acceptance tests caused by Ubuntu upgrade from Chrome 64 to 65.
  - Fix bug in `/budgets` view where “Spending By Budget, Per Calendar Month” chart was showing only inactive budgets instead of only active budgets.
  - [Issue #178](#) - UI support for splitting Transactions between multiple Budgets.
  - Have frontend forms submit as JSON POST instead of urlencoded.
  - Properly capture Chrome console logs during acceptance tests.
  - Bump `versionfinder` requirement version to 0.1.3 to work with pip 9.0.2.
  - On help view, show long version string if we have it.
  - [Issue #177](#) - Fix bug in `flask_runde` logging.
  - Many workarounds for flaky acceptance tests, including some for the selenium/Chrome “Element is not clickable at point... Other element would receive the click” error.
  - `biweeklybudget.screenscraper.ScreenScraper` - Save webdriver and browser logs on failure, and set Chrome to capture all logs.
  - `biweeklybudget.screenscraper.ScreenScraper` - Add option to explicitly set a User-Agent on Chrome or PhantomJS.
  - [Issue #192](#) - Fix bug where the `is_` fields weren't set on OFXTransactions when created via `ofxgetter` remote API.
  - `ofxgetter` - add support to list all accounts at the Institution of one account
  - `ofxgetter` - add ability to specify how many days of data to retrieve

### 6.9.4 0.7.1 (2018-01-10)

- [Issue #170](#) - Upgrade **all** python dependencies to their latest versions.
- [Issue #171](#) - Upgrade Docker base image from `python:3.6.3-alpine3.4` to `python:3.6.4-alpine3.7`.
- [Issue #157](#) - Remove PhantomJS from Docker image, as it's broken and shouldn't be needed.
- Switch TravisCI builds from Docker (`sudo: false`) to VM (`sudo: enabled`) infrastructure.

### 6.9.5 0.7.0 (2018-01-07)

**This version has a remote OFX upload incompatibility. See below.**

- [Issue #156](#) - Add headless chrome support to `screenscraper.py`.
- Remove `pluggy` transient dependency from `requirements.txt`; was breaking builds.
- Following `pytest`, drop testing of and support for Python 3.3.
- [Issue #159](#) - Implement internationalization of volume and distance units for Fuel Log pages. This change introduces five new settings: `FUEL_VOLUME_UNIT`, `FUEL_VOLUME_ABBREVIATION`, `DISTANCE_UNIT`, `DISTANCE_UNIT_ABBREVIATION` and `FUEL_ECO_ABBREVIATION`.
- [Issue #154](#) - Fix documentation errors on the Getting Started page, “Running `ofxgetter` in Docker” section.
- [Issue #152](#) - Fix for bug where new Transactions could be entered against inactive budgets. Ensure that existing transactions against inactive budgets can still be edited, but existing transactions cannot be changed to an inactive budget.
- [Issue #161](#) - Fix bug where Transactions against inactive budgets weren’t counted towards `payperiod` overall or per-budget totals.
- [Issue #163](#) - Include next payment amount on Credit Payoffs view.
- [Issue #84](#) - Remove vendored-in `ofxparse` package now that [my PR #127](#) has been merged and released on PyPI. **Important note:** The version of `ofxparse` is changed in this release. If you are using `ofxgetter -r` (remote API mode), the versions of `ofxparse` (and therefore `biweeklybudget/ofxgetter`) must match between the client and server.
- [Issue #165](#) - Remove vendored-in `wishlist` package now that [my PR #8](#) has been merged and released on PyPI.
- [Issue #155](#) - Refactor `ofxgetter` to fix bug where `SETTINGS_MODULE` was still required even if running remotely.

## 6.9.6 0.6.0 (2017-11-11)

- [PR #140](#) - Support user-configurable currencies and currency formatting. This isn’t all-out localization, but adds `CURRENCY_CODE` and `LOCALE_NAME` configuration settings to control the currency symbol and formatting used in the user interface and logs.
- [PR #141](#) - Switch acceptance tests from PhantomJS to headless Chrome.
- Switch docs build screenshot script to use headless Chrome instead of PhantomJS.
- [Issue #142](#) - Speed up acceptance tests. The acceptance tests recently crossed the 20-minute barrier, which is unacceptable. This makes some improvements to the tests, mainly around combining classes that can be combined and also using `mysql/mysqldump` to refresh the DB, instead of refreshing and recreating via the ORM. That offers a approximately 50-90% speed improvement for each of the 43 refreshes. Unfortunately, it seems that the majority of time is taken up by `pytest-selenium`; see [Issue 142](#) for further information.
- [Issue #125](#) - Switch Docker image base from `python:3.6.1` (Debian) to `python:3.6.3-alpine3.4` (Alpine Linux); drops final image size from 876MB to 274MB. (*Note:* Alpine linux does not have `/bin/bash`.)
- [Issue #138](#) - Improvements to build process
  - Run acceptance tests against the built Docker container during runs of the `docker tox` environment / `tests/docker_build.py`.
  - Reminder to sign git release tags
  - Add `dev/release.py` script to handle GitHub releases.
- [Issue #139](#) - Add field to Budget model to allow omitting specific budgets from spending graphs (the graphs on the Budgets view).

## 6.9.7 0.5.0 (2017-10-28)

This release includes database migrations.

- [Issue #118](#) - PR to fix bugs in the `wishlist` dependency package, and vendor that patched version in under `biweeklybudget.vendored.wishlist`.
- [Issue #113](#) - vendor in other git requirements (`ofxclient` and `ofxparse`) that seem unmaintained or inactive, so we can install via `pip`.
- [Issue #115](#) - In Transactions view, add ability to filter by budget.
- Change `BiweeklyPayPeriod` class to never convert to floats (always use `decimal.Decimal` types).
- [Issue #124](#) - Major changes to the `ofxgetter` and `ofxbackfiller` console scripts; centralize all database access in them to the new `biweeklybudget.ofxapi.local.OfxApiLocal` class and allow these scripts to function remotely, interacting with the ReST API instead of requiring direct database access.
- [Issue #123](#) - Modify the Credit Payoffs view to allow removal of Increase and Onetime Payment settings lines.
- [Issue #131](#) - Add better example data for screenshots.
- [Issue #117](#) and [#133](#) - Implement and then revert out a failed attempt at automatic balancing of budgets in the previous pay period.
- [Issue #114](#)
  - Add `transfer_id` field and `transfer` relationship to Transaction model, to link the halves of budget transfer transactions in the database. The alembic migration for this release iterates all Transactions in the database, and populates these links based on inferences of the description, date, `account_id` and notes fields of sequential pairs of Transactions. (Note: this migration would likely miss some links if two transfers were created simultaneously, and ended up with the Transaction IDs interleaved).
  - Identify transfer Transactions on the Edit Transaction modal, and provide link to the matching Transaction.
  - Add graph of spending by budget to Budgets view.
- [Issue #133](#) - Change `BiweeklyPayPeriod` model to only use actual spent amount when creating remaining amount on payperiods in the past. Previously, all pay periods calculated the overall “remaining” amount as income minus the greater of `allocated` or `spent`; this resulted in pay periods in the past still including `allocated-but-not-spent` amounts counted against “remaining”.

## 6.9.8 0.4.0 (2017-08-22)

- Have `ofxgetter` enable `ofxclient` logging when running at DEBUG level (`-vv`).
- Bump `ofxclient` requirement to my `vanguard-fix` branch for [PR #47](#).
- [Issue #101](#) - Fix static example amounts on `/projects` view.
- [Issue #103](#) - Show most recent MPG in notification box after adding fuel fill.
- [Issue #97](#) - Fix integration tests that are date-specific and break on certain dates (run all integration tests as if it were a fixed date).
- [Issue #104](#) - Relatively major changes to add calculation of Credit account payoff times and amounts.
- [Issue #107](#) - Fix bug where Budget Transfer modal dialog would always default to current date, even when viewing past or future pay periods.
- [Issue #48](#) - UI support for adding and editing accounts.

### 6.9.9 0.3.0 (2017-07-09)

- Issue #88 - Add tracking of cost for Projects and Bills of Materials (BoM) for them.
- Add script / entry point to sync Amazon Wishlist with a Project.
- Issue #74 - Another attempt at working over-balance notification.

### 6.9.10 0.2.0 (2017-07-02)

- Fix `/pay_period_for` redirect to be a 302 instead of 301, add redirect logging, remove some old debug logging from that view.
- Fix logging exception in `db_event_handlers` on initial data load.
- Switch `ofxparse` requirement to use upstream repo now that <https://github.com/jseutter/ofxparse/pull/127> is merged.
- Issue #83 - Fix 500 error preventing display of balance chart on `/` view when an account has a None ledger balance.
- Issue #86 - Allow budget transfers to periodic budgets.
- Issue #74 - Warning notification for low balance should take current pay period's overall allocated sum, minus reconciled transactions, into account.
- Fix some template bugs that were causing HTML to be escaped into plaintext.
- Issue #15 - Add pay period totals table to index page.
- Refactor form generation in UI to use new FormBuilder javascript class (DRY).
- Fix date-sensitive acceptance test.
- Issue #87 - Add fuel log / fuel economy tracking.

### 6.9.11 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.

### 6.9.12 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.

### 6.9.13 0.1.0 (2017-05-07)

- Initial Release

## 6.10 biweeklybudget

### 6.10.1 biweeklybudget package

#### Subpackages

#### `biweeklybudget.flaskapp` package

#### Subpackages

#### `biweeklybudget.flaskapp.views` package

#### Submodules

#### `biweeklybudget.flaskapp.views.accounts` module

```
class biweeklybudget.flaskapp.views.accounts.AccountAjax
    Bases: flask.views.MethodView
    Handle GET /ajax/account/<int:account_id> endpoint.
    get (account_id)
    methods = {'GET' }
```

```

class biweeklybudget.flaskapp.views.accounts.AccountFormHandler
    Bases: biweeklybudget.flaskapp.views.formhandlerview.FormHandlerView
    Handle POST /forms/account
    methods = {'POST'}
    submit (data)
        Handle form submission; create or update models in the DB. Raises an Exception for any errors.
        Parameters data (dict) – submitted form data
        Returns message describing changes to DB (i.e. link to created record)
        Return type str
    validate (data)
        Validate the form data. Return None if it is valid, or else a hash of field names to list of error strings for
        each field.
        Parameters data (dict) – submitted form data
        Returns None if no errors, or hash of field name to errors for that field

class biweeklybudget.flaskapp.views.accounts.AccountTxfrFormHandler
    Bases: biweeklybudget.flaskapp.views.formhandlerview.FormHandlerView
    Handle POST /forms/account_transfer
    methods = {'POST'}
    submit (data)
        Handle form submission; create or update models in the DB. Raises an Exception for any errors.
        Parameters data (dict) – submitted form data
        Returns message describing changes to DB (i.e. link to created record)
        Return type str
    validate (data)
        Validate the form data. Return None if it is valid, or else a hash of field names to list of error strings for
        each field.
        Parameters data (dict) – submitted form data
        Returns None if no errors, or hash of field name to errors for that field

class biweeklybudget.flaskapp.views.accounts.AccountsView
    Bases: flask.views.MethodView
    Render the GET /accounts view using the accounts.html template.
    get ()
    methods = {'GET'}

class biweeklybudget.flaskapp.views.accounts.OneAccountView
    Bases: flask.views.MethodView
    Render the /accounts/<int:acct_id> view using the account.html template.
    get (acct_id)
    methods = {'GET'}

```

## biweeklybudget.flaskapp.views.budgets module

**class** `biweeklybudget.flaskapp.views.budgets.BudgetAjax`

Bases: `flask.views.MethodView`

Handle GET /ajax/budget/<int:budget\_id> endpoint.

**get** (*budget\_id*)

**methods** = {'GET'}

**class** `biweeklybudget.flaskapp.views.budgets.BudgetFormHandler`

Bases: `biweeklybudget.flaskapp.views.formhandlerview.FormHandlerView`

Handle POST /forms/budget

**methods** = {'POST'}

**submit** (*data*)

Handle form submission; create or update models in the DB. Raises an Exception for any errors.

**Parameters** *data* (*dict*) – submitted form data

**Returns** message describing changes to DB (i.e. link to created record)

**Return type** `str`

**validate** (*data*)

Validate the form data. Return None if it is valid, or else a hash of field names to list of error strings for each field.

**Parameters** *data* (*dict*) – submitted form data

**Returns** None if no errors, or hash of field name to errors for that field

**class** `biweeklybudget.flaskapp.views.budgets.BudgetSpendingChartView`

Bases: `flask.views.MethodView`

Handle GET /ajax/chart-data/budget-spending/<str:aggregation> endpoint.

**\_budget\_names** ()

**\_by\_month** ()

**\_by\_pay\_period** ()

**get** (*aggregation*)

**methods** = {'GET'}

**class** `biweeklybudget.flaskapp.views.budgets.BudgetTxfrFormHandler`

Bases: `biweeklybudget.flaskapp.views.formhandlerview.FormHandlerView`

Handle POST /forms/budget\_transfer

**methods** = {'POST'}

**submit** (*data*)

Handle form submission; create or update models in the DB. Raises an Exception for any errors.

**Parameters** *data* (*dict*) – submitted form data

**Returns** message describing changes to DB (i.e. link to created record)

**Return type** `str`

**validate** (*data*)

Validate the form data. Return None if it is valid, or else a hash of field names to list of error strings for each field.

**Parameters** *data* (*dict*) – submitted form data

**Returns** None if no errors, or hash of field name to errors for that field

**class** `biweeklybudget.flaskapp.views.budgets.BudgetsView`

Bases: `flask.views.MethodView`

Render the GET `/budgets` view using the `budgets.html` template.

**get** ()

**methods** = {'GET'}

**class** `biweeklybudget.flaskapp.views.budgets.OneBudgetView`

Bases: `flask.views.MethodView`

Render the GET `/budgets/<int:budget_id>` view using the `budgets.html` template.

**get** (*budget\_id*)

**methods** = {'GET'}

## biweeklybudget.flaskapp.views.credit\_payoffs module

**class** `biweeklybudget.flaskapp.views.credit_payoffs.AccountOfxAjax`

Bases: `flask.views.MethodView`

Handle GET `/ajax/account_ofx_ajax/<int:account_id>` endpoint.

**get** (*account\_id*)

**methods** = {'GET'}

**class** `biweeklybudget.flaskapp.views.credit_payoffs.AccountOfxFormHandler`

Bases: `biweeklybudget.flaskapp.views.formhandlerview.FormHandlerView`

Handle POST `/forms/credit-payoff-account-ofx`

**methods** = {'POST'}

**submit** (*data*)

Handle form submission; create or update models in the DB. Raises an Exception for any errors.

**Parameters** *data* (*dict*) – submitted form data

**Returns** message describing changes to DB (i.e. link to created record)

**Return type** `str`

**validate** (*data*)

Validate the form data. Return None if it is valid, or else a hash of field names to list of error strings for each field.

**Parameters** *data* (*dict*) – submitted form data

**Returns** None if no errors, or hash of field name to errors for that field

**class** `biweeklybudget.flaskapp.views.credit_payoffs.CreditPayoffsView`

Bases: `flask.views.MethodView`

Render the top-level GET `/accounts/credit-payoff` view using `credit-payoffs.html` template.

`_payment_settings_dict` (*settings\_json*)

Given the JSON string payment settings, return a dict of payment settings as expected by *InterestHelper* kwargs.

**Parameters** `settings_json` (*str*) – payment settings JSON

**Returns** payment settings dict

**Return type** `dict`

`_payoffs_list` (*ih*)

Return a payoffs list suitable for rendering.

**Parameters** `ih` (*biweeklybudget.interest.InterestHelper*) – interest helper instance

**Returns** list of payoffs suitable for rendering

**Return type** `list`

`get` ()

`methods` = {'GET'}

**class** `biweeklybudget.flaskapp.views.credit_payoffs.PayoffSettingsFormHandler`

Bases: `flask.views.MethodView`

Handle POST /settings/credit-payoff

`methods` = {'POST'}

`post` ()

Handle form submission; create or update models in the DB. Raises an Exception for any errors.

**Returns** message describing changes to DB (i.e. link to created record)

**Return type** `str`

## biweeklybudget.flaskapp.views.example module

**class** `biweeklybudget.flaskapp.views.example.ExampleView`

Bases: `flask.views.MethodView`

`get` ()

`methods` = {'GET'}

## biweeklybudget.flaskapp.views.formhandlerview module

**class** `biweeklybudget.flaskapp.views.formhandlerview.FormHandlerView`

Bases: `flask.views.MethodView`

`_validate_date_ymd` (*key*, *data*, *errors*)

Validate a YYYY-mm-dd date field.

**Parameters**

- **key** (*str*) – the key in data to look at
- **data** (*dict*) – the form data
- **err\_list** (*dict*) – list of error messages for the field

**Returns** updated err\_list

**Return type** dict

**`_validate_decimal`** (*key, data, errors*)

Validate a Decimal field.

**Parameters**

- **key** (*str*) – the key in data to look at
- **data** (*dict*) – the form data
- **err\_list** (*dict*) – list of error messages for the field

**Returns** updated err\_list

**Return type** dict

**`_validate_float`** (*key, data, errors*)

Validate a float field.

**Parameters**

- **key** (*str*) – the key in data to look at
- **data** (*dict*) – the form data
- **err\_list** (*dict*) – list of error messages for the field

**Returns** updated err\_list

**Return type** dict

**`_validate_int`** (*key, data, errors*)

Validate an integer field.

**Parameters**

- **key** (*str*) – the key in data to look at
- **data** (*dict*) – the form data
- **err\_list** (*dict*) – list of error messages for the field

**Returns** updated err\_list

**Return type** dict

**`_validate_not_empty`** (*key, data, errors*)

Validate that a string is not empty.

**Parameters**

- **key** (*str*) – the key in data to look at
- **data** (*dict*) – the form data
- **err\_list** (*dict*) – list of error messages for the field

**Returns** updated err\_list

**Return type** dict

**`fix_string`** (*s*)

Strip a string. If the result is empty, return None. Otherwise return the result.

**Parameters** **s** (*str*) – form data value

**Returns** stripped string or None

```
methods = {'POST'}
```

```
post ()
```

Handle a POST request for a form. Validate it, if valid update the DB.

Returns a JSON hash with the following structure:

'errors' -> hash of field names to list of error strings 'error\_message' -> string error message 'success' -> boolean 'success\_message' -> string success message

```
submit (data)
```

Handle form submission; create or update models in the DB.

**Parameters** `data` (*dict*) – submitted form data

**Returns** message describing changes to DB

**Return type** `str`

```
validate (data)
```

Validate the form data. Return None if it is valid, or else a hash of field names to list of error strings for each field.

**Parameters** `data` (*dict*) – submitted form data

**Returns** None if no errors, or hash of field name to errors for that field

## biweeklybudget.flaskapp.views.fuel module

```
class biweeklybudget.flaskapp.views.fuel.FuelAjax
```

Bases: *biweeklybudget.flaskapp.views.searchableajaxview.SearchableAjaxView*

Handle GET /ajax/fuelLog endpoint.

```
__filterhack (qs, s, args)
```

DataTables 1.10.12 has built-in support for filtering based on a value in a specific column; when this is done, the filter value is set in `columns[N][search][value]` where N is the column number. However, the python datatables package used here only supports the global `search[value]` input, not the per-column one.

However, the DataTable search is implemented by passing a callable to `table.searchable()` which takes two arguments, the current Query that's being built, and the user's `search[value]` input; this must then return a Query object with the search applied.

In python datatables 0.4.9, this code path is triggered on `if callable(self.search_func) and search.get("value", None):`

As such, we can “trick” the table to use per-column searching (currently only if global searching is not being used) by examining the per-column search values in the request, and setting the search function to one (this method) that uses those values instead of the global `search[value]`.

**Parameters**

- `qs` (`sqlalchemy.orm.query.Query`) – Query currently being built
- `s` (*str*) – user search value
- `args` (*dict*) – args

**Returns** Query with searching applied

**Return type** `sqlalchemy.orm.query.Query`

```

get ()
    Render and return JSON response for GET /ajax/ofx

methods = {'GET'}

class biweeklybudget.flaskapp.views.fuel.FuelLogFormHandler
    Bases: biweeklybudget.flaskapp.views.formhandlerview.FormHandlerView

    Handle POST /forms/fuel

methods = {'POST'}

submit (data)
    Handle form submission; create or update models in the DB. Raises an Exception for any errors.

    Parameters data (dict) – submitted form data

    Returns message describing changes to DB (i.e. link to created record)

    Return type str

validate (data)
    Validate the form data. Return None if it is valid, or else a hash of field names to list of error strings for
    each field.

    Parameters data (dict) – submitted form data

    Returns None if no errors, or hash of field name to errors for that field

class biweeklybudget.flaskapp.views.fuel.FuelMPGChartView
    Bases: flask.views.MethodView

    Handle GET /ajax/chart-data/fuel-economy endpoint.

get ()

methods = {'GET'}

class biweeklybudget.flaskapp.views.fuel.FuelPriceChartView
    Bases: flask.views.MethodView

    Handle GET /ajax/chart-data/fuel-prices endpoint.

get ()

methods = {'GET'}

class biweeklybudget.flaskapp.views.fuel.FuelView
    Bases: flask.views.MethodView

    Render the GET /fuel view using the fuel.html template.

get ()

methods = {'GET'}

class biweeklybudget.flaskapp.views.fuel.VehicleAjax
    Bases: flask.views.MethodView

    Handle GET /ajax/vehicle/<int:vehicle_id> endpoint.

get (vehicle_id)

methods = {'GET'}

class biweeklybudget.flaskapp.views.fuel.VehicleFormHandler
    Bases: biweeklybudget.flaskapp.views.formhandlerview.FormHandlerView

```

Handle POST /forms/vehicle

**methods** = {'POST'}

**submit** (*data*)

Handle form submission; create or update models in the DB. Raises an Exception for any errors.

**Parameters** *data* (*dict*) – submitted form data

**Returns** message describing changes to DB (i.e. link to created record)

**Return type** *str*

**validate** (*data*)

Validate the form data. Return None if it is valid, or else a hash of field names to list of error strings for each field.

**Parameters** *data* (*dict*) – submitted form data

**Returns** None if no errors, or hash of field name to errors for that field

### biweeklybudget.flaskapp.views.help module

**class** biweeklybudget.flaskapp.views.help.**HelpView**

Bases: flask.views.MethodView

Render the GET /help view using the help.html template.

**get** ()

**methods** = {'GET'}

### biweeklybudget.flaskapp.views.index module

**class** biweeklybudget.flaskapp.views.index.**AcctBalanaceChartView**

Bases: flask.views.MethodView

Handle GET /ajax/chart-data/account-balances endpoint.

**get** ()

**methods** = {'GET'}

**class** biweeklybudget.flaskapp.views.index.**IndexView**

Bases: flask.views.MethodView

Render the GET /view using the index.html template.

**get** ()

**methods** = {'GET'}

### biweeklybudget.flaskapp.views.ofx module

**class** biweeklybudget.flaskapp.views.ofx.**OfxAccounts**

Bases: flask.views.MethodView

Handle GET /api/ofx/accounts endpoint.

This returns the JSON-ified return value from `get_accounts()` and will usually be called from `get_accounts()`.

```
get ()
```

```
methods = {'GET'}
```

```
class biweeklybudget.flaskapp.views.ofx.OfxAjax
```

Bases: *biweeklybudget.flaskapp.views.searchableajaxview.SearchableAjaxView*

Handle GET /ajax/ofx endpoint.

```
_filterhack (qs, s, args)
```

DataTables 1.10.12 has built-in support for filtering based on a value in a specific column; when this is done, the filter value is set in `columns[N][search][value]` where N is the column number. However, the python datatables package used here only supports the global `search[value]` input, not the per-column one.

However, the DataTable search is implemented by passing a callable to `table.searchable()` which takes two arguments, the current Query that's being built, and the user's `search[value]` input; this must then return a Query object with the search applied.

In python datatables 0.4.9, this code path is triggered on `if callable(self.search_func) and search.get("value", None):`

As such, we can “trick” the table to use per-column searching (currently only if global searching is not being used) by examining the per-column search values in the request, and setting the search function to one (this method) that uses those values instead of the global `search[value]`.

#### Parameters

- `qs` (`sqlalchemy.orm.query.Query`) – Query currently being built
- `s` (`str`) – user search value
- `args` (`dict`) – args

**Returns** Query with searching applied

**Return type** `sqlalchemy.orm.query.Query`

```
get ()
```

Render and return JSON response for GET /ajax/ofx

```
methods = {'GET'}
```

```
class biweeklybudget.flaskapp.views.ofx.OfxStatementPost
```

Bases: `flask.views.MethodView`

Handle POST /api/ofx/statement endpoint.

This is a ReST API bridge between `update_statement_ofx()` on the client side and `update_statement_ofx()` on the server side.

```
methods = {'POST'}
```

```
post ()
```

Handle POST to /api/ofx/statement (from `update_statement_ofx()`) to upload a new OFX Statement (via `update_statement_ofx()`).

The POSTed JSON should have the following keys:

- `acct_id` (int) the Account ID the Statement is for
- `mtime` (str) base64-encoded, pickled representation of the file modification time of the OFX file
- `filename` (str) the file name of the OFX file

- `ofx` (str) base64-encoded, pickled representation of the `ofxparse.ofxparse.Ofx` instance representing the Statement

Returns a JSON object with the following fields:

- `success` (bool) whether the operation was successful
- `message` (str) message describing success or error message

For successful operations, the JSON object will contain the following additional fields:

- `count_new` (int) count of new transactions added
- `count_updated` (int) count of transactions updated
- `statement_id` (int) ID of the newly-added statement

HTTP Status Codes:

- 201 - Statement successfully added
- 500 - DuplicateFileException
- 400 - Any other error/exception

```
class biweeklybudget.flaskapp.views.ofx.OfxTransAjax
```

```
    Bases: flask.views.MethodView
```

```
    Handle GET /ajax/ofx/<int:acct_id>/<str:fitid> endpoint.
```

```
    get (acct_id, fitid)
```

```
    methods = {'GET'}
```

```
class biweeklybudget.flaskapp.views.ofx.OfxTransView
```

```
    Bases: flask.views.MethodView
```

```
    Render the GET /ofx/<int:acct_id>/<str:fitid> view using the ofx.html template.
```

```
    get (acct_id, fitid)
```

```
    methods = {'GET'}
```

```
class biweeklybudget.flaskapp.views.ofx.OfxView
```

```
    Bases: flask.views.MethodView
```

```
    Render the GET /ofx view using the ofx.html template.
```

```
    get ()
```

```
    methods = {'GET'}
```

## **biweeklybudget.flaskapp.views.payperiods module**

```
class biweeklybudget.flaskapp.views.payperiods.PayPeriodView
```

```
    Bases: flask.views.MethodView
```

```
    Render the single PayPeriod GET /payperiod/YYYY-MM-DD view using the payperiod.html template.
```

```
    get (period_date)
```

```
    methods = {'GET'}
```

```
    suffix_for_period (curr_pp, pp)
```

```
        Generate the suffix to use for the given pay period in the view
```

**Parameters**

- `curr_pp` (`BiweeklyPayPeriod`) – the current (today) pay period
- `pp` (`BiweeklyPayPeriod`) – the pay period in question

**Returns** suffix for the pay period

**Return type** `str`

**class** `biweeklybudget.flaskapp.views.payperiods.PayPeriodsView`

Bases: `flask.views.MethodView`

Render the top-level GET `/payperiods` view using `payperiods.html` template

`get()`

`methods = {'GET'}`

**class** `biweeklybudget.flaskapp.views.payperiods.PeriodForDateView`

Bases: `flask.views.MethodView`

Render a redirect from a given date to the pay period for that date

`get()`

`methods = {'GET'}`

**class** `biweeklybudget.flaskapp.views.payperiods.SchedToTransFormHandler`

Bases: `biweeklybudget.flaskapp.views.formhandlerview.FormHandlerView`

Handle POST `/forms/sched_to_trans`

`methods = {'POST'}`

`submit(data)`

Handle form submission; create or update models in the DB. Raises an Exception for any errors.

**Parameters** `data` (`dict`) – submitted form data

**Returns** message describing changes to DB (i.e. link to created record)

**Return type** `str`

`validate(data)`

Validate the form data. Return None if it is valid, or else a hash of field names to list of error strings for each field.

**Parameters** `data` (`dict`) – submitted form data

**Returns** None if no errors, or hash of field name to errors for that field

**class** `biweeklybudget.flaskapp.views.payperiods.SkipSchedTransFormHandler`

Bases: `biweeklybudget.flaskapp.views.formhandlerview.FormHandlerView`

Handle POST `/forms/skip_sched_trans`

`methods = {'POST'}`

`submit(data)`

Handle form submission; create or update models in the DB. Raises an Exception for any errors.

**Parameters** `data` (`dict`) – submitted form data

**Returns** message describing changes to DB (i.e. link to created record)

**Return type** `str`

**validate** (*data*)

Validate the form data. Return None if it is valid, or else a hash of field names to list of error strings for each field.

**Parameters** *data* (*dict*) – submitted form data

**Returns** None if no errors, or hash of field name to errors for that field

## **biweeklybudget.flaskapp.views.plaid module**

**class** `biweeklybudget.flaskapp.views.plaid.PlaidHandleLink`

Bases: `flask.views.MethodView`

Handle POST `/ajax/plaid/handle_link` endpoint.

**methods** = {'POST'}

**post** ()

**class** `biweeklybudget.flaskapp.views.plaid.PlaidJs`

Bases: `flask.views.MethodView`

Handle GET `/plaid.js` endpoint, for CI/test or production/real.

**get** ()

**methods** = {'GET'}

**class** `biweeklybudget.flaskapp.views.plaid.PlaidLinkToken`

Bases: `flask.views.MethodView`

Handle POST `/ajax/plaid/create_link_token` endpoint.

**methods** = {'POST'}

**post** ()

**class** `biweeklybudget.flaskapp.views.plaid.PlaidRefreshAccounts`

Bases: `flask.views.MethodView`

Handle POST `/ajax/plaid/refresh_item_accounts` endpoint.

**methods** = {'POST'}

**post** ()

**class** `biweeklybudget.flaskapp.views.plaid.PlaidUpdate`

Bases: `flask.views.MethodView`

Handle GET or POST `/plaid-update`

This single endpoint has multiple functions:

- If GET with no query parameters, displays a form template to use to interactively update Plaid accounts.
- If GET or POST with an `item_ids` query parameter, performs a Plaid update (via `_update()`) of the specified CSV list of Plaid Item IDs, or all Plaid Items if the value is `ALL`. The POST method also accepts an optional `num_days` parameter specifying an integer number of days of transactions to update. The response from this endpoint can be in one of three forms:
  - If the Accept HTTP header is set to `application/json`, return a JSON list of update results. Each list item is the JSON-ified value of `as_dict`.
  - If the Accept HTTP header is set to `text/plain`, return a plain text human-readable summary of the update operation.

- Otherwise, return a templated view of the update operation results, as would be returned to a browser.

`_form()`

`_update(ids: str, num_days: int = 30)`

Handle an update for Plaid accounts by instantiating a `PlaidUpdater`, calling its `update()` method with the proper arguments, and then returning the result in a form determined by the `Accept` header.

#### Parameters

- **ids** (*str*) – a comma-separated string listing the `PlaidItem` IDs to update, or the string `ALL` to update all Items.
- **num\_days** (*int*) – number of days to retrieve transactions for; default 30

`get()`

Handle GET. If the `item_ids` query parameter is set, then return `_update()`, else return `_form()`.

`methods = {'GET', 'POST'}`

`post()`

Handle POST. If the `item_ids` query parameter is set, then return `_update()`, else return a HTTP 400. If the optional `num_days` query parameter is set, pass that on to the update method.

**class** `biweeklybudget.flaskapp.views.plaid.PlaidUpdateItemInfo`

Bases: `flask.views.MethodView`

Handle POST `/ajax/plaid/update_item_info` endpoint.

`methods = {'POST'}`

`post()`

`biweeklybudget.flaskapp.views.plaid.set_url_rules(a)`

## biweeklybudget.flaskapp.views.projects module

**class** `biweeklybudget.flaskapp.views.projects.BoMItemAjax`

Bases: `flask.views.MethodView`

Render the GET `/ajax/projects/bom_item/<int:id>` JSON view.

`get(id)`

`methods = {'GET'}`

**class** `biweeklybudget.flaskapp.views.projects.BoMItemFormHandler`

Bases: `biweeklybudget.flaskapp.views.formhandlerview.FormHandlerView`

Handle POST `/forms/bom_item`

`methods = {'POST'}`

`submit(data)`

Handle form submission; create or update models in the DB. Raises an Exception for any errors.

**Parameters** `data` (*dict*) – submitted form data

**Returns** message describing changes to DB (i.e. link to created record)

**Return type** `str`

`validate(data)`

Validate the form data. Return `None` if it is valid, or else a hash of field names to list of error strings for each field.

**Parameters** `data` (*dict*) – submitted form data

**Returns** None if no errors, or hash of field name to errors for that field

**class** `biweeklybudget.flaskapp.views.projects.BoMItemView`

Bases: `flask.views.MethodView`

Render the GET `/project/<int:project_id>` view using the `bomitem.html` template.

**get** (*project\_id*)

**methods** = {'GET'}

**class** `biweeklybudget.flaskapp.views.projects.BoMItemsAjax`

Bases: `biweeklybudget.flaskapp.views.searchableajaxview.SearchableAjaxView`

Handle GET `/ajax/projects/<int:project_id>/bom_items` endpoint.

**`_filterhack`** (*qs, s, args*)

DataTables 1.10.12 has built-in support for filtering based on a value in a specific column; when this is done, the filter value is set in `columns[N][search][value]` where N is the column number. However, the python datatables package used here only supports the global `search[value]` input, not the per-column one.

However, the DataTable search is implemented by passing a callable to `table.searchable()` which takes two arguments, the current Query that's being built, and the user's `search[value]` input; this must then return a Query object with the search applied.

In python datatables 0.4.9, this code path is triggered on `if callable(self.search_func) and search.get("value", None):`

As such, we can “trick” the table to use per-column searching (currently only if global searching is not being used) by examining the per-column search values in the request, and setting the search function to one (this method) that uses those values instead of the global `search[value]`.

**Parameters**

- `qs` (`sqlalchemy.orm.query.Query`) – Query currently being built
- `s` (*str*) – user search value
- `args` (*dict*) – args

**Returns** Query with searching applied

**Return type** `sqlalchemy.orm.query.Query`

**get** (*project\_id*)

Render and return JSON response for GET `/ajax/projects/<int:project_id>/bom_items`

**methods** = {'GET'}

**class** `biweeklybudget.flaskapp.views.projects.ProjectAjax`

Bases: `flask.views.MethodView`

Render the GET `/ajax/projects/<int:project_id>` JSON view.

**get** (*project\_id*)

**methods** = {'GET'}

**class** `biweeklybudget.flaskapp.views.projects.ProjectsAjax`

Bases: `biweeklybudget.flaskapp.views.searchableajaxview.SearchableAjaxView`

Handle GET `/ajax/projects` endpoint.

**\_filterhack** (*qs, s, args*)

DataTables 1.10.12 has built-in support for filtering based on a value in a specific column; when this is done, the filter value is set in `columns[N][search][value]` where N is the column number. However, the python datatables package used here only supports the global `search[value]` input, not the per-column one.

However, the DataTable search is implemented by passing a callable to `table.searchable()` which takes two arguments, the current Query that's being built, and the user's `search[value]` input; this must then return a Query object with the search applied.

In python datatables 0.4.9, this code path is triggered on `if callable(self.search_func) and search.get("value", None):`

As such, we can “trick” the table to use per-column searching (currently only if global searching is not being used) by examining the per-column search values in the request, and setting the search function to one (this method) that uses those values instead of the global `search[value]`.

**Parameters**

- **qs** (`sqlalchemy.orm.query.Query`) – Query currently being built
- **s** (*str*) – user search value
- **args** (*dict*) – args

**Returns** Query with searching applied

**Return type** `sqlalchemy.orm.query.Query`

**get** ()

Render and return JSON response for GET /ajax/projects

**methods** = {'GET'}

**class** `biweeklybudget.flaskapp.views.projects.ProjectsFormHandler`

Bases: `biweeklybudget.flaskapp.views.formhandlerview.FormHandlerView`

Handle POST /forms/projects

**methods** = {'POST'}

**submit** (*data*)

Handle form submission; create or update models in the DB. Raises an Exception for any errors.

**Parameters** **data** (*dict*) – submitted form data

**Returns** message describing changes to DB (i.e. link to created record)

**Return type** *str*

**validate** (*data*)

Validate the form data. Return None if it is valid, or else a hash of field names to list of error strings for each field.

**Parameters** **data** (*dict*) – submitted form data

**Returns** None if no errors, or hash of field name to errors for that field

**class** `biweeklybudget.flaskapp.views.projects.ProjectsView`

Bases: `flask.views.MethodView`

Render the GET /projects view using the `projects.html` template.

**get** ()

**methods** = {'GET'}

## biweeklybudget.flaskapp.views.reconcile module

**class** biweeklybudget.flaskapp.views.reconcile.**OfxUnreconciledAjax**

Bases: flask.views.MethodView

Handle GET /ajax/unreconciled/ofx endpoint.

**get** ()

**methods** = {'GET'}

**class** biweeklybudget.flaskapp.views.reconcile.**ReconcileAjax**

Bases: flask.views.MethodView

Handle POST /ajax/reconcile endpoint.

**methods** = {'POST'}

**post** ()

Handle POST /ajax/reconcile

Request is a JSON dict with two keys, “reconciled” and “ofxIgnored”. “reconciled” value is a dict of integer transaction ID keys, to values which are either a string reason why the Transaction is being reconciled as “No OFX” or a 2-item list of OFXTransaction acct\_id and fitid. “ofxIgnored” is a dict with string keys which are strings identifying an OFXTransaction in the form “<ACCT\_ID>%<FITID>”, and values are a string reason why the OFXTransaction is being reconciled without a matching Transaction.

Response is a JSON dict. Keys are success (boolean) and either error\_message (string) or success\_message (string).

**Returns** JSON response

**class** biweeklybudget.flaskapp.views.reconcile.**ReconcileView**

Bases: flask.views.MethodView

Render the top-level GET /reconcile view using reconcile.html template.

**get** ()

**methods** = {'GET'}

**class** biweeklybudget.flaskapp.views.reconcile.**TransUnreconciledAjax**

Bases: flask.views.MethodView

Handle GET /ajax/unreconciled/trans endpoint.

**get** ()

**methods** = {'GET'}

**class** biweeklybudget.flaskapp.views.reconcile.**TxnReconcileAjax**

Bases: flask.views.MethodView

Handle GET /ajax/reconcile/<int:reconcile\_id> endpoint.

**get** (*reconcile\_id*)

**methods** = {'GET'}

## biweeklybudget.flaskapp.views.scheduled module

**class** biweeklybudget.flaskapp.views.scheduled.**OneScheduledAjax**

Bases: flask.views.MethodView

Handle GET /ajax/scheduled/<int:sched\_trans\_id> endpoint.

**get** (*sched\_trans\_id*)

**methods** = {'GET'}

**class** `biweeklybudget.flaskapp.views.scheduled.SchedTransFormHandler`

Bases: `biweeklybudget.flaskapp.views.formhandlerview.FormHandlerView`

Handle POST /forms/scheduled

**methods** = {'POST'}

**submit** (*data*)

Handle form submission; create or update models in the DB. Raises an Exception for any errors.

**Parameters** *data* (*dict*) – submitted form data

**Returns** message describing changes to DB (i.e. link to created record)

**Return type** `str`

**validate** (*data*)

Validate the form data. Return None if it is valid, or else a hash of field names to list of error strings for each field.

**Parameters** *data* (*dict*) – submitted form data

**Returns** None if no errors, or hash of field name to errors for that field

**class** `biweeklybudget.flaskapp.views.scheduled.ScheduledAjax`

Bases: `biweeklybudget.flaskapp.views.searchableajaxview.SearchableAjaxView`

Handle GET /ajax/scheduled endpoint.

**`_filterhack`** (*qs, s, args*)

DataTables 1.10.12 has built-in support for filtering based on a value in a specific column; when this is done, the filter value is set in `columns[N][search][value]` where N is the column number. However, the python datatables package used here only supports the global `search[value]` input, not the per-column one.

However, the DataTable search is implemented by passing a callable to `table.searchable()` which takes two arguments, the current Query that's being built, and the user's `search[value]` input; this must then return a Query object with the search applied.

In python datatables 0.4.9, this code path is triggered on `if callable(self.search_func) and search.get("value", None):`

As such, we can “trick” the table to use per-column searching (currently only if global searching is not being used) by examining the per-column search values in the request, and setting the search function to one (this method) that uses those values instead of the global `search[value]`.

**Parameters**

- **qs** (`sqlalchemy.orm.query.Query`) – Query currently being built
- **s** (*str*) – user search value
- **args** (*dict*) – args

**Returns** Query with searching applied

**Return type** `sqlalchemy.orm.query.Query`

**get** ()

Render and return JSON response for GET /ajax/ofx

```
methods = {'GET'}
```

```
class biweeklybudget.flaskapp.views.scheduled.ScheduledTransView
```

```
    Bases: flask.views.MethodView
```

```
    get (sched_trans_id)
```

```
        Render the GET /scheduled/<int:sched_trans_id> view using the scheduled.html template.
```

```
    methods = {'GET'}
```

```
class biweeklybudget.flaskapp.views.scheduled.ScheduledView
```

```
    Bases: flask.views.MethodView
```

```
    get ()
```

```
        Render the GET /scheduled view using the scheduled.html template.
```

```
    methods = {'GET'}
```

### **biweeklybudget.flaskapp.views.searchableajaxview module**

```
class biweeklybudget.flaskapp.views.searchableajaxview.SearchableAjaxView
```

```
    Bases: flask.views.MethodView
```

```
    MethodView with helper methods for searching via DataTables ajax.
```

```
    _args_dict (args)
```

```
        Given a 1-dimensional dict of request parameters like those used by DataTables (i.e. keys like columns[2][search][value]), return a multidimensional dict representation of the same.
```

```
        Returns deep/nested dict
```

```
        Return type dict
```

```
    _args_set_type (a)
```

```
        Given a string portion of something in the argument dict, return it as the correct type.
```

```
        Parameters a (str) – args dict key or value
```

```
        Returns a in the proper type
```

```
    _filterhack (qs, s, args)
```

```
        DataTables 1.10.12 has built-in support for filtering based on a value in a specific column; when this is done, the filter value is set in columns[N][search][value] where N is the column number. However, the python datatables package used here only supports the global search[value] input, not the per-column one.
```

```
        However, the DataTable search is implemented by passing a callable to table.searchable() which takes two arguments, the current Query that's being built, and the user's search[value] input; this must then return a Query object with the search applied.
```

```
        In python datatables 0.4.9, this code path is triggered on if callable(self.search_func) and search.get("value", None):
```

```
        As such, we can “trick” the table to use per-column searching (currently only if global searching is not being used) by examining the per-column search values in the request, and setting the search function to one (this method) that uses those values instead of the global search[value].
```

```
        Parameters
```

- **qs** (sqlalchemy.orm.query.Query) – Query currently being built
- **s** (str) – user search value

- **args** (*dict*) – args

**Returns** Query with searching applied

**Return type** `sqlalchemy.orm.query.Query`

**`__have_column_search`** (*args*)

Determine if we have a column filter/search in effect, and if so, should use `__filterhack()` as our search function.

**Parameters** **args** (*dict*) – current request arguments

**Returns** whether or not request asks for column filtering

**Return type** `bool`

**get** ()

Render and return JSON response for GET.

**methods** = {'GET'}

### biweeklybudget.flaskapp.views.transactions module

**class** `biweeklybudget.flaskapp.views.transactions.OneTransactionAjax`

Bases: `flask.views.MethodView`

Handle GET /ajax/transactions/<int:trans\_id> endpoint.

**get** (*trans\_id*)

**methods** = {'GET'}

**class** `biweeklybudget.flaskapp.views.transactions.OneTransactionView`

Bases: `flask.views.MethodView`

**get** (*trans\_id*)

Render the GET /transactions/<int:trans\_id> view using the `transactions.html` template.

**methods** = {'GET'}

**class** `biweeklybudget.flaskapp.views.transactions.TransactionFormHandler`

Bases: `biweeklybudget.flaskapp.views.formhandlerview.FormHandlerView`

Handle POST /forms/transaction

**methods** = {'POST'}

**submit** (*data*)

Handle form submission; create or update models in the DB. Raises an Exception for any errors.

**Parameters** **data** (*dict*) – submitted form data

**Returns** message describing changes to DB (i.e. link to created record)

**Return type** `str`

**validate** (*data*)

Validate the form data. Return None if it is valid, or else a hash of field names to list of error strings for each field.

**Parameters** **data** (*dict*) – submitted form data

**Returns** None if no errors, or hash of field name to errors for that field

**class** `biweeklybudget.flaskapp.views.transactions.TransactionsAjax`

Bases: `biweeklybudget.flaskapp.views.searchableajaxview.SearchableAjaxView`

Handle GET /ajax/transactions endpoint.

**`_filterhack`** (*qs, s, args*)

DataTables 1.10.12 has built-in support for filtering based on a value in a specific column; when this is done, the filter value is set in `columns[N][search][value]` where N is the column number. However, the python datatables package used here only supports the global `search[value]` input, not the per-column one.

However, the DataTable search is implemented by passing a callable to `table.searchable()` which takes two arguments, the current Query that's being built, and the user's `search[value]` input; this must then return a Query object with the search applied.

In python datatables 0.4.9, this code path is triggered on `if callable(self.search_func) and search.get("value", None):`

As such, we can “trick” the table to use per-column searching (currently only if global searching is not being used) by examining the per-column search values in the request, and setting the search function to one (this method) that uses those values instead of the global `search[value]`.

#### Parameters

- **`qs`** (`sqlalchemy.orm.query.Query`) – Query currently being built
- **`s`** (*str*) – user search value
- **`args`** (*dict*) – args

**Returns** Query with searching applied

**Return type** `sqlalchemy.orm.query.Query`

**`get`** ()

Render and return JSON response for GET /ajax/ofx

**`methods`** = {'GET'}

**class** `biweeklybudget.flaskapp.views.transactions.TransactionsView`

Bases: `flask.views.MethodView`

**`get`** ()

Render the GET /transactions view using the `transactions.html` template.

**`methods`** = {'GET'}

## **biweeklybudget.flaskapp.views.utils module**

**class** `biweeklybudget.flaskapp.views.utils.DateTestJS`

Bases: `flask.views.MethodView`

Handle GET /utils/datetest.js endpoint.

**`get`** ()

**`methods`** = {'GET'}

`biweeklybudget.flaskapp.views.utils.set_url_rules` (*a*)

## Submodules

### biweeklybudget.flaskapp.app module

`biweeklybudget.flaskapp.app.before_request()`

When running in debug mode, clear jinja cache.

`biweeklybudget.flaskapp.app.shutdown_session(exception=None)`

### biweeklybudget.flaskapp.cli\_commands module

**class** `biweeklybudget.flaskapp.cli_commands.CustomLoggingWSGIRequestHandler` (*request, client\_address, server*)

Bases: `werkzeug.serving.WSGIRequestHandler`

Extend werkzeug request handler to include processing time in logs

**handle** ()

Handles a request ignoring dropped connections.

**log\_request** (*code='-', size='-'*)

Log an accepted request.

This is called by `send_response()`.

**send\_response** (*\*args, \*\*kw*)

Send the response header and log the response code.

`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.context_processors.add_currency_symbol()`

Context processor to inject the proper currency symbol into the Jinja2 context as the “CURRENCY\_SYM” variable.

**Returns** proper currency symbol for our locale and currency

**Return type** `str`

`biweeklybudget.flaskapp.context_processors.notifications()`

Add notifications to template context for all templates.

**Returns** template context with notifications added

**Return type** `dict`

`biweeklybudget.flaskapp.context_processors.settings()`

Add settings to template context for all templates.

**Returns** template context with settings added

**Return type** `dict`

## biweeklybudget.flaskapp.filters module

`biweeklybudget.flaskapp.filters.acct_icon_filter(acct)`

Given an Account, return the proper classes for an account type icon for it.

**Parameters** `acct` (`biweeklybudget.models.account.Account`) – the account

**Returns** string icon classes

**Return type** `str`

`biweeklybudget.flaskapp.filters.ago_filter(dt)`

Format a datetime using `humanize.naturaltime`, “ago”

**Parameters** `dt` (`datetime.datetime`) – datetime to compare to now

**Returns** ago string

**Return type** `str`

`biweeklybudget.flaskapp.filters.budget_cell_filter(d)`

Given a dictionary of budget IDs to names and amounts like that returned by `_dict_for_trans()`, return the `<td>` content for those budgets.

`biweeklybudget.flaskapp.filters.dateymd_filter(dt)`

Format a datetime using `%Y-%m-%d`

**Parameters** `dt` (`datetime.datetime`) – datetime to format

**Returns** formatted date

**Return type** `str`

`biweeklybudget.flaskapp.filters.decimal_to_percent(d)`

`biweeklybudget.flaskapp.filters.dict_to_class_args(j)`

`biweeklybudget.flaskapp.filters.dollars_filter(x)`

Format as currency using `fmt_currency()`.

**Parameters** `x` – currency amount, int, float, decimal, etc.

**Returns** formatted currency

**Return type** `str`

`biweeklybudget.flaskapp.filters.isodate_filter(dt)`

Format a datetime using `%Y-%m-%d %H:%M:%S`

**Parameters** `dt` (`datetime.datetime`) – datetime to format

**Returns** formatted date

**Return type** `str`

`biweeklybudget.flaskapp.filters.monthsyears(num)`

`biweeklybudget.flaskapp.filters.period_panel_color_filter(x)`

Given the remaining amount for a pay period, return “red” if less than zero, “yellow” if less than 100, or otherwise “green”.

**Parameters** `x` (`float`) – PayPeriod remaining amount

**Returns** “red”, “yellow” or “green”

**Return type** `str`

`biweeklybudget.flaskapp.filters.pluralize_filter` (*word*, *number=1*)

If *number* is greater than one, return *word* with an “s” appended, else return *word* unmodified.

**Parameters**

- **word** (*string*) – the word to pluralize or not
- **number** (*int*) – the number to check for greater-than-one-ness

**Returns** *word*, pluralized or not

**Return type** `str`

`biweeklybudget.flaskapp.filters.reddollars_filter` (*x*)

Return a string similar to `dollars_filter` but in red text if negative.

**Parameters** **x** – dollar amount, int, float, decimal, etc.

**Returns** formatted currency

**Return type** `str`

## biweeklybudget.flaskapp.jinja\_tests module

`biweeklybudget.flaskapp.jinja_tests.is_stale_data` (*dt*)

Given a datetime object, return True if we consider it to be “stale” data, False otherwise.

**Parameters** **dt** (*datetime*) – datetime for age of the data

**Returns** True if data is stale, False otherwise

**Return type** `bool`

## biweeklybudget.flaskapp.jsonencoder module

```
class biweeklybudget.flaskapp.jsonencoder.MagicJSONEncoder (*, skipkeys=False,
    ensure_ascii=True,
    check_circular=True,
    allow_nan=True,
    sort_keys=False,
    indent=None, sep-
    arators=None, de-
    fault=None)
```

Bases: `json.encoder.JSONEncoder`

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

**default** (*o*)

Implement this method in a subclass such that it returns a serializable object for *o*, or calls the base implementation (to raise a `TypeError`).

For example, to support arbitrary iterators, you could implement `default` like this:

```
def default(self, o):
    try:
        iterable = iter(o)
    except TypeError:
        pass
    else:
        return list(iterable)
```

(continues on next page)

(continued from previous page)

```
# Let the base class default method raise the TypeError
return JSONEncoder.default(self, o)
```

## biweeklybudget.flaskapp.notifications module

**class** biweeklybudget.flaskapp.notifications.**NotificationsController**

Bases: `object`

**static** `budget_account_sum` (*sess=None*)

Return the sum of current balances for all is\_budget\_source accounts.

**Returns** Combined balance of all budget source accounts

**Return type** `float`

**static** `budget_account_unreconciled` (*sess=None*)

Return the sum of unreconciled txns for all is\_budget\_source accounts.

**Returns** Combined unreconciled amount 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` (*sess=None*)

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` (*sess=None*)

Return the number of unreconciled OFXTransactions.

**Returns** number of unreconciled OFXTransactions

**Return type** `int`

**static** `pp_sum` (*sess=None*)

Return the overall allocated sum for the current payperiod minus the sum of all reconciled Transactions for the pay period.

**Returns** overall allocated sum for the current pay period minus the sum of all reconciled Transactions for the pay period.

**Return type** `float`

**static** `standing_budgets_sum` (*sess=None*)

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 = {'acct_type': <sqlalchemy.orm.attributes.InstrumentedAttribute ob

acct_type
    Type of account (Enum AcctType)

all_statements
    Relationship to all OFXStatement for this Account

apr
    Finance rate (APR) for credit accounts

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

effective_apr
    Return the effective APR for a credit account. If prime_rate_margin is not Null, return that added to
    the current US Prime Rate. Otherwise, return apr.

    Returns Effective account APR

    Return type decimal.Decimal

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

interest_class_name
    Name of the biweeklybudget.interest._InterestCalculation subclass used to calculate
    interest for this account.

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`

**last\_interest\_charge**

Return the amount of the last interest charge for this account. Raise an exception if one could not be identified.

**Returns** amount of last interest charge for this account

**Return type** `decimal.Decimal`

**min\_payment\_class\_name**

Name of the `biweeklybudget.interest._MinPaymentFormula` subclass used to calculate minimum payments for this account.

**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

**plaid\_account**

`PlaidAccount` this account is linked with

**plaid\_account\_id**

Plaid Token for this account

**plaid\_configured**

Return whether or not this account is configured for Plaid.

**Returns** whether or not this account is configured for Plaid.

**Return type** `bool`

**plaid\_item\_id**

Plaid Item ID for this account

**prime\_rate\_margin**

Margin added to the US Prime Rate to determine APR, for credit accounts.

**re\_interest\_charge**

regex for matching transactions as interest charges

**re\_interest\_paid**

regex for matching transactions as interest paid

**re\_late\_fee**

regex for matching transactions as late fees

**re\_other\_fee**

regex for matching transactions as other fees

**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*

An enumeration.

**Bank** = 1

**Cash** = 4

**Credit** = 2

**Investment** = 3

**Other** = 5

`as_dict`

`transferrable_types = <bound method AcctType.transferrable_types of <enum 'AcctType'>>`

**exception** `biweeklybudget.models.account.NoInterestChargedError(acct)`

Bases: `Exception`

Exception raised when an *Account* does not have an OFXTransaction for interest charged within the last 32 days.

## biweeklybudget.models.account\_balance module

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

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

`_sa_class_manager = {'account': <sqlalchemy.orm.attributes.InstrumentedAttribute object>`

**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`

`_dict_properties = []`

Class properties to include in *as\_dict* result.

**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 = {'budget_transactions': <sqlalchemy.orm.attributes.InstrumentedAttribute object>}
```

```
budget_transactions
```

```
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
```

```
omit_from_graphs
```

```
whether or not to omit this budget from spending graphs
```

```
planned_transactions
```

```
scheduled_transactions
```

```
starting_balance
```

```
starting balance for periodic budgets
```

**biweeklybudget.models.budget\_transaction module**

```
class biweeklybudget.models.budget_transaction.BudgetTransaction (**kwargs)
```

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

Represents the portion (amount) of a Transaction that is allocated against a specific budget. There will be one or more BudgetTransactions associated with each *Transaction*.

```
_sa_class_manager = {'amount': <sqlalchemy.orm.attributes.InstrumentedAttribute object>}
```

```
amount
```

```
Amount of the transaction against this budget
```

```
budget
```

```
Relationship - the Budget this transaction is against
```

```
budget_id
```

```
ID of the Budget this transaction is against
```

**id**  
Primary Key

**trans\_id**  
ID of the Transaction this is part of

**transaction**  
Relationship - the *Transaction* this is part of

### biweeklybudget.models.dbsetting module

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

    _sa_class_manager = {'default_value': <sqlalchemy.orm.attributes.InstrumentedAttribut

    default_value
        Default value - usually JSON

    is_json
        Whether setting is JSON, or plain text

    name
        Primary Key

    value
        Setting value - usually JSON
```

### biweeklybudget.models.fuel module

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

    _previous_entry()
        Get the previous fill for this vehicle by odometer reading, or None.

        Returns the previous fill for this vehicle, by odometer reading, or None.

        Return type biweeklybudget.models.fuel.FuelFill

    _sa_class_manager = {'calculated_miles': <sqlalchemy.orm.attributes.InstrumentedAttri

    calculate_mpg()
        Calculate calculated_mpg field.

        Returns True if recalculate, False if unable to calculate

        Return type bool

    calculated_miles
        Number of miles actually traveled since the last fill.

    calculated_mpg
        Calculated MPG, based on last fill

    cost_per_gallon
        Fuel cost per gallon
```

**date**  
date of the fill

**fill\_location**  
Location of fill - usually a gas station name/address

**gallons**  
Total amount of fuel (gallons)

**id**  
Primary Key

**level\_after**  
Fuel level after fill, as a percentage (Integer 0-100)

**level\_before**  
Fuel level before fill, as a percentage (Integer 0-100)

**notes**  
Notes

**odometer\_miles**  
Odometer reading of the vehicle, in miles

**reported\_miles**  
Number of miles the vehicle thinks it's traveled since the last fill.

**reported\_mpg**  
MPG as reported by the vehicle itself

**total\_cost**  
Total cost of fill

**validate\_gallons** (\_, *value*)

**validate\_odometer\_miles** (\_, *value*)

**vehicle**  
The vehicle

**vehicle\_id**  
ID of the vehicle

```
class biweeklybudget.models.fuel.Vehicle(**kwargs)
Bases: sqlalchemy.ext.declarative.api.Base, biweeklybudget.models.base.ModelAsDict
_sa_class_manager = {'fuellog': <sqlalchemy.orm.attributes.InstrumentedAttribute object>
fuellog
id
    Primary Key
is_active
    whether active or historical
name
    Name of vehicle
```

## 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 = {'account': <sqlalchemy.orm.attributes.InstrumentedAttribute object>
  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
  ofx_trans
  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 = {'account': <sqlalchemy.orm.attributes.InstrumentedAttribute object>
  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
  first_statement_by_date
    Return the first OFXStatement on or after self.date_posted.
    Returns first OFXStatement on or after self.date_posted
    Return type biweeklybudget.models.ofx_statement.OFXStatement
  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**

**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

**update\_is\_fields** ()

Method to update all *is\_\** fields on this instance, given the *re\_\** properties of *account*.

## biweeklybudget.models.plaid\_accounts module

**class** biweeklybudget.models.plaid\_accounts.PlaidAccount (\*\*kwargs)

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

**\_sa\_class\_manager** = {'account': <sqlalchemy.orm.attributes.InstrumentedAttribute object>

**account**

**account\_id**  
Plaid Account ID

**account\_subtype**  
Plaid account subtype

**account\_type**  
Plaid account type

**item\_id**  
Plaid Item ID

**mask**  
mask

**name**  
Name of the account

**plaid\_item**  
PlaidItem this PlaidAccount is associated with

### biweeklybudget.models.plaid\_items module

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

    _sa_class_manager = {'access_token': <sqlalchemy.orm.attributes.InstrumentedAttribute
    access_token
        Plaid item access token

    all_accounts
        Relationship to all PlaidAccount for this Item

    institution_id
        institution ID

    institution_name
        institution name

    item_id
        Primary Key - plaid Item ID

    last_updated
        When this item was last updated
```

### biweeklybudget.models.projects module

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

    _sa_class_manager = {'id': <sqlalchemy.orm.attributes.InstrumentedAttribute object>,
    id
        Primary Key

    is_active
        whether active or historical
```

**line\_cost**

The total cost for this BoM Item, unit\_cost times quantity

**Returns** total line cost

**Return type** decimal.Decimal

**name**

Name of item

**notes**

Notes / Description

**project**

Relationship to the *Project* this item is for

**project\_id**

Project ID

**quantity**

Quantity Required

**unit\_cost**

Unit Cost / Cost Each

**url**

URL

**class** biweeklybudget.models.projects.**Project** (\*\*kwargs)

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

**\_sa\_class\_manager** = {'id': <sqlalchemy.orm.attributes.InstrumentedAttribute object>},

**id**

Primary Key

**is\_active**

whether active or historical

**name**

Name of project

**notes**

Notes / Description

**remaining\_cost**

Return the remaining cost of all line items (*BoMItem*) for this project which are still active

**Returns** remaining cost of this project

**Return type** float

**total\_cost**

Return the total cost of all line items (*BoMItem*) for this project.

**Returns** total cost of this project

**Return type** float

**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 = {'id': <sqlalchemy.orm.attributes.InstrumentedAttribute object>,
  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 = {'account': <sqlalchemy.orm.attributes.InstrumentedAttribute object>,
  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

**transactions**

**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*

Class that describes Transactions that have actually occurred, against one account and one or more budgets.

Note that in addition to the usual class attributes, the constructor of this class also accepts a budget\_amounts keyword argument, which passes its value on to *set\_budget\_amounts()*.

**\_dict\_properties** = ['actual\_amount']

Class properties to include in *as\_dict* result.

**\_sa\_class\_manager** = {'account': <sqlalchemy.orm.attributes.InstrumentedAttribute object>

**account**

Relationship - *Account* this transaction is against

**account\_id**

ID of the account this transaction is against

**actual\_amount**

Actual amount of the transaction.

**Returns** actual total amount of the transaction

**Return type** decimal.Decimal

**budget\_transactions**

**budgeted\_amount**

Budgeted amount of the transaction, if it was budgeted ahead of time via a *ScheduledTransaction*. This attribute is only set by *submit()* and *submit()*. And, for some incorrect reason, by *biweeklybudget.models.utils.do\_budget\_transfer()*.

**date**

date of the transaction

**description**

description

**id**

Primary Key

**notes**

free-form notes

**planned\_budget**

Relationship - the *Budget* this transaction was planned to be funded by, if it was planned ahead via a *ScheduledTransaction*.

**planned\_budget\_id**

ID of the Budget this transaction was planned to be funded by, if it was planned ahead via a *ScheduledTransaction*

**reconcile****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

**set\_budget\_amounts** (*budget\_amounts*)

Manage child *BudgetTransaction* objects corresponding to budget allocations of the amount of this transaction. Given a dictionary (*budget\_amounts*) of budgets (either int ID or *Budget* instances) to Decimal amounts, ensure that the BudgetTransactions for this Transaction match those amounts.

This method does NOT commit changes; it will modify database state and add the modifications to this object's session, but the calling code must commit changes.

**Parameters** *budget\_amounts* (*dict*) – Mapping of one or more Budgets to the amount of this Transaction allocated to that Budget. Keys may be either an int *id* or a *Budget* instance, values must be a Decimal.

**transfer**

Relationship - the *Transaction* that makes up the other half/side of a transfer, if this transaction was for a transfer.

**transfer\_id**

If the transaction is one half of a transfer, the Transaction ID of the other half/side of the transfer.

**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 = {'id': <sqlalchemy.orm.attributes.InstrumentedAttribute object>,
```

**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

## biweeklybudget.models.utils module

`biweeklybudget.models.utils.do_budget_transfer` (*db\_sess*, *txn\_date*, *amount*, *account*,  
*from\_budget*, *to\_budget*, *notes=None*)

Transfer a given amount from *from\_budget* to *to\_budget* on *txn\_date*. This method does NOT commit database changes. There are places where we rely on this function not committing changes.

### Parameters

- **db\_sess** (*sqlalchemy.orm.session.Session*) – active database session to use for queries
- **txn\_date** (*datetime.date*) – date to make the transfer Transactions on
- **amount** (*float*) – amount of money to transfer
- **account** (*biweeklybudget.models.account.Account*) –
- **from\_budget** (*biweeklybudget.models.budget\_model.Budget*) –
- **to\_budget** (*biweeklybudget.models.budget\_model.Budget*) –
- **notes** (*str*) – Notes to add to the Transaction

**Returns** list of Transactions created for the transfer

**Return type** list of *Transaction* objects

## biweeklybudget.ofxapi package

`biweeklybudget.ofxapi.apiclient` (*api\_url=None*, *ca\_bundle=None*, *client\_cert=None*,  
*client\_key=None*)

## Submodules

### biweeklybudget.ofxapi.exceptions module

**exception** `biweeklybudget.ofxapi.exceptions.DuplicateFileException` (*acct\_id*,  
*filename*,  
*stmt\_id*)

Bases: `Exception`

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

### biweeklybudget.ofxapi.local module

**class** `biweeklybudget.ofxapi.local.OfxApiLocal` (*db\_sess*)

Bases: `object`

**\_create\_statement** (*acct*, *ofx*, *mtime*, *filename*)

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

#### Parameters

- **acct** (`biweeklybudget.models.account.Account`) – the Account this statement is for
- **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

**\_new\_updated\_counts** ()

Return integer counts of the number of `OFXTransaction` objects that have been created and updated.

**Returns** 2-tuple of new OFXTransactions created, OFXTransactions updated

**Return type** `tuple`

**\_update\_bank\_or\_credit** (*acct*, *ofx*, *stmt*)

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

#### Parameters

- **acct** (`biweeklybudget.models.account.Account`) – the Account this statement is for
- **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`** (*acct, ofx, stmt*)

Update a single OFX file for this Investment account.

**Parameters**

- **acct** (`biweeklybudget.models.account.Account`) – the Account this statement is for
- **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`

**`get_accounts`** ()

Query the database for all *ofxgetter-enabled Accounts* that have a non-empty `biweeklybudget.models.account.Account.ofxgetter_config` and a non-None `biweeklybudget.models.account.Account.vault_creds_path`. Return a dict of string *Account name* to dict with keys:

- `vault_path` - `vault_creds_path`
- `config` - `ofxgetter_config`
- `id` - `id`
- `cat_memo` - `ofx_cat_memo_to_name`

**Returns** dict of account names to configuration

**Return type** dict

**`update_statement_ofx`** (*acct\_id, ofx, mtime=None, filename=None*)

Update a single statement for the specified account, from an OFX file.

**Parameters**

- **acct\_id** (*int*) – Account ID that statement is for
- **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** 3-tuple of the int ID of the *OFXStatement* created by this run, int count of new *OFXTransaction* created, and int count of *OFXTransaction* updated

**Return type** tuple

**Raises** `RuntimeError` on error parsing OFX or unknown account type; `DuplicateFileException` if the file (according to the OFX signon date/time) has already been recorded.

## biweeklybudget.ofxapi.remote module

```
class biweeklybudget.ofxapi.remote.OfxApiRemote (api_base_url, ca_bundle=None,
client_cert_path=None,
client_key_path=None)
```

Bases: object

Remote OFX API client, used by ofxgetter/ofxbackfiller when running on a remote system.

#### `get_accounts()`

Query the database for all *ofxgetter-enabled Accounts* that have a non-empty *biweeklybudget.models.account.Account.ofxgetter\_config* and a non-None *biweeklybudget.models.account.Account.vault\_creds\_path*. Return a dict of string *Account name* to dict with keys:

- *vault\_path* - *vault\_creds\_path*
- *config* - *ofxgetter\_config*
- *id* - *id*
- *cat\_memo* - *ofx\_cat\_memo\_to\_name*

**Returns** dict of account names to configuration

**Return type** dict

#### `update_statement_ofx(acct_id, ofx, mtime=None, filename=None)`

Update a single statement for the specified account, from an OFX file.

##### Parameters

- **acct\_id** (*int*) – Account ID that statement is for
- **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** 3-tuple of the int ID of the *OFXStatement* created by this run, int count of new *OFXTransaction* created, and int count of *OFXTransaction* updated

**Return type** tuple

**Raises** *RuntimeError* on error parsing OFX or unknown account type; *DuplicateFileException* if the file (according to the OFX signon date/time) has already been recorded.

## Submodules

### biweeklybudget.backfill\_ofx module

**class** `biweeklybudget.backfill_ofx.OfxBackfiller` (*client, savedir*)

Bases: `object`

Class to backfill OFX in database from files on disk.

`_do_account_dir` (*acct\_id, path*)

Handle all OFX statements in a per-account directory.

##### Parameters

- **acct\_id** (*int*) – account database ID
- **path** (*str*) – absolute path to per-account directory

`_do_one_file` (*acct\_id, path*)

Parse one OFX file and use OFXUpdater to upsert it into the DB.

### Parameters

- `acct_id` (*int*) – Account ID number
- `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. Build 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` (**Decimal.decimal**) 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.
- `reconcile_id` (**int**) the ID of the TxnReconcile, or None
- `budgets` (**dict**) dict of information on the Budgets this Transaction is against. Keys are budget IDs (**int**), values are dicts with keys “amount” (**Decimal**) and “name” (**string**).

**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` (**Decimal.decimal**) the transaction amount
- `budgeted_amount` (**Decimal.decimal**) 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.
- `reconcile_id` (**int**) the ID of the TxnReconcile, or None
- `planned_budget_id` (**int**) the id of the Budget the transaction was planned against, if any. May be None.
- `planned_budget_name` (**str**) the name of the Budget the transaction was planned against, if any. May be None.
- `budgets` (**dict**) dict of information on the Budgets this Transaction is against. Keys are budget IDs (**int**), values are dicts with keys “amount” (**Decimal**) and “name” (**string**).

**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` (*Decimal.decimal*) - the periodic budget *starting\_balance*.
- `allocated` (*Decimal.decimal*) - 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` (*Decimal.decimal*) - the sum of all actual `Transaction` amounts against the budget this period.

- `trans_total` (*Decimal.decimal*) - the sum of spent amounts for Transactions that have them, or allocated amounts for ScheduledTransactions.
- `remaining` (*Decimal.decimal*) - 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` (*Decimal.decimal*) total amount allocated via *ScheduledTransaction*, *Transaction* (counting the *budgeted\_amount* for Transactions that have one), or *Budget* (not counting income budgets).
- `spent` (*Decimal.decimal*) total amount actually spent via *Transaction*.
- `income` (*Decimal.decimal*) 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` (*Decimal.decimal*) income minus the greater of `allocated` or `spent` for current or future pay periods, or minus `spent` for pay periods ending in the past (*is\_in\_past*)

**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` (**Decimal.decimal**) the transaction amount
- `budgeted_amount` (**Decimal.decimal**) 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.
- `budgets` (**dict**) dict of information on the Budgets this Transaction is against. Keys are budget IDs (**int**), values are dicts with keys “amount” (**Decimal**) and “name” (**string**).
- `reconcile_id` (**int**) the ID of the TxnReconcile, or None
- `planned_budget_id` (**int**) the id of the Budget the transaction was planned against, if any. May be None.
- `planned_budget_name` (**str**) the name of the Budget the transaction was planned against, if any. May be 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

**`clear_cache`** ()

Clear the cached transaction, budget and sum data stored in `self._data_cache` and returned by `_data`.

**`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`

**is\_in\_past**

**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 *BiweeklyPayPeriod*s as a `timedelta`.

**Returns** interval between *BiweeklyPayPeriod*s

**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_account_re_change(session)
    Handler for change of one of:
```

- `re_interest_paid`
- `re_interest_charge`
- `re_late_fee`
- `re_other_fee`
- `re_payment`

When one of these regexes is changed on an Account, we trigger a re-run of `update_is_fields()` on all OFXTransactions for the account.

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

```
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_or_deleted_budget_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_budget_trans_amount_change(**kwargs)`

Handle change of *BudgetTransaction.amount* for existing instances (*trans\_id* is not None). For new or deleted instances, we rely on *handle\_new\_or\_deleted\_budget\_transaction()* called via *handle\_before\_flush()*.

If the *BudgetTransaction*'s *budget* uses 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.handle_new_or_deleted_budget_transaction(session)`

*before\_flush* event handler (`sqlalchemy.orm.events.SessionEvents.before_flush()`) on the DB session, to handle creation of *new* *BudgetTransactions* or deletion of *BudgetTransactions*. For updates to existing *BudgetTransactions*, we rely on *handle\_budget\_trans\_amount\_change()*.

If the *BudgetTransaction*'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_ofx_transaction_new_or_change(session)`

*before\_flush* event handler (`sqlalchemy.orm.events.SessionEvents.before_flush()`) on the DB session, to handle setting the *is\_\** fields on new or changed *OFXTransaction* instances according to its *Account*.

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

`biweeklybudget.db_event_handlers.init_event_listeners(db_session, engine)`

Initialize/register all SQLAlchemy event listeners.

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

**Parameters**

- **db\_session** (*sqlalchemy.orm.scoping.scoped\_session*) – the Database Session
- **engine** (*sqlalchemy.engine.Engine*) – top-level Database Engine instance

`biweeklybudget.db_event_handlers.query_profile_after(conn, cursor, statement, parameters, context, _)`

Query profiling database event listener, to be added as listener on the *Engine*'s *after\_cursor\_execute* event.

For information, see: <http://docs.sqlalchemy.org/en/latest/faq/performance.html#query-profiling>

`biweeklybudget.db_event_handlers.query_profile_before(conn, cursor, statement, parameters, context, _)`

Query profiling database event listener, to be added as listener on the *Engine*'s *before\_cursor\_execute* event.

For information, see: <http://docs.sqlalchemy.org/en/latest/faq/performance.html#query-profiling>

## biweeklybudget.initdb module

`biweeklybudget.initdb.main()`

`biweeklybudget.initdb.parse_args()`

## biweeklybudget.interest module

**class** `biweeklybudget.interest.AdbCompoundedDaily` (*apr*)

Bases: `biweeklybudget.interest._InterestCalculation`

Average Daily Balance method, compounded daily (like American Express).

**calculate** (*principal, first\_d, last\_d, transactions={}*)

Calculate compound interest for the specified principal.

### Parameters

- **principal** (*decimal.Decimal*) – balance at beginning of statement period
- **first\_d** (*datetime.date*) – date of beginning of statement period
- **last\_d** (*datetime.date*) – last date of statement period
- **transactions** (*dict*) – dict of *datetime.date* to float amount adjust the balance by on the specified dates.

**Returns** dict describing the result: `end_balance` (float), `interest_paid` (float)

**Return type** `dict`

**description** = 'Average Daily Balance Compounded Daily (AmEx)'

Human-readable string name of the interest calculation type.

**class** `biweeklybudget.interest.CCStatement` (*interest\_cls, principal, min\_payment\_cls, billing\_period, transactions={}, end\_balance=None, interest\_amt=None*)

Bases: `object`

Represent a credit card statement (one billing period).

**apr**

**billing\_period**

Return the Billing Period for this statement.

**Returns** billing period for this statement

**Return type** `_BillingPeriod`

**end\_date**

**interest**

**minimum\_payment**

Return the minimum payment for the next billing cycle.

**Returns** minimum payment for the next billing cycle

**Return type** `decimal.Decimal`

**next\_with\_transactions** (*transactions={}*)

Return a new `CCStatement` reflecting the next billing period, with a payment of *amount* applied to it.

**Parameters** **transactions** (*dict*) – dict of transactions, *datetime.date* to *Decimal*

**Returns** next period statement, with transactions applied

**Return type** *CCStatement*

**pay** (*amount*)

Return a new *CCStatement* reflecting the next billing period, with a payment of *amount* applied to it at the middle of the period.

**Parameters** **amount** (*decimal.Decimal*) – amount to pay during the next statement period

**Returns** next period statement, with payment applied

**Return type** *CCStatement*

**principal**

**start\_date**

**class** `biweeklybudget.interest.FixedPaymentMethod` (*max\_total\_payment=None*, *increases={}, onetimes={}*) *in-*

Bases: *biweeklybudget.interest.\_PayoffMethod*

TESTING ONLY - pay the same amount on every statement.

**description** = 'TESTING ONLY - Fixed Payment for All Statements'

**find\_payments** (*statements*)

Given a list of statements, return a list of payment amounts to make on each of the statements.

**Parameters** **statements** (*list*) – statements to pay, list of *CCStatement*

**Returns** list of payment amounts to make, same order as *statements*

**Return type** *list*

**show\_in\_ui** = **False**

**class** `biweeklybudget.interest.HighestBalanceFirstMethod` (*max\_total\_payment=None*, *increases={}, onetimes={}*)

Bases: *biweeklybudget.interest.\_PayoffMethod*

Pay statements off from highest to lowest balance.

**description** = 'Highest to Lowest Balance'

**find\_payments** (*statements*)

Given a list of statements, return a list of payment amounts to make on each of the statements.

**Parameters** **statements** (*list*) – statements to pay, list of *CCStatement*

**Returns** list of payment amounts to make, same order as *statements*

**Return type** *list*

**show\_in\_ui** = **True**

**class** `biweeklybudget.interest.HighestInterestRateFirstMethod` (*max\_total\_payment=None*, *increases={}, onetimes={}*)

Bases: *biweeklybudget.interest.\_PayoffMethod*

Pay statements off from highest to lowest interest rate.

**description** = 'Highest to Lowest Interest Rate'

**find\_payments** (*statements*)

Given a list of statements, return a list of payment amounts to make on each of the statements.

**Parameters** `statements` (*list*) – statements to pay, list of *CCStatement*

**Returns** list of payment amounts to make, same order as `statements`

**Return type** *list*

`show_in_ui = True`

`biweeklybudget.interest.INTEREST_CALCULATION_NAMES = {'AdbCompoundedDaily': {'cls': <class 'biweeklybudget.interest.AdbCompoundedDaily'>, 'docstring': 'AdbCompoundedDaily'}}`  
Dict mapping interest calculation class names to their description and docstring.

**class** `biweeklybudget.interest.InterestHelper` (*db\_sess*, *increases={}*, *onetimes={}*)  
Bases: `object`

`_calc_payoff_method` (*cls*)

Calculate payoffs using one method.

**Parameters** `cls` (`biweeklybudget.interest._PayoffMethod`) – payoff method class

**Returns** Dict with integer *account\_id* as the key, and values are dicts with keys “payoff\_months” (int), “total\_payments” (Decimal), “total\_interest” (Decimal), “next\_payment” (Decimal).

**Return type** *dict*

`_get_credit_accounts` ()

Return a dict of *account\_id* to *Account* for all Credit type accounts with OFX data present.

**Returns** dict of *account\_id* to *Account* instance

**Return type** *dict*

`_make_statements` (*accounts*)

Make *CCStatement* instances for each account; return a dict of *account\_id* to *CCStatement* instance.

**Parameters** `accounts` (*dict*) – dict of (int) *account\_id* to *Account* instance

**Returns** dict of (int) *account\_id* to *CCStatement* instance

**Return type** *dict*

**accounts**

Return a dict of *account\_id* to *Account* for all Credit type accounts with OFX data present.

**Returns** dict of *account\_id* to *Account* instance

**Return type** *dict*

`calculate_payoffs` ()

Calculate payoffs for each account/statement.

**Returns** dict of payoff information. Keys are payoff method names. Values are dicts, with keys “description” (str description of the payoff method), “doc” (the docstring of the class), and “results”. The “results” dict has integer *account\_id* as the key, and values are dicts with keys “payoff\_months” (int), “total\_payments” (Decimal), “total\_interest” (Decimal) and *next\_payment* (Decimal).

**Return type** *dict*

**min\_payments**

Return a dict of *account\_id* to minimum payment for the latest statement, for each account.

**Returns** dict of *account\_id* to minimum payment (Decimal)

**Return type** *dict*

```
class biweeklybudget.interest.LowestBalanceFirstMethod(max_total_payment=None,
                                                    increases={}, onetimes={})
```

Bases: *biweeklybudget.interest.\_PayoffMethod*

Pay statements off from lowest to highest balance, a.k.a. the “snowball” method.

**description** = 'Lowest to Highest Balance (a.k.a. Snowball Method)'

**find\_payments** (*statements*)

Given a list of statements, return a list of payment amounts to make on each of the statements.

**Parameters** **statements** (*list*) – statements to pay, list of *CCStatement*

**Returns** list of payment amounts to make, same order as *statements*

**Return type** *list*

**show\_in\_ui** = True

```
class biweeklybudget.interest.LowestInterestRateFirstMethod(max_total_payment=None,
                                                            increases={}, one-
                                                            times={})
```

Bases: *biweeklybudget.interest.\_PayoffMethod*

Pay statements off from lowest to highest interest rate.

**description** = 'Lowest to Highest Interest Rate'

**find\_payments** (*statements*)

Given a list of statements, return a list of payment amounts to make on each of the statements.

**Parameters** **statements** (*list*) – statements to pay, list of *CCStatement*

**Returns** list of payment amounts to make, same order as *statements*

**Return type** *list*

**show\_in\_ui** = True

```
biweeklybudget.interest.MIN_PAYMENT_FORMULA_NAMES = {'MinPaymentAmEx': {'cls': <class 'b
Dict mapping Minimum Payment Formula class names to their description and docstring.
```

```
class biweeklybudget.interest.MinPaymentAmEx
Bases: biweeklybudget.interest._MinPaymentFormula
```

Interest on last statement plus 1% of balance, or \$35 if balance is less than \$35.

**calculate** (*balance, interest*)

Calculate the minimum payment for a statement with the given balance and interest amount.

**Parameters**

- **balance** (*decimal.Decimal*) – balance amount for the statement
- **interest** (*decimal.Decimal*) – interest charged for the statement period

**Returns** minimum payment for the statement

**Return type** *decimal.Decimal*

**description** = 'AmEx - Greatest of Interest Plus 1% of Principal, or \$35'  
human-readable string description of the formula

```
class biweeklybudget.interest.MinPaymentCiti
Bases: biweeklybudget.interest._MinPaymentFormula
```

Greater of: - \$25; - The new balance, if it's less than \$25; - 1 percent of the new balance, plus the current statement's interest charges or minimum interest charges, plus late fees; - 1.5% of the new balance, rounded to the nearest dollar amount.

In all cases, add past fees and finance charges due, plus any amount in excess of credit line.

**calculate** (*balance, interest*)

Calculate the minimum payment for a statement with the given balance and interest amount.

**Parameters**

- **balance** (*decimal.Decimal*) – balance amount for the statement
- **interest** (*decimal.Decimal*) – interest charged for the statement period

**Returns** minimum payment for the statement

**Return type** *decimal.Decimal*

**description** = 'Citi - Greatest of 1.5% of Principal, or 1% of Principal plus interest'  
human-readable string description of the formula

**class** `biweeklybudget.interest.MinPaymentDiscover`

Bases: `biweeklybudget.interest._MinPaymentFormula`

Greater of: - \$35; or - 2% of the New Balance shown on your billing statement; or - \$20, plus any of the following charges as shown on your billing statement: fees for any debt protection product that you enrolled in on or after 2/1/2015; Interest Charges; and Late Fees.

**calculate** (*balance, interest*)

Calculate the minimum payment for a statement with the given balance and interest amount.

**Parameters**

- **balance** (*decimal.Decimal*) – balance amount for the statement
- **interest** (*decimal.Decimal*) – interest charged for the statement period

**Returns** minimum payment for the statement

**Return type** *decimal.Decimal*

**description** = 'Discover - Greatest of 2% of Principal, or \$20 plus Interest, or \$35'  
human-readable string description of the formula

**class** `biweeklybudget.interest.MinPaymentMethod` (*max\_total\_payment=None, in-creates={}, onetimes={}*)

Bases: `biweeklybudget.interest._PayoffMethod`

Pay only the minimum on each statement.

**description** = 'Minimum Payment Only'

**find\_payments** (*statements*)

Given a list of statements, return a list of payment amounts to make on each of the statements.

**Parameters** **statements** (*list*) – statements to pay, list of *CCStatement*

**Returns** list of payment amounts to make, same order as *statements*

**Return type** *list*

**show\_in\_ui** = `True`

`biweeklybudget.interest.PAYOFF_METHOD_NAMES` = {'FixedPaymentMethod': {'cls': <class 'biweeklybudget.interest.FixedPaymentMethod'>, 'description': 'Fixed Payment Method', 'docstring': 'Fixed Payment Method'}}  
Dict mapping Payoff Method class names to their description and docstring.

---

```

class biweeklybudget.interest.SimpleInterest (apr)
    Bases: biweeklybudget.interest._InterestCalculation

    Simple interest, charged on balance at the end of the billing period.

    calculate (principal, first_d, last_d, transactions={})
        Calculate compound interest for the specified principal.

        Parameters
            • principal (decimal.Decimal) – balance at beginning of statement period
            • first_d (datetime.date) – date of beginning of statement period
            • last_d (datetime.date) – last date of statement period
            • transactions (dict) – dict of datetime.date to float amount adjust the balance by on
              the specified dates.

        Returns dict describing the result: end_balance (float), interest_paid (float)

        Return type dict

    description = 'Interest charged once on the balance at end of period.'
        Human-readable string name of the interest calculation type.

class biweeklybudget.interest._BillingPeriod (end_date, start_date=None)
    Bases: object

    description = None
        human-readable string description of the billing period type

    end_date
    next_period
        Return the next billing period after this one.

        Returns next billing period

        Return type _BillingPeriod

    payment_date
    prev_period
        Return the previous billing period before this one.

        Returns previous billing period

        Return type _BillingPeriod

    start_date

class biweeklybudget.interest._InterestCalculation (apr)
    Bases: object

    apr

    calculate (principal, first_d, last_d, transactions={})
        Calculate compound interest for the specified principal.

        Parameters
            • principal (decimal.Decimal) – balance at beginning of statement period
            • first_d (datetime.date) – date of beginning of statement period
            • last_d (datetime.date) – last date of statement period

```

- **transactions** (*dict*) – dict of datetime.date to float amount adjust the balance by on the specified dates.

**Returns** dict describing the result: end\_balance (float), interest\_paid (float)

**Return type** dict

**description = None**

Human-readable string name of the interest calculation type.

**class** biweeklybudget.interest.\_MinPaymentFormula

Bases: object

**calculate** (*balance, interest*)

Calculate the minimum payment for a statement with the given balance and interest amount.

**Parameters**

- **balance** (*decimal.Decimal*) – balance amount for the statement
- **interest** (*decimal.Decimal*) – interest charged for the statement period

**Returns** minimum payment for the statement

**Return type** decimal.Decimal

**description = None**

human-readable string description of the formula

**class** biweeklybudget.interest.\_PayoffMethod (*max\_total\_payment=None, increases={}, onetimes={}*)

Bases: object

A payoff method for multiple cards; a method of figuring out how much to pay on each card, each month.

**description = None**

human-readable string name of the payoff method

**find\_payments** (*statements*)

Given a list of statements, return a list of payment amounts to make on each of the statements.

**Parameters statements** (*list*) – statements to pay, list of *CCStatement*

**Returns** list of payment amounts to make, same order as *statements*

**Return type** list

**max\_total\_for\_period** (*period*)

Given a *\_BillingPeriod*, calculate the maximum total payment for that period, including both *self.\_max\_total* and the increases and onetimes specified on the class constructor.

**Parameters period** (*\_BillingPeriod*) – billing period to get maximum total payment for

**Returns** maximum total payment for the period

**Return type** decimal.Decimal

biweeklybudget.interest.calculate\_payoffs (*payment\_method, statements*)

Calculate the amount of time (in years) and total amount of money required to pay off the cards associated with the given list of statements. Return a list of (*float* number of years, *decimal.Decimal* amount paid, *decimal.Decimal* first payment amount) tuples for each item in *statements*.

**Parameters**

- **payment\_method** (*\_PayoffMethod*) – method used for calculating payment amount to make on each statement; subclass of *\_PayoffMethod*

- **statements** (*list*) – list of *CCStatement* objects to pay off.

**Returns** list of (*float* number of billing periods, *decimal.Decimal* amount paid, *decimal.Decimal* first payment amount) tuples for each item in *statements*

**Return type** *list*

`biweeklybudget.interest.subclass_dict` (*klass*)

### biweeklybudget.load\_data module

`biweeklybudget.load_data.main` ()

`biweeklybudget.load_data.parse_args` ()

### biweeklybudget.ofxgetter module

**class** `biweeklybudget.ofxgetter.OfxGetter` (*client*, *savendir*='.')

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** (*client*)

Return a dict of account information of ofxgetter-enabled accounts, str account name to dict of information about the account.

**Parameters** **client** (Instance of *OfxApiLocal* or *OfxApiRemote*) – API client

**Returns** dict of account information; see `get_accounts` () for details.

**Return type** dict

**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.plaid\_updater module

```
class biweeklybudget.plaid_updater.PlaidUpdateResult (item: biweeklybudget.models.plaid_items.PlaidItem,
                                                    success: bool, updated: int,
                                                    added: int, exc: Optional[Exception], stmt_ids: Optional[List[int]])
```

Bases: object

Describes the result of updating a single account via Plaid.

**as\_dict**

```
class biweeklybudget.plaid_updater.PlaidUpdater
```

Bases: object

**\_do\_item** (*item*, *days*)

Request transactions from Plaid for one Item. Update balances and transactions for each Account in that item.

**Parameters**

- **item** (*PlaidItem*) – the item to update
- **days** (*int*) – number of days of transactions to get from Plaid

**Return type** *PlaidUpdateResult*

**\_get\_transactions** (*access\_token: str*, *start\_dt: datetime.datetime*, *end\_dt: datetime.datetime*)

**\_new\_updated\_counts** ()

Return integer counts of the number of *OFXTransaction* objects that have been created and updated.

**Returns** 2-tuple of new *OFXTransactions* created, *OFXTransactions* updated

**Return type** tuple

**\_stmt\_for\_acct** (*account: biweeklybudget.models.account.Account*, *plaid\_acct\_info: dict*,  
*plaid\_txns: List[dict]*, *end\_dt: datetime.datetime*)

Put Plaid transaction data to the DB

**Parameters**

- **account** – the account to update
- **plaid\_acct\_info** – dict of account information from Plaid
- **txns** – list of transactions from Plaid
- **end\_dt** – current time, as of when transactions were retrieved

**\_update\_bank\_or\_credit** (*end\_dt: datetime.datetime, account: biweeklybudget.models.account.Account, plaid\_acct\_info: dict, plaid\_txns: List[dict], stmt: biweeklybudget.models.ofx\_statement.OFXStatement*)

**\_update\_investment** (*end\_dt: datetime.datetime, account: biweeklybudget.models.account.Account, plaid\_acct\_info: dict, stmt: biweeklybudget.models.ofx\_statement.OFXStatement*)

**classmethod available\_items** ()

Return a list of *PlaidItem* objects that can be updated via Plaid.

**Returns** *PlaidItem* that can be updated via Plaid

**Return type** list of *PlaidItem* objects

**update** (*items=None, days=30*)

Update account balances and transactions from Plaid, for either all Plaid Items that are available or a specified list of Item IDs.

**Parameters**

- **items** (*list or None*) – a list of *PlaidItem* objects to update
- **days** (*int*) – number of days of transactions to get from Plaid

**Returns** list of *PlaidUpdateResult* instances

**Return type** list

## biweeklybudget.prime\_rate module

**class** `biweeklybudget.prime_rate.PrimeRateCalculator` (*db\_session*)

Bases: `object`

**\_get\_prime\_rate** ()

Get the US Prime Rate from MarketWatch; update the DB and return the value.

**Returns** current US Prime Rate

**Return type** `decimal.Decimal`

**\_rate\_from\_marketwatch** ()

**calculate\_apr** (*margin*)

Calculate an APR based on the prime rate.

**Parameters** **margin** (`decimal.Decimal`) – margin added to Prime Rate to get APR

**Returns** effective APR

**Return type** `decimal.Decimal`

**prime\_rate**

Return the current US Prime Rate

**Returns** current US Prime Rate

**Return type** `decimal.Decimal`

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

Base class for screen-scraping bank/financial websites.

`_post_screenshot ()``_pre_screenshot ()``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, useragent=None)`

get a webdriver browser instance

**Parameters**

- **browser\_name** (*str*) – name of browser to get. Can be one of “firefox”, “chrome”, “chrome-headless”, or “phantomjs”
- **useragent** (*str*) – Optionally override the browser’s default user-agent string with this value. Supported for phantomjs or chrome.

`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.BIWEEKLYBUDGET_TEST_TIMESTAMP = None`  
 int - FOR ACCEPTANCE TESTS ONLY - This is used to “fudge” the current time to the specified integer timestamp. Used for acceptance tests only. Do NOT set this outside of acceptance testing.
- `biweeklybudget.settings.CURRENCY_CODE = 'USD'`  
 An ISO 4217 Currency Code specifying the currency to use for all monetary amounts, i.e. “USD”, “EUR”, etc. This setting only effects how monetary values are displayed in the UI, logs, etc. Currently defaults to “USD”. For further information, see *Currency Formatting and Localization*.
- `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.DISTANCE_UNIT = 'Miles'`  
 The full written name of your unit of distance for fuel economy calculations and the Fuel Log. As an example, Miles or Kilometers.
- `biweeklybudget.settings.DISTANCE_UNIT_ABBREVIATION = 'Mi.'`  
 Abbreviation of `biweeklybudget.settings.DISTANCE_UNIT`, such as Mi. or KM.
- `biweeklybudget.settings.FUEL_BUDGET_ID = 1`  
 int - Budget ID to select as default when inputting Fuel Log entries. This must be the database ID of a valid budget.
- `biweeklybudget.settings.FUEL_ECO_ABBREVIATION = 'MPG'`  
 Abbreviation for your distance-per-volume fuel economy measurement, such as MPG or KM/L.
- `biweeklybudget.settings.FUEL_VOLUME_ABBREVIATION = 'Gal.'`  
 Abbreviation of `biweeklybudget.settings.FUEL_VOLUME_UNIT`, such as Gal. or L.
- `biweeklybudget.settings.FUEL_VOLUME_UNIT = 'Gallons'`  
 The full written name of your unit of measure for volume of fuel, to be used for the Fuel Log feature. As an example, Gallons or Litres.
- `biweeklybudget.settings.LOCALE_NAME = 'en_US'`  
 A RFC 5646/BCP 47 Language Tag with a Region suffix to use for number (currency) formatting, i.e. “en\_US”, “en\_GB”, “de\_DE”, etc. If this is not specified (None), it will be looked up from environment variables in the following order: LC\_ALL, LC\_MONETARY, LANG. If none of those variables are set to a valid locale name (not including the “C” locale, which does not specify currency formatting) and this variable is not set, the application will default to “en\_US”. This setting only effects how monetary values are displayed in the UI, logs, etc. For further information, see *Currency Formatting and Localization*.
- `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.PLAID_CLIENT_ID = None`  
 Plaid Client ID
- `biweeklybudget.settings.PLAID_COUNTRY_CODES = None`  
 PLAID\_COUNTRY\_CODES is a comma-separated list of countries for which users will be able to select institutions from.

`biweeklybudget.settings.PLAID_ENV = None`

Plaid environment name. Use 'sandbox' to test with Plaid's Sandbox environment (username: user\_good, password: pass\_good). Use *development* to test with live users and credentials and *production* to go live

`biweeklybudget.settings.PLAID_PRODUCTS = 'transactions'`

PLAID\_PRODUCTS is a comma-separated list of products to use when initializing Link. Note that this list must contain 'assets' in order for the app to be able to create and retrieve asset reports.

`biweeklybudget.settings.PLAID_SECRET = None`

Plaid Secret (client secret)

`biweeklybudget.settings.PLAID_USER_ID = '1'`

PLAID\_USER\_ID is a unique per-user ID for users of Plaid applications. Since this is a single-user app, we just hard-code to "1"

`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(days=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 CI builds, not the real default.

`biweeklybudget.settings_example.DEFAULT_ACCOUNT_ID = 1`

Account ID to show first in dropdown lists

`biweeklybudget.settings_example.FUEL_BUDGET_ID = 1`

int - Budget ID to select as default when inputting Fuel Log entries. This must be the database ID of a valid budget.

`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.PLAID_CLIENT_ID = None`

Plaid Client ID

`biweeklybudget.settings_example.PLAID_COUNTRY_CODES = None`

PLAID\_COUNTRY\_CODES is a comma-separated list of countries for which users will be able to select institutions from.

`biweeklybudget.settings_example.PLAID_ENV = None`

Plaid environment name. Use 'sandbox' to test with Plaid's Sandbox environment (username: user\_good, password: pass\_good). Use *development* to test with live users and credentials and *production* to go live

```
biweeklybudget.settings_example.PLAID_PRODUCTS = 'transactions'
```

PLAID\_PRODUCTS is a comma-separated list of products to use when initializing Link. Note that this list must contain 'assets' in order for the app to be able to create and retrieve asset reports.

```
biweeklybudget.settings_example.PLAID_SECRET = None
```

Plaid Secret (client secret)

```
biweeklybudget.settings_example.PLAID_USER_ID = '1'
```

PLAID\_USER\_ID is a unique per-user ID for users of Plaid applications. Since this is a single-user app, we just hard-code to "1"

```
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(days=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

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

Given an integer day of month ( $1 \leq n \leq 31$ ), return that number with the appropriate suffix (st|nd|rd|th).

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.decode_json_datetime(d)
```

Return a *datetime.datetime* for a *datetime* that was serialized with *MagicJSONEncoder*.

**Parameters** *d* (*dict*) – dict from deserialized JSON

**Returns** *datetime* represented by dict

**Return type** *datetime.datetime*

```
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.fmt_currency` (*amt*)

Using `LOCALE_NAME` and `CURRENCY_CODE`, return *amt* formatted as currency.

**Parameters** *amt* – The amount to format; any numeric type.

**Returns** *amt* formatted for the appropriate locale and currency

**Return type** `str`

`biweeklybudget.utils.in_directory` (*path*)

`biweeklybudget.utils.plaid_client` () → `plaid.api.plaid_api.PlaidApi`

Return an initialized Plaid API client instance.

### biweeklybudget.vault module

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

Bases: `Exception`

**class** `biweeklybudget.vault.Vault` (*addr=None, token\_path=None*)

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.version module

### biweeklybudget.wishlist2project module

**class** `biweeklybudget.wishlist2project.WishlistToProject`

Bases: `object`

**\_do\_project** (*list\_url, project*)

Update a project with information from its wishlist.

**Parameters**

- *list\_url* (`str`) – Amazon wishlist URL
- *project* (`Project`) – the project to update

**Returns** whether or not the update was successful

**Return type** `bool`

**\_get\_wishlist\_projects** ()

Find all projects with descriptions that begin with a wishlist URL.

**Returns** list of (`url`, `Project` object) tuples

**Return type** `list`

**\_project\_items** (*proj*)

Return all of the `BoMItem`s for the specified project, as a dict of URL to `BoMItem`.

**Parameters** `proj` (*Project*) – the project to get items for

**Returns** item URLs to BoMItems

**Return type** `dict`

**static** `_url_is_wishlist` (*url*)

Determine if the given string or URL matches a wishlist.

**Parameters** `url` (*str*) – URL or string to test

**Returns** whether url is a wishlist URL

**Return type** `bool`

`_wishlist_items` (*list\_url*)

Get the items on the specified wishlist.

**Parameters** `list_url` (*str*) – wishlist URL

**Returns** dict of item URL to item details dict

**Return type** `dict`

`run` ()

Run the synchronization.

**Returns** 2-tuple; count of successful syncs, total count of projects with associated wishlists

**Return type** `tuple`

`biweeklybudget.wishlist2project.main` ()

`biweeklybudget.wishlist2project.parse_args` ()

## 6.11 UI JavaScript Docs

### 6.11.1 Files

#### `jsdoc.account_transfer_modal`

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

`accountTransferDivForm` ()

Generate the HTML for the form on the Modal

`accountTransferModal` (*txfr\_date*)

Show the modal popup for transferring between accounts. Uses `accountTransferDivForm` () to generate the form.

#### Arguments

- `txfr_date` (*string*) – The date, as a “yyyy-mm-dd” string, to default the form to. If null or undefined, will default to `BIWEEKLYBUDGET_DEFAULT_DATE`.

#### `jsdoc.accounts_modal`

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

**accountModal** (*id*, *dataTableObj*)

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

**Arguments**

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

**accountModalDivFillAndShow** (*msg*)

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

**accountModalDivForm** ()

Generate the HTML for the form on the Modal

**accountModalDivHandleType** ()

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

### jsdoc.bom\_items

File: biweeklybudget/flaskapp/static/js/bom\_items.js

**reloadProject** ()

Reload the top-level project information on the page.

### jsdoc.bom\_items\_modal

File: biweeklybudget/flaskapp/static/js/bom\_items\_modal.js

**bomItemModal** (*id*)

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

**Arguments**

- **id** (*number*) – the ID of the BoM Item to show a modal for, or null to show modal to add a new Transaction.

**bomItemModalDivFillAndShow** (*msg*)

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

**bomItemModalDivForm** ()

Generate the HTML for the form on the Modal

### jsdoc.budget\_transfer\_modal

File: biweeklybudget/flaskapp/static/js/budget\_transfer\_modal.js

**budgetTransferDivForm** ()

Generate the HTML for the form on the Modal

**budgetTransferModal** (*txfr\_date*)

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

**Arguments**

- **txfr\_date** (*string*) – The date, as a “yyyy-mm-dd” string, to default the form to. If null or undefined, will default to BIWEEKLYBUDGET\_DEFAULT\_DATE.

## 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.creditPayoffErrorModal

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

### **creditPayoffErrorModal** (*acct\_id*)

Trigger Ajax to get account OFX statement information. Ajax callback to create the form and display the modal is *creditPayoffErrorModalForm()*.

#### Arguments

- **acct\_id** (*number*) – the ID of the Account to show data for.

### **creditPayoffErrorModalForm** (*data*)

Generate the HTML for the form on the Credit Payoff Error Modal and show the modal. This is an Ajax callback triggered by a request to /ajax/account\_ofx\_ajax/<int:account\_id> in *creditPayoffErrorModal()*. The response data is generated by *AccountOfxAjax*.

## jsdoc.credit\_payoffs

File: biweeklybudget/flaskapp/static/js/credit\_payoffs.js

### **addIncrease** (*settings*)

Link handler to add another “starting on, increase payments by” form to the credit payoff page.

### **addOnetime** (*settings*)

Link handler to add another one time payment form to the credit payoff page.

### **loadSettings** ()

Load settings from embedded JSON. Called on page load.

### **nextIndex** (*prefix*)

Return the next index for the form with an ID beginning with a given string.

#### Arguments

- **prefix** (*string*) – The prefix of the form IDs.

**Returns** `int` – next form index

#### **recalcPayoffs** ()

Button handler to serialize and submit the forms, to save user input and recalculate the payoff amounts.

#### **removeIncrease** (*idx*)

Remove the specified Increase form.

#### **removeOnetime** (*idx*)

Remove the specified Onetime form.

#### **serializeForms** ()

Serialize the form data into an object and return it.

**Returns** `Object` – serialized forms.

#### **setChanged** ()

Event handler to activate the “Save & Recalculate” button when user input fields have changed.

### jsdoc.custom

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

#### **fmt\_currency** (*value*)

Format a float as currency. If *value* is null, return `&nbsp;`. Otherwise, construct a new instance of `Intl.NumberFormat` and use it to format the currency to a string. The formatter is called with the `LOCALE_NAME` and `CURRENCY_CODE` variables, which are templated into the header of `base.html` using the values specified in the Python settings module.

#### Arguments

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

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

#### **fmt\_null** (*o*)

Format a null object as “&nbsp;”

#### Arguments

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

**Returns** `Object|string` – `o` if not null, `&nbsp;` 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.formBuilder

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

#### **FormBuilder** (*id*)

Create a new FormBuilder to generate an HTML form

**Arguments**

- **id** (*String*) – The form HTML element ID.

`FormBuilder.addCheckbox` (*id, name, label, checked, options*)

Add a checkbox to the form.

**Arguments**

- **id** (*String*) – The id of the form element
- **name** (*String*) – The name of the form element
- **label** (*String*) – The label text for the form element
- **checked** (*Boolean*) – Whether to default to checked or not
- **options** (*Object*) –
- **options.inputHtml** (*String*) – extra HTML string to include in the actual input element (*optional; defaults to null*)

**Returns FormBuilder** – this

`FormBuilder.addCurrency` (*id, name, label, options*)

Add a text input for currency to the form.

**Arguments**

- **id** (*String*) – The id of the form element
- **name** (*String*) – The name of the form element
- **label** (*String*) – The label text for the form element
- **options** (*Object*) –
- **options.htmlClass** (*String*) – The HTML class to apply to the element; defaults to `form-control`.
- **options.helpBlock** (*String*) – Content for block of help text after input; defaults to `null`.
- **options.groupHtml** (*String*) – Additional HTML to add to the outermost form-group div. This is where we'd usually add a default style/display. Defaults to `null`.

**Returns FormBuilder** – this

`FormBuilder.addDatePicker` (*id, name, label, options*)

Add a date picker input to the form.

**Arguments**

- **id** (*String*) – The id of the form element
- **name** (*String*) – The name of the form element
- **label** (*String*) – The label text for the form element
- **options** (*Object*) –
- **options.groupHtml** (*String*) – Additional HTML to add to the outermost

**Returns FormBuilder** – this

`FormBuilder.addHTML` (*content*)

Add a string of HTML to the form.

**Arguments**

- **content** (*String*) – HTML

**Returns FormBuilder** – this

`FormBuilder.addHidden(id, name, value)`

Add a hidden input to the form.

**Arguments**

- **id** (*String*) – The id of the form element
- **name** (*String*) – The name of the form element
- **value** (*String*) – The value of the form element

**Returns FormBuilder** – this

`FormBuilder.addLabelToValueSelect(id, name, label, selectOptions, defaultValue, addNone, options)`

Add a select element to the form, taking an Object of options where keys are the labels and values are the values. This is a convenience wrapper around `budgetTransferDivForm()`.

**Arguments**

- **id** (*String*) – The id of the form element
- **name** (*String*) – The name of the form element
- **label** (*String*) – The label text for the form element
- **selectOptions** (*Object*) – the options for the select, label to value
- **defaultValue** (*String*) – A value to select as the default
- **addNone** (*Boolean*) – If true, prepend an option with a value of “None” and an empty label.
- **options** (*Object*) – Options for rendering the control. Passed through unmodified to `FormBuilder.addSelect()`; see that for details.

**Returns FormBuilder** – this

`FormBuilder.addP(content)`

Add a paragraph (p tag) to the form.

**Arguments**

- **content** (*String*) – The content of the p tag.

**Returns FormBuilder** – this

`FormBuilder.addRadioInline(name, label, options)`

Add an inline radio button set to the form.

Options is an Array of Objects, each object having keys `id`, `value` and `label`. Optional keys are `checked` (`Boolean`) and `onchange`, which will have its value placed literally in the HTML.

**Arguments**

- **name** (*String*) – The name of the form element
- **label** (*String*) – The label text for the form element
- **options** (*Array*) – the options for the select; array of objects each having the following attributes:
  - **options.id** (*String*) – the ID for the option
  - **options.value** (*String*) – the value for the option

- **options.label** (*String*) – the label for the option
- **options.checked** (*Boolean*) – whether the option should be checked by default (*optional; defaults to false*)
- **options.inputHtml** (*String*) – extra HTML string to include in the actual input element (*optional; defaults to null*)

**Returns FormBuilder** – this

`FormBuilder.addSelect` (*id, name, label, selectOptions, options*)

Add a select element to the form.

#### Arguments

- **id** (*String*) – The id of the form element
- **name** (*String*) – The name of the form element
- **label** (*String*) – The label text for the form element
- **selectOptions** (*Array*) – the options for the select, array of objects (order is preserved) each having the following attributes:
- **selectOptions.label** (*String*) – the label for the option
- **selectOptions.value** (*String*) – the value for the option
- **selectOptions.selected** (*Boolean*) – whether the option should be the default selected value (*optional; defaults to False*)
- **options** (*Object*) –
- **options.htmlClass** (*String*) – The HTML class to apply to the element; defaults to `form-control`.
- **options.helpBlock** (*String*) – Content for block of help text after input; defaults to null.
- **options.groupHtml** (*String*) – Additional HTML to add to the outermost form-group div. This is where we'd usually add a default style/display. Defaults to null.

**Returns FormBuilder** – this

`FormBuilder.addText` (*id, name, label, options*)

Add a text input to the form.

#### Arguments

- **id** (*String*) – The id of the form element
- **name** (*String*) – The name of the form element
- **label** (*String*) – The label text for the form element
- **options** (*Object*) –
- **options.groupHtml** (*String*) – Additional HTML to add to the outermost
- **options.inputHtml** (*String*) – extra HTML string to include in the actual input element (*optional; defaults to null*)
- **options.helpBlock** (*String*) – Content for block of help text after input; defaults to null.

**Returns FormBuilder** – this

`FormBuilder.addTextArea` (*id*, *name*, *label*, *options*)

Add a Text Area to the form.

**Arguments**

- **id** (*String*) – The id of the form element
- **name** (*String*) – The name of the form element
- **label** (*String*) – The label text for the form element
- **options** (*Object*) –
- **options.groupHtml** (*String*) – Additional HTML to add to the outermost
- **options.inputHtml** (*String*) – extra HTML string to include in the actual input element (*optional; defaults to null*)
- **options.helpBlock** (*String*) – Content for block of help text after input; defaults to null.

**Returns** `FormBuilder` – this

`FormBuilder.render` ()

Return complete rendered HTML for the form.

**Returns** `String` – form HTML

## jsdoc.forms

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

**handleForm** (*container\_id*, *form\_id*, *post\_url*, *dataTableObj*, *serialize\_func*)

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()`
- **serialize\_func** (*Object*) – If set (i.e. not undefined), this is a function used to serialize the form in place of `serializeForm()`. This function will be passed the ID of the form (`form_id`) and should return an Object suitable for passing to `JSON.stringify()`.

**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.

**handleInlineForm** (*container\_id, form\_id, post\_url, dataTableObj*)

Generic function to handle form submission with server-side validation of an inline (non-modal) form.

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()`

**handleInlineFormError** (*jqXHR, textStatus, errorThrown, container\_id, form\_id*)

Handle an error in the HTTP request to submit the inline (non-modal) form.

**handleInlineFormSubmitted** (*data, container\_id, form\_id, dataTableObj*)

Handle the response from the API URL that the form data is POSTed to, for an inline (non-modal) form.

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.fuel

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

**fuelLogModal** (*dataTableObj*)

Show the modal to add a fuel log entry. This function calls *fuelModalDivForm()* to generate the form HTML, *schedModalDivFillAndShow()* to populate the form for editing, and *handleForm()* to handle the Submit action.

**Arguments**

- **dataTableObj** (*Object | null*) – passed on to *handleForm()*

**fuelModalDivForm** ()

Generate the HTML for the form on the Modal

**vehicleModal** (*id*)

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

**Arguments**

- **id** (*number*) – the ID of the Vehicle to show a modal for, or null to show modal to add a new Vehicle.

**vehicleModalDivFillAndShow** (*msg*)

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

**vehicleModalDivForm** ()

Generate the HTML for the form on the Modal

**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.plaid\_prod

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

**plaidLink** ()

Initiate a Plaid link. Perform the link process and retrieve a public token; POST it to `/ajax/plaid/handle_link`.

**plaidRefresh** (*item\_id*)

Call the `/ajax/plaid/refresh_item_accounts` endpoint and then reload this page.

**plaidUpdate** (*item\_id*)

Update the existing Plaid account / Link.

### jsdoc.projects

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

**activateProject** (*proj\_id*)

Handler for links to activate a project.

**deactivateProject** (*proj\_id*)

Handler for links to deactivate a project.

**handleProjectAdded** ()

Handler for when a project is added via the form.

### 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.

**ignoreOfxTrans** (*acct\_id*, *fitid*)

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

#### Arguments

- **acct\_id** (*number*) – the Account ID of the OFXTransaction
- **fitid** (*string*) – the FitID of the OFXTransaction

**ignoreOfxTransDivForm** (*acct\_id, fitid*)

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

**Arguments**

- **acct\_id** (*number*) – the Account ID of the OFXTransaction
- **fitid** (*string*) – the FitID of the OFXTransaction

**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

**reconcileDoUnreconcileNoTrans** (*acct\_id, fitid*)

Unreconcile a reconciled NoTrans OFXTransaction. This removes *acct\_id* + “%” + *fitid* from the *ofxIgnored* variable and regenerates the OFXTransaction’s div.

**Arguments**

- **acct\_id** (*number*) – the Account ID of the OFXTransaction
- **fitid** (*string*) – the FitID of the OFXTransaction

**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

**reconcileOfxNoTrans(acct\_id, fitid, note)**

Reconcile an OFXTransaction without a matching Transaction. Called from the Save button handler in *ignoreOfxTrans()*.

**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. Called from *reconcileShowTransactions()*, *makeTransSaveCallback()* and *updateReconcileTrans()*.

**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>/<fitid> 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

### **budgetSplitBlur** ()

Triggered when a form element for budget splits loses focus. Calls `validateTransModalSplits()` and updates the warning div with the result.

### **getObjectValueKey** (*obj*, *val*)

Return the first property of *obj* with *val* as its value, or null.

#### Arguments

- **obj** – the object to check
- **val** – the value to look for

### **selectBudget** (*sel\_num*, *budg\_id*)

Select a budget in a budget select element. If *sel\_num* is null then select in `#trans_frm_budget`, else it is expected to be an integer and the selection will be made in `trans_frm_budget_<sel_num>`.

#### Arguments

- **sel\_num** (*number | null*) – The `trans_frm_budget_` Select element suffix, or else null for the `trans_frm_budget` select.
- **budg\_id** (*number*) – The ID of the budget to select.

### **transModal** (*id*, *dataTableObj*)

Show the Transaction modal popup, optionally populated with information for one Transaction. This function calls `transModalDivForm()` to generate the form HTML, `transModalDivFillAndShow()` to populate the form for editing, and `handleForm()` to handle the Submit action (using `transModalFormSerialize()` as a custom serialization function).

#### Arguments

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

**transModalAddSplitBudget** ()

Handler for the “Add Budget” link on trans modal when using budget split.

**transModalBudgetSplitRowHtml** (*row\_num*)

Generate HTML for a budget div inside the split budgets div.

**Arguments**

- **row\_num** (*Integer*) – the budget split row number

**transModalDivFillAndShow** (*msg*)

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

**transModalDivForm** ()

Generate the HTML for the form on the Modal

**transModalFormSerialize** (*form\_id*)

Custom serialization function passed to *handleForm()* for Transaction modal forms generated by *transModal()*. This handles serialization of Transaction forms that may have a budget split, generating data with a *budgets* Object (hash/mapping/dict) with budget ID keys and amount values, suitable for passing directly to *set\_budget\_amounts()*.

**Arguments**

- **form\_id** (*String*) – the ID of the form on the page.

**transModalHandleSplit** ()

Handler for change of the “Budget Split?” (*#trans\_frm\_is\_split*) checkbox.

**transModalSplitBudgetChanged** (*row\_num*)

Called when a budget split dropdown is changed. If its amount box is empty, set it to the transaction amount minus the sum of all other budget splits.

**Arguments**

- **row\_num** (*Integer*) – the budget split row number

**validateTransModalsplits** ()

Function to validate Transaction modal split budgets. Returns null if valid or otherwise a String error message.

## CHAPTER 7

---

### Indices and tables

---

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



### b

biweeklybudget, 54  
biweeklybudget.backfill\_ofx, 97  
biweeklybudget.biweeklypayperiod, 98  
biweeklybudget.cliutils, 103  
biweeklybudget.db, 103  
biweeklybudget.db\_event\_handlers, 104  
biweeklybudget.flaskapp, 54  
biweeklybudget.flaskapp.app, 75  
biweeklybudget.flaskapp.cli\_commands, 75  
biweeklybudget.flaskapp.context\_processors, 75  
biweeklybudget.flaskapp.filters, 76  
biweeklybudget.flaskapp.jinja\_tests, 77  
biweeklybudget.flaskapp.jsonencoder, 77  
biweeklybudget.flaskapp.notifications, 78  
biweeklybudget.flaskapp.views, 54  
biweeklybudget.flaskapp.views.accounts, 54  
biweeklybudget.flaskapp.views.budgets, 56  
biweeklybudget.flaskapp.views.credit\_payoffs, 57  
biweeklybudget.flaskapp.views.example, 58  
biweeklybudget.flaskapp.views.formhandlerview, 58  
biweeklybudget.flaskapp.views.fuel, 60  
biweeklybudget.flaskapp.views.help, 62  
biweeklybudget.flaskapp.views.index, 62  
biweeklybudget.flaskapp.views.ofx, 62  
biweeklybudget.flaskapp.views.payperiods, 64  
biweeklybudget.flaskapp.views.plaid, 66  
biweeklybudget.flaskapp.views.projects, 67  
biweeklybudget.flaskapp.views.reconcile, 70  
biweeklybudget.flaskapp.views.scheduled, 70  
biweeklybudget.flaskapp.views.searchableajaxview, 72  
biweeklybudget.flaskapp.views.transactions, 73  
biweeklybudget.flaskapp.views.utils, 74  
biweeklybudget.initdb, 106  
biweeklybudget.interest, 106  
biweeklybudget.load\_data, 113  
biweeklybudget.models, 79  
biweeklybudget.models.account, 79  
biweeklybudget.models.account\_balance, 82  
biweeklybudget.models.base, 82  
biweeklybudget.models.budget\_model, 83  
biweeklybudget.models.budget\_transaction, 83  
biweeklybudget.models.dbsetting, 84  
biweeklybudget.models.fuel, 84  
biweeklybudget.models.ofx\_statement, 86  
biweeklybudget.models.ofx\_transaction, 87  
biweeklybudget.models.plaid\_accounts, 88  
biweeklybudget.models.plaid\_items, 89  
biweeklybudget.models.projects, 89  
biweeklybudget.models.reconcile\_rule, 91  
biweeklybudget.models.scheduled\_transaction, 91  
biweeklybudget.models.transaction, 92  
biweeklybudget.models.txn\_reconcile, 93  
biweeklybudget.models.utils, 94  
biweeklybudget.ofxapi, 94  
biweeklybudget.ofxapi.exceptions, 95  
biweeklybudget.ofxapi.local, 95  
biweeklybudget.ofxapi.remote, 96  
biweeklybudget.ofxgetter, 113

`biweeklybudget.plaid_updater`, 114  
`biweeklybudget.prime_rate`, 115  
`biweeklybudget.screenscraper`, 116  
`biweeklybudget.settings`, 117  
`biweeklybudget.settings_example`, 118  
`biweeklybudget.utils`, 119  
`biweeklybudget.vault`, 120  
`biweeklybudget.version`, 120  
`biweeklybudget.wishlist2project`, 120

## Symbols

- \_BillingPeriod (class in *biweeklybudget.interest*), 111  
 \_InterestCalculation (class in *biweeklybudget.interest*), 111  
 \_MinPaymentFormula (class in *biweeklybudget.interest*), 112  
 \_PayoffMethod (class in *biweeklybudget.interest*), 112  
 \_alembic\_get\_current\_rev() (in module *biweeklybudget.db*), 103  
 \_args\_dict() (biweeklybudget.get.flaskapp.views.searchableajaxview.SearchableAjaxView method), 72  
 \_args\_set\_type() (biweeklybudget.get.flaskapp.views.searchableajaxview.SearchableAjaxView method), 72  
 \_budget\_names() (biweeklybudget.get.flaskapp.views.budgets.BudgetSpendingChartView method), 56  
 \_by\_month() (biweeklybudget.get.flaskapp.views.budgets.BudgetSpendingChartView method), 56  
 \_by\_pay\_period() (biweeklybudget.get.flaskapp.views.budgets.BudgetSpendingChartView method), 56  
 \_calc\_payoff\_method() (biweeklybudget.get.interest.InterestHelper method), 108  
 \_create\_statement() (biweeklybudget.get.ofxapi.local.OfxApiLocal method), 95  
 \_data (biweeklybudget.biweeklypayperiod.BiweeklyPayPeriod attribute), 98  
 \_dict\_for\_sched\_trans() (biweeklybudget.get.biweeklypayperiod.BiweeklyPayPeriod method), 98  
 \_dict\_for\_trans() (biweeklybudget.get.biweeklypayperiod.BiweeklyPayPeriod method), 99  
 \_dict\_properties (biweeklybudget.get.models.base.ModelAsDict attribute), 82  
 \_dict\_properties (biweeklybudget.get.models.transaction.Transaction attribute), 92  
 \_do\_account\_dir() (biweeklybudget.get.backfill\_ofx.OfxBackfiller method), 97  
 \_do\_item() (biweeklybudget.get.plaid\_updater.PlaidUpdater method), 114  
 \_do\_one\_file() (biweeklybudget.get.backfill\_ofx.OfxBackfiller method), 97  
 \_do\_project() (biweeklybudget.get.wishlist2project.WishlistToProject method), 120  
 \_filterhack() (biweeklybudget.get.flaskapp.views.fuel.FuelAjax method), 60  
 \_filterhack() (biweeklybudget.get.flaskapp.views.ofx.OfxAjax method), 63  
 \_filterhack() (biweeklybudget.get.flaskapp.views.projects.BoMItemsAjax method), 68  
 \_filterhack() (biweeklybudget.get.flaskapp.views.projects.ProjectsAjax method), 68  
 \_filterhack() (biweeklybudget.get.flaskapp.views.scheduled.ScheduledAjax method), 71  
 \_filterhack() (biweeklybudget.get.flaskapp.views.searchableajaxview.SearchableAjaxView method), 72  
 \_filterhack() (biweeklybudget.get.flaskapp.views.transactions.TransactionsAjax method), 74  
 \_form() (biweeklybudget.get.flaskapp.views.plaid.PlaidUpdate method), 67  
 \_get\_credit\_accounts() (biweeklybud-

|   |   |
|---|---|
| <code>get.interest.InterestHelper</code> method), 108   | <code>get.prime_rate.PrimeRateCalculator</code> method), 115  |
| <code>_get_ofx_scraper()</code> ( <code>biweeklybudget.ofxgetter.OfxGetter</code> method), 113                                    | <code>_sa_class_manager</code> ( <code>biweeklybudget.models.account.Account</code> attribute), 79                            |
| <code>_get_prime_rate()</code> ( <code>biweeklybudget.prime_rate.PrimeRateCalculator</code> method), 115                          | <code>_sa_class_manager</code> ( <code>biweeklybudget.models.account_balance.AccountBalance</code> attribute), 82             |
| <code>_get_transactions()</code> ( <code>biweeklybudget.plaid_updater.PlaidUpdater</code> method), 114                            | <code>_sa_class_manager</code> ( <code>biweeklybudget.models.budget_model.Budget</code> attribute), 83                        |
| <code>_get_wishlist_projects()</code> ( <code>biweeklybudget.wishlist2project.WishlistToProject</code> method), 120               | <code>_sa_class_manager</code> ( <code>biweeklybudget.models.budget_transaction.BudgetTransaction</code> attribute), 83       |
| <code>_have_column_search()</code> ( <code>biweeklybudget.flaskapp.views.searchableajaxview.SearchableAjaxView</code> method), 73 | <code>_sa_class_manager</code> ( <code>biweeklybudget.models.dbsetting.DBSetting</code> attribute), 84                        |
| <code>_income_budget_ids</code> ( <code>biweeklybudget.biweeklypayperiod.BiweeklyPayPeriod</code> attribute), 99                  | <code>_sa_class_manager</code> ( <code>biweeklybudget.models.fuel.FuelFill</code> attribute), 84                              |
| <code>_make_budget_sums()</code> ( <code>biweeklybudget.biweeklypayperiod.BiweeklyPayPeriod</code> method), 99                    | <code>_sa_class_manager</code> ( <code>biweeklybudget.models.fuel.Vehicle</code> attribute), 85                               |
| <code>_make_combined_transactions()</code> ( <code>biweeklybudget.biweeklypayperiod.BiweeklyPayPeriod</code> method), 100         | <code>_sa_class_manager</code> ( <code>biweeklybudget.models.ofx_statement.OFXStatement</code> attribute), 86                 |
| <code>_make_overall_sums()</code> ( <code>biweeklybudget.biweeklypayperiod.BiweeklyPayPeriod</code> method), 100                  | <code>_sa_class_manager</code> ( <code>biweeklybudget.models.ofx_transaction.OFXTransaction</code> attribute), 87             |
| <code>_make_statements()</code> ( <code>biweeklybudget.interest.InterestHelper</code> method), 108                                | <code>_sa_class_manager</code> ( <code>biweeklybudget.models.plaid_accounts.PlaidAccount</code> attribute), 88                |
| <code>_new_updated_counts()</code> ( <code>biweeklybudget.ofxapi.local.OfxApiLocal</code> method), 95                             | <code>_sa_class_manager</code> ( <code>biweeklybudget.models.plaid_items.PlaidItem</code> attribute), 89                      |
| <code>_new_updated_counts()</code> ( <code>biweeklybudget.plaid_updater.PlaidUpdater</code> method), 114                          | <code>_sa_class_manager</code> ( <code>biweeklybudget.models.projects.BoMItem</code> attribute), 89                           |
| <code>_ofx_to_db()</code> ( <code>biweeklybudget.ofxgetter.OfxGetter</code> method), 113  | <code>_sa_class_manager</code> ( <code>biweeklybudget.models.projects.Project</code> attribute), 90                           |
| <code>_payment_settings_dict()</code> ( <code>biweeklybudget.flaskapp.views.credit_payoffs.CreditPayoffsView</code> method), 57   | <code>_sa_class_manager</code> ( <code>biweeklybudget.models.reconcile_rule.ReconcileRule</code> attribute), 91               |
| <code>_payoffs_list()</code> ( <code>biweeklybudget.flaskapp.views.credit_payoffs.CreditPayoffsView</code> method), 58            | <code>_sa_class_manager</code> ( <code>biweeklybudget.models.scheduled_transaction.ScheduledTransaction</code> attribute), 91 |
| <code>_post_screenshot()</code> ( <code>biweeklybudget.screenscraper.ScreenScraper</code> method), 116                            | <code>_sa_class_manager</code> ( <code>biweeklybudget.models.transaction.Transaction</code> attribute), 92                    |
| <code>_pre_screenshot()</code> ( <code>biweeklybudget.screenscraper.ScreenScraper</code> method), 116                             | <code>_sa_class_manager</code> ( <code>biweeklybudget.models.txn_reconcile.TxnReconcile</code> attribute), 93                 |
| <code>_previous_entry()</code> ( <code>biweeklybudget.models.fuel.FuelFill</code> method), 84                                     | <code>_scheduled_transactions_date()</code> ( <code>biweeklybudget.biweeklypayperiod.BiweeklyPayPeriod</code> method), 100    |
| <code>_project_items()</code> ( <code>biweeklybudget.wishlist2project.WishlistToProject</code> method), 120                       | <code>_scheduled_transactions_monthly()</code> ( <code>biweeklybudget.biweeklypayperiod.BiweeklyPayPeriod</code> method), 100 |
| <code>_rate_from_marketwatch()</code> ( <code>biweeklybudget</code> method), 115  |   |

|  |  |
|--|--|
| <code>method</code> ), 100   | <code>account</code> ( <i>biweeklybud-</i>                                 |
| <code>_scheduled_transactions_per_period()</code>                          | <code>get.models.account_balance.AccountBalance</code>                     |
| ( <i>biweeklybudget.biweeklypayperiod.BiweeklyPayPeriod</i>                | <code>attribute</code> ), 82   |
| <code>method</code> ), 100   | <code>account</code> ( <i>biweeklybud-</i>                                 |
| <code>_stmt_for_acct()</code> ( <i>biweeklybud-</i>                        | <code>get.models.ofx_statement.OFXStatement</code>                         |
| <code>get.plaid_updater.PlaidUpdater</code> <code>method</code> ),         | <code>attribute</code> ), 86   |
| 114  | <code>account</code> ( <i>biweeklybud-</i>                                 |
| <code>_trans_dict()</code> ( <i>biweeklybud-</i>                           | <code>get.models.ofx_transaction.OFXTransaction</code>                     |
| <code>get.biweeklypayperiod.BiweeklyPayPeriod</code>                       | <code>attribute</code> ), 87   |
| <code>method</code> ), 100   | <code>account</code> ( <i>biweeklybud-</i>                                 |
| <code>_transactions()</code> ( <i>biweeklybud-</i>                         | <code>get.models.plaid_accounts.PlaidAccount</code>                        |
| <code>get.biweeklypayperiod.BiweeklyPayPeriod</code>                       | <code>attribute</code> ), 88   |
| <code>method</code> ), 101   | <code>account</code> ( <i>biweeklybud-</i>                                 |
| <code>_update()</code> ( <i>biweeklybud-</i>                               | <code>get.models.scheduled_transaction.ScheduledTransaction</code>         |
| <code>get.flaskapp.views.plaid.PlaidUpdate</code> <code>method</code> ),   | <code>attribute</code> ), 91   |
| 67   | <code>account</code> ( <i>biweeklybud-</i>                                 |
| <code>_update_bank_or_credit()</code> ( <i>biweeklybud-</i>                | <code>get.models.transaction.Transaction</code> <code>attribute</code> ),  |
| <code>get.ofxapi.local.OfxApiLocal</code> <code>method</code> ), 95        | 92   |
| <code>_update_bank_or_credit()</code> ( <i>biweeklybud-</i>                | <code>Account</code> ( <i>class in biweeklybudget.models.account</i> ), 79 |
| <code>get.plaid_updater.PlaidUpdater</code> <code>method</code> ),         | <code>account_amount</code> ( <i>biweeklybud-</i>                          |
| 115  | <code>get.models.ofx_transaction.OFXTransaction</code>                     |
| <code>_update_investment()</code> ( <i>biweeklybud-</i>                    | <code>attribute</code> ), 87   |
| <code>get.ofxapi.local.OfxApiLocal</code> <code>method</code> ), 95        | <code>account_id</code> ( <i>biweeklybud-</i>                              |
| <code>_update_investment()</code> ( <i>biweeklybud-</i>                    | <code>get.models.account_balance.AccountBalance</code>                     |
| <code>get.plaid_updater.PlaidUpdater</code> <code>method</code> ),         | <code>attribute</code> ), 82   |
| 115  | <code>account_id</code> ( <i>biweeklybud-</i>                              |
| <code>_url_is_wishlist()</code> ( <i>biweeklybud-</i>                      | <code>get.models.ofx_statement.OFXStatement</code>                         |
| <code>get.wishlist2project.WishlistToProject</code> <code>static</code>    | <code>attribute</code> ), 86   |
| <code>method</code> ), 121   | <code>account_id</code> ( <i>biweeklybud-</i>                              |
| <code>_validate_date_ymd()</code> ( <i>biweeklybud-</i>                    | <code>get.models.ofx_transaction.OFXTransaction</code>                     |
| <code>get.flaskapp.views.formhandlerview.FormHandlerView</code>            | <code>attribute</code> ), 87   |
| <code>method</code> ), 58  | <code>account_id</code> ( <i>biweeklybud-</i>                              |
| <code>_validate_decimal()</code> ( <i>biweeklybud-</i>                     | <code>get.models.plaid_accounts.PlaidAccount</code>                        |
| <code>get.flaskapp.views.formhandlerview.FormHandlerView</code>            | <code>attribute</code> ), 88   |
| <code>method</code> ), 59  | <code>account_id</code> ( <i>biweeklybud-</i>                              |
| <code>_validate_float()</code> ( <i>biweeklybud-</i>                       | <code>get.models.scheduled_transaction.ScheduledTransaction</code>         |
| <code>get.flaskapp.views.formhandlerview.FormHandlerView</code>            | <code>attribute</code> ), 91   |
| <code>method</code> ), 59  | <code>account_id</code> ( <i>biweeklybud-</i>                              |
| <code>_validate_int()</code> ( <i>biweeklybud-</i>                         | <code>get.models.transaction.Transaction</code> <code>attribute</code> ),  |
| <code>get.flaskapp.views.formhandlerview.FormHandlerView</code>            | 92   |
| <code>method</code> ), 59  | <code>account_subtype</code> ( <i>biweeklybud-</i>                         |
| <code>_validate_not_empty()</code> ( <i>biweeklybud-</i>                   | <code>get.models.plaid_accounts.PlaidAccount</code>                        |
| <code>get.flaskapp.views.formhandlerview.FormHandlerView</code>            | <code>attribute</code> ), 89   |
| <code>method</code> ), 59  | <code>account_type</code> ( <i>biweeklybud-</i>                            |
| <code>_wishlist_items()</code> ( <i>biweeklybud-</i>                       | <code>get.models.plaid_accounts.PlaidAccount</code>                        |
| <code>get.wishlist2project.WishlistToProject</code> <code>method</code> ), | <code>attribute</code> ), 89   |
| 121  | <code>AccountAjax</code> ( <i>class in biweeklybud-</i>                    |
| <code>_write_ofx_file()</code> ( <i>biweeklybud-</i>                       | <code>get.flaskapp.views.accounts</code> ), 54                             |
| <code>get.ofxgetter.OfxGetter</code> <code>method</code> ), 113            | <code>AccountBalance</code> ( <i>class in biweeklybud-</i>                 |
|  | <code>get.models.account_balance</code> ), 82                              |
|  | <code>AccountFormHandler</code> ( <i>class in biweeklybud-</i>             |
|  | <code>get.flaskapp.views.accounts</code> ), 54                             |
| <code>access_token</code> ( <i>biweeklybud-</i>                            | <code>accountModal()</code> ( <i>built-in function</i> ), 121              |
| <code>get.models.plaid_items.PlaidItem</code> <code>attribute</code> ),    | <code>accountModalDivFillAndShow()</code> ( <i>built-in func-</i>          |
| 89   |  |

## A

- tion), 122
- accountModalDivForm() (built-in function), 122
- accountModalDivHandleType() (built-in function), 122
- AccountOfxAjax (class in `biweeklybudget.flaskapp.views.credit_payoffs`), 57
- AccountOfxFormHandler (class in `biweeklybudget.flaskapp.views.credit_payoffs`), 57
- accounts (`biweeklybudget.interest.InterestHelper` attribute), 108
- accounts() (`biweeklybudget.ofxgetter.OfxGetter` static method), 113
- AccountsView (class in `biweeklybudget.flaskapp.views.accounts`), 55
- accountTransferDivForm() (built-in function), 121
- accountTransferModal() (built-in function), 121
- AccountTxfrFormHandler (class in `biweeklybudget.flaskapp.views.accounts`), 55
- acct\_icon\_filter() (in module `biweeklybudget.flaskapp.filters`), 76
- acct\_type (`biweeklybudget.models.account.Account` attribute), 79
- acct\_type (`biweeklybudget.models.ofx_statement.OFXStatement` attribute), 86
- AcctBalanceChartView (class in `biweeklybudget.flaskapp.views.index`), 62
- acctid (`biweeklybudget.models.ofx_statement.OFXStatement` attribute), 86
- AcctType (class in `biweeklybudget.models.account`), 81
- activateProject() (built-in function), 131
- actual\_amount (`biweeklybudget.models.transaction.Transaction` attribute), 92
- AdbCompoundedDaily (class in `biweeklybudget.interest`), 106
- add\_currency\_symbol() (in module `biweeklybudget.flaskapp.context_processors`), 75
- addIncrease() (built-in function), 123
- addOnetime() (built-in function), 123
- ago\_filter() (in module `biweeklybudget.flaskapp.filters`), 76
- all\_accounts (`biweeklybudget.models.plaid_items.PlaidItem` attribute), 89
- all\_statements (`biweeklybudget.models.account.Account` attribute), 79
- amount (`biweeklybudget.models.budget_transaction.BudgetTransaction` attribute), 83
- amount (`biweeklybudget.models.ofx_transaction.OFXTransaction` attribute), 87
- amount (`biweeklybudget.models.scheduled_transaction.ScheduledTransaction` attribute), 91
- apiclient() (in module `biweeklybudget.ofxapi`), 94
- apr (`biweeklybudget.interest._InterestCalculation` attribute), 111
- apr (`biweeklybudget.interest.CCStatement` attribute), 106
- apr (`biweeklybudget.models.account.Account` attribute), 79
- as\_dict (`biweeklybudget.models.account.AcctType` attribute), 81
- as\_dict (`biweeklybudget.models.base.ModelAsDict` attribute), 82
- as\_dict (`biweeklybudget.models.plaid_updater.PlaidUpdateResult` attribute), 114
- as\_of (`biweeklybudget.models.ofx_statement.OFXStatement` attribute), 86
- avail (`biweeklybudget.models.account_balance.AccountBalance` attribute), 82
- avail\_bal (`biweeklybudget.models.ofx_statement.OFXStatement` attribute), 86
- avail\_bal\_as\_of (`biweeklybudget.models.ofx_statement.OFXStatement` attribute), 86
- avail\_date (`biweeklybudget.models.account_balance.AccountBalance` attribute), 82
- available\_items() (`biweeklybudget.models.plaid_updater.PlaidUpdater` class method), 115
- ## B
- balance (`biweeklybudget.models.account.Account` attribute), 79
- Bank (`biweeklybudget.models.account.AcctType` attribute), 81
- bankid (`biweeklybudget.models.ofx_statement.OFXStatement` attribute), 86
- before\_request() (in module `biweeklybudget.flaskapp.app`), 75
- billing\_period (`biweeklybudget.interest.CCStatement` attribute), 106
- biweeklybudget (module), 54
- biweeklybudget.backfill\_ofx (module), 97
- biweeklybudget.biweeklypayperiod (module), 98
- biweeklybudget.cliutils (module), 103
- biweeklybudget.db (module), 103

biweeklybudget.db\_event\_handlers (*module*), 104  
 biweeklybudget.flaskapp (*module*), 54  
 biweeklybudget.flaskapp.app (*module*), 75  
 biweeklybudget.flaskapp.cli\_commands (*module*), 75  
 biweeklybudget.flaskapp.context\_processors (*module*), 75  
 biweeklybudget.flaskapp.filters (*module*), 76  
 biweeklybudget.flaskapp.jinja\_tests (*module*), 77  
 biweeklybudget.flaskapp.jsonencoder (*module*), 77  
 biweeklybudget.flaskapp.notifications (*module*), 78  
 biweeklybudget.flaskapp.views (*module*), 54  
 biweeklybudget.flaskapp.views.accounts (*module*), 54  
 biweeklybudget.flaskapp.views.budgets (*module*), 56  
 biweeklybudget.flaskapp.views.credit\_payoffs (*module*), 57  
 biweeklybudget.flaskapp.views.example (*module*), 58  
 biweeklybudget.flaskapp.views.formhandlerview (*module*), 58  
 biweeklybudget.flaskapp.views.fuel (*module*), 60  
 biweeklybudget.flaskapp.views.help (*module*), 62  
 biweeklybudget.flaskapp.views.index (*module*), 62  
 biweeklybudget.flaskapp.views.ofx (*module*), 62  
 biweeklybudget.flaskapp.views.payperiods (*module*), 64  
 biweeklybudget.flaskapp.views.plaid (*module*), 66  
 biweeklybudget.flaskapp.views.projects (*module*), 67  
 biweeklybudget.flaskapp.views.reconcile (*module*), 70  
 biweeklybudget.flaskapp.views.scheduled (*module*), 70  
 biweeklybudget.flaskapp.views.searchable (*module*), 72  
 biweeklybudget.flaskapp.views.transactions (*module*), 73  
 biweeklybudget.flaskapp.views.utils (*module*), 74  
 biweeklybudget.initdb (*module*), 106  
 biweeklybudget.interest (*module*), 106  
 biweeklybudget.load\_data (*module*), 113  
 biweeklybudget.models (*module*), 79  
 biweeklybudget.models.account (*module*), 79  
 biweeklybudget.models.account\_balance (*module*), 82  
 biweeklybudget.models.base (*module*), 82  
 biweeklybudget.models.budget\_model (*module*), 83  
 biweeklybudget.models.budget\_transaction (*module*), 83  
 biweeklybudget.models.dbsetting (*module*), 84  
 biweeklybudget.models.fuel (*module*), 84  
 biweeklybudget.models.ofx\_statement (*module*), 86  
 biweeklybudget.models.ofx\_transaction (*module*), 87  
 biweeklybudget.models.plaid\_accounts (*module*), 88  
 biweeklybudget.models.plaid\_items (*module*), 89  
 biweeklybudget.models.projects (*module*), 89  
 biweeklybudget.models.reconcile\_rule (*module*), 91  
 biweeklybudget.models.scheduled\_transaction (*module*), 91  
 biweeklybudget.models.transaction (*module*), 92  
 biweeklybudget.models.txn\_reconcile (*module*), 93  
 biweeklybudget.models.utils (*module*), 94  
 biweeklybudget.ofxapi (*module*), 94  
 biweeklybudget.ofxapi.exceptions (*module*), 95  
 biweeklybudget.ofxapi.local (*module*), 95  
 biweeklybudget.ofxapi.remote (*module*), 96  
 biweeklybudget.ofxgetter (*module*), 113  
 biweeklybudget.plaid\_updater (*module*), 114  
 biweeklybudget.prime\_rate (*module*), 115  
 biweeklybudget.screenscraper (*module*), 116  
 biweeklybudget.settings (*module*), 117  
 biweeklybudget.settings\_example (*module*), 118  
 biweeklybudget.utils (*module*), 119  
 biweeklybudget.vault (*module*), 120  
 biweeklybudget.version (*module*), 120  
 biweeklybudget.wishlist2project (*module*), 120  
 BIWEEKLYBUDGET\_TEST\_TIMESTAMP (*in module biweeklybudget.settings*), 117  
 BiweeklyPayPeriod (*class in biweeklybudget.biweeklypayperiod*), 98  
 BoMItem (*class in biweeklybudget.models.projects*), 89  
 BoMItemAjax (*class in biweeklybud-*

`get.flaskapp.views.projects`), 67

`BoMItemFormHandler` (class in `biweeklybudget.get.flaskapp.views.projects`), 67

`bomItemModal()` (built-in function), 122

`bomItemModalDivFillAndShow()` (built-in function), 122

`bomItemModalDivForm()` (built-in function), 122

`BoMItemsAjax` (class in `biweeklybudget.get.flaskapp.views.projects`), 68

`BoMItemView` (class in `biweeklybudget.get.flaskapp.views.projects`), 68

`brokerid` (`biweeklybudget.get.models.ofx_statement.OFXStatement` attribute), 86

`budget` (`biweeklybudget.get.models.budget_transaction.BudgetTransaction` attribute), 83

`budget` (`biweeklybudget.get.models.scheduled_transaction.ScheduledTransaction` attribute), 91

`Budget` (class in `biweeklybudget.get.models.budget_model`), 83

`budget_account_sum()` (`biweeklybudget.get.flaskapp.notifications.NotificationsController` static method), 78

`budget_account_unreconciled()` (`biweeklybudget.get.flaskapp.notifications.NotificationsController` static method), 78

`budget_cell_filter()` (in module `biweeklybudget.get.flaskapp.filters`), 76

`budget_id` (`biweeklybudget.get.models.budget_transaction.BudgetTransaction` attribute), 83

`budget_id` (`biweeklybudget.get.models.scheduled_transaction.ScheduledTransaction` attribute), 91

`budget_sums` (`biweeklybudget.get.biweeklypayerperiod.BiweeklyPayPeriod` attribute), 101

`budget_transactions` (`biweeklybudget.get.models.budget_model.Budget` attribute), 83

`budget_transactions` (`biweeklybudget.get.models.transaction.Transaction` attribute), 92

`BudgetAjax` (class in `biweeklybudget.get.flaskapp.views.budgets`), 56

`budgeted_amount` (`biweeklybudget.get.models.transaction.Transaction` attribute), 92

`BudgetFormHandler` (class in `biweeklybudget.get.flaskapp.views.budgets`), 56

`budgetModal()` (built-in function), 123

`budgetModalDivFillAndShow()` (built-in function), 123

`budgetModalDivForm()` (built-in function), 123

`budgetModalDivHandleType()` (built-in function), 123

`BudgetSpendingChartView` (class in `biweeklybudget.get.flaskapp.views.budgets`), 56

`budgetSplitBlur()` (built-in function), 135

`BudgetsView` (class in `biweeklybudget.get.flaskapp.views.budgets`), 57

`BudgetTransaction` (class in `biweeklybudget.get.models.budget_transaction`), 83

`budgetTransferDivForm()` (built-in function), 122

`budgetTransferModal()` (built-in function), 122

`BudgetTxfrFormHandler` (class in `biweeklybudget.get.flaskapp.views.budgets`), 56

## C

`calculate()` (`biweeklybudget.get.interest.InterestCalculation` method), 111

`calculate()` (`biweeklybudget.get.interest.MinPaymentFormula` method), 112

`calculate()` (`biweeklybudget.get.interest.AdbCompoundedDaily` method), 106

`calculate()` (`biweeklybudget.get.interest.MinPaymentAmEx` method), 109

`calculate()` (`biweeklybudget.get.interest.MinPaymentCiti` method), 110

`calculate()` (`biweeklybudget.get.interest.MinPaymentDiscover` method), 110

`calculate()` (`biweeklybudget.get.interest.SimpleInterest` method), 111

`calculate_apr()` (`biweeklybudget.get.prime_rate.PrimeRateCalculator` method), 115

`calculate_mpg()` (`biweeklybudget.get.models.fuel.FuelFill` method), 84

`calculate_payoffs()` (`biweeklybudget.get.interest.InterestHelper` method), 108

`calculate_payoffs()` (in module `biweeklybudget.get.interest`), 112

`calculated_miles` (`biweeklybudget.get.models.fuel.FuelFill` attribute), 84

`calculated_mpg` (`biweeklybudget.get.models.fuel.FuelFill` attribute), 84

`Cash` (`biweeklybudget.get.models.account.AcctType` attribute), 81

`CCStatement` (class in `biweeklybudget.get.interest`), 106

- checknum (biweeklybudget.models.ofx\_transaction.OFXTransaction attribute), 87
- clean\_fitid() (built-in function), 131
- cleanup\_db() (in module biweeklybudget.db), 103
- clear\_cache() (biweeklybudget.models.biweeklypayperiod.BiweeklyPayPeriod method), 101
- cost\_per\_gallon (biweeklybudget.models.fuel.FuelFill attribute), 84
- Credit (biweeklybudget.models.account.AcctType attribute), 81
- credit\_limit (biweeklybudget.models.account.Account attribute), 79
- creditPayoffErrorModal() (built-in function), 123
- creditPayoffErrorModalForm() (built-in function), 123
- CreditPayoffsView (class in biweeklybudget.flaskapp.views.credit\_payoffs), 57
- currency (biweeklybudget.models.ofx\_statement.OFXStatement attribute), 86
- CURRENCY\_CODE (in module biweeklybudget.settings), 117
- current\_balance (biweeklybudget.models.budget\_model.Budget attribute), 83
- CustomLoggingWSGIRequestHandler (class in biweeklybudget.flaskapp.cli\_commands), 75
- ## D
- date (biweeklybudget.models.fuel.FuelFill attribute), 84
- date (biweeklybudget.models.scheduled\_transaction.ScheduledTransaction attribute), 91
- date (biweeklybudget.models.transaction.Transaction attribute), 92
- date\_posted (biweeklybudget.models.ofx\_transaction.OFXTransaction attribute), 87
- date\_suffix() (in module biweeklybudget.utils), 119
- DateTestJS (class in biweeklybudget.flaskapp.views.utils), 74
- dateymd\_filter() (in module biweeklybudget.flaskapp.filters), 76
- day\_of\_month (biweeklybudget.models.scheduled\_transaction.ScheduledTransaction attribute), 91
- DB\_CONNSTRING (in module biweeklybudget.settings), 117
- DB\_CONNSTRING (in module biweeklybudget.settings\_example), 118
- db\_session (in module biweeklybudget.db), 104
- DBSetting (class in biweeklybudget.models.dbsetting), 84
- deactivateProject() (built-in function), 131
- decimal\_to\_percent() (in module biweeklybudget.flaskapp.filters), 76
- decode\_json\_datetime() (in module biweeklybudget.utils), 119
- default() (biweeklybudget.flaskapp.jsonencoder.MagicJSONEncoder method), 77
- DEFAULT\_ACCOUNT\_ID (in module biweeklybudget.settings), 117
- DEFAULT\_ACCOUNT\_ID (in module biweeklybudget.settings\_example), 118
- default\_value (biweeklybudget.models.dbsetting.DBSetting attribute), 84
- description (biweeklybudget.interest.\_BillingPeriod attribute), 111
- description (biweeklybudget.interest.\_InterestCalculation attribute), 112
- description (biweeklybudget.interest.\_MinPaymentFormula attribute), 112
- description (biweeklybudget.interest.\_PayoffMethod attribute), 112
- description (biweeklybudget.interest.AdbCompoundedDaily attribute), 106
- description (biweeklybudget.interest.FixedPaymentMethod attribute), 107
- description (biweeklybudget.interest.HighestBalanceFirstMethod attribute), 107
- description (biweeklybudget.interest.HighestInterestRateFirstMethod attribute), 107
- description (biweeklybudget.interest.LowestBalanceFirstMethod attribute), 109
- description (biweeklybudget.interest.LowestInterestRateFirstMethod attribute), 109
- description (biweeklybudget.interest.MinPaymentAmEx attribute), 109
- description (biweeklybudget.interest.MinPaymentCiti attribute), 110
- description (biweeklybudget.interest.MinPaymentDiscover attribute), 110
- description (biweeklybud-

`get.interest.MinPaymentMethod` (attribute), 110

`description` (*biweeklybudget.interest.SimpleInterest* attribute), 111

`description` (*biweeklybudget.models.account.Account* attribute), 79

`description` (*biweeklybudget.models.budget\_model.Budget* attribute), 83

`description` (*biweeklybudget.models.ofx\_transaction.OFXTransaction* attribute), 87

`description` (*biweeklybudget.models.scheduled\_transaction.ScheduledTransaction* attribute), 91

`description` (*biweeklybudget.models.transaction.Transaction* attribute), 92

`dict_to_class_args()` (in module *biweeklybudget.flaskapp.filters*), 76

`DISTANCE_UNIT` (in module *biweeklybudget.settings*), 117

`DISTANCE_UNIT_ABBREVIATION` (in module *biweeklybudget.settings*), 117

`do_budget_transfer()` (in module *biweeklybudget.models.utils*), 94

`do_screenshot()` (*biweeklybudget.screenscraper.ScreenScraper* method), 116

`doc_readystate_is_complete()` (*biweeklybudget.screenscraper.ScreenScraper* method), 116

`dollars_filter()` (in module *biweeklybudget.flaskapp.filters*), 76

`dtnow()` (in module *biweeklybudget.utils*), 119

`DuplicateFileException`, 95

## E

`effective_apr` (*biweeklybudget.models.account.Account* attribute), 79

`end_date` (*biweeklybudget.biweeklypayperiod.BiweeklyPayPeriod* attribute), 101

`end_date` (*biweeklybudget.interest.\_BillingPeriod* attribute), 111

`end_date` (*biweeklybudget.interest.CCStatement* attribute), 106

`engine` (in module *biweeklybudget.db*), 104

`error_screenshot()` (*biweeklybudget.screenscraper.ScreenScraper* method), 116

`ExampleView` (class in *biweeklybudget.flaskapp.views.example*), 58

## F

`file_mtime` (*biweeklybudget.models.ofx\_statement.OFXStatement* attribute), 86

`filename` (*biweeklybudget.models.ofx\_statement.OFXStatement* attribute), 86

`fill_location` (*biweeklybudget.models.fuel.FuelFill* attribute), 85

`filter_query()` (*biweeklybudget.biweeklypayperiod.BiweeklyPayPeriod* method), 102

`find_payments()` (*biweeklybudget.interest.\_PayoffMethod* method), 112

`find_payments()` (*biweeklybudget.interest.FixedPaymentMethod* method), 107

`find_payments()` (*biweeklybudget.interest.HighestBalanceFirstMethod* method), 107

`find_payments()` (*biweeklybudget.interest.HighestInterestRateFirstMethod* method), 107

`find_payments()` (*biweeklybudget.interest.LowestBalanceFirstMethod* method), 109

`find_payments()` (*biweeklybudget.interest.LowestInterestRateFirstMethod* method), 109

`find_payments()` (*biweeklybudget.interest.MinPaymentMethod* method), 110

`first_statement_by_date` (*biweeklybudget.models.ofx\_transaction.OFXTransaction* attribute), 87

`fitid` (*biweeklybudget.models.ofx\_transaction.OFXTransaction* attribute), 87

`fix_string()` (*biweeklybudget.flaskapp.views.formhandlerview.FormHandlerView* method), 59

`fix_werkzeug_logger()` (in module *biweeklybudget.utils*), 119

`FixedPaymentMethod` (class in *biweeklybudget.interest*), 107

`fmt_currency()` (built-in function), 124

`fmt_currency()` (in module *biweeklybudget.utils*), 119

`fmt_null()` (built-in function), 124

`for_ofxgetter` (*biweeklybudget.models.account.Account* attribute), 79

`FormBuilder()` (built-in function), 124

`FormBuilder.addCheckbox()` (*FormBuilder* method), 125

`FormBuilder.addCurrency()` (*FormBuilder*

- method*), 125
- FormBuilder.addDatePicker() (*FormBuilder method*), 125
- FormBuilder.addHidden() (*FormBuilder method*), 126
- FormBuilder.addHTML() (*FormBuilder method*), 125
- FormBuilder.addLabelToValueSelect() (*FormBuilder method*), 126
- FormBuilder.addP() (*FormBuilder method*), 126
- FormBuilder.addRadioInline() (*FormBuilder method*), 126
- FormBuilder.addSelect() (*FormBuilder method*), 127
- FormBuilder.addText() (*FormBuilder method*), 127
- FormBuilder.addTextArea() (*FormBuilder method*), 127
- FormBuilder.render() (*FormBuilder method*), 128
- FormHandlerView (*class in biweeklybudget.flaskapp.views.formhandlerview*), 58
- FUEL\_BUDGET\_ID (*in module biweeklybudget.settings*), 117
- FUEL\_BUDGET\_ID (*in module biweeklybudget.settings\_example*), 118
- FUEL\_ECO\_ABBREVIATION (*in module biweeklybudget.settings*), 117
- FUEL\_VOLUME\_ABBREVIATION (*in module biweeklybudget.settings*), 117
- FUEL\_VOLUME\_UNIT (*in module biweeklybudget.settings*), 117
- FuelAjax (*class in biweeklybudget.flaskapp.views.fuel*), 60
- FuelFill (*class in biweeklybudget.models.fuel*), 84
- fuellog (*biweeklybudget.models.fuel.Vehicle attribute*), 85
- FuelLogFormHandler (*class in biweeklybudget.flaskapp.views.fuel*), 61
- fuelLogModal() (*built-in function*), 129
- fuelModalDivForm() (*built-in function*), 130
- FuelMPGChartView (*class in biweeklybudget.flaskapp.views.fuel*), 61
- FuelPriceChartView (*class in biweeklybudget.flaskapp.views.fuel*), 61
- FuelView (*class in biweeklybudget.flaskapp.views.fuel*), 61
- G**
- gallons (*biweeklybudget.models.fuel.FuelFill attribute*), 85
- get() (*biweeklybudget.flaskapp.views.accounts.AccountAjax method*), 54
- get() (*biweeklybudget.flaskapp.views.accounts.AccountsView method*), 55
- get() (*biweeklybudget.flaskapp.views.accounts.OneAccountView method*), 55
- get() (*biweeklybudget.flaskapp.views.budgets.BudgetAjax method*), 56
- get() (*biweeklybudget.flaskapp.views.budgets.BudgetSpendingChartView method*), 56
- get() (*biweeklybudget.flaskapp.views.budgets.BudgetsView method*), 57
- get() (*biweeklybudget.flaskapp.views.budgets.OneBudgetView method*), 57
- get() (*biweeklybudget.flaskapp.views.credit\_payoffs.AccountOfxAjax method*), 57
- get() (*biweeklybudget.flaskapp.views.credit\_payoffs.CreditPayoffsView method*), 58
- get() (*biweeklybudget.flaskapp.views.example.ExampleView method*), 58
- get() (*biweeklybudget.flaskapp.views.fuel.FuelAjax method*), 60
- get() (*biweeklybudget.flaskapp.views.fuel.FuelMPGChartView method*), 61
- get() (*biweeklybudget.flaskapp.views.fuel.FuelPriceChartView method*), 61
- get() (*biweeklybudget.flaskapp.views.fuel.FuelView method*), 61
- get() (*biweeklybudget.flaskapp.views.fuel.VehicleAjax method*), 61
- get() (*biweeklybudget.flaskapp.views.help.HelpView method*), 62
- get() (*biweeklybudget.flaskapp.views.index.AcctBalanaceChartView method*), 62
- get() (*biweeklybudget.flaskapp.views.index.IndexView method*), 62
- get() (*biweeklybudget.flaskapp.views.ofx.OfxAccounts method*), 62
- get() (*biweeklybudget.flaskapp.views.ofx.OfxAjax method*), 63
- get() (*biweeklybudget.flaskapp.views.ofx.OfxTransAjax method*), 64
- get() (*biweeklybudget.flaskapp.views.ofx.OfxTransView method*), 64
- get() (*biweeklybudget.flaskapp.views.ofx.OfxView method*), 64
- get() (*biweeklybudget.flaskapp.views.payperiods.PayPeriodsView method*), 65
- get() (*biweeklybudget.flaskapp.views.payperiods.PayPeriodView method*), 64
- get() (*biweeklybudget.flaskapp.views.payperiods.PeriodForDateView method*), 65
- get() (*biweeklybudget.flaskapp.views.plaid.PlaidJs method*), 66
- get() (*biweeklybudget.flaskapp.views.plaid.PlaidUpdate method*), 67

get () (*biweeklybudget.flaskapp.views.projects.BoMItemAjaxView* method), 67

get () (*biweeklybudget.flaskapp.views.projects.BoMItemsAjaxView* method), 68

get () (*biweeklybudget.flaskapp.views.projects.BoMItemView* method), 68

get () (*biweeklybudget.flaskapp.views.projects.ProjectAjaxView* method), 68

get () (*biweeklybudget.flaskapp.views.projects.ProjectsAjaxView* method), 69

get () (*biweeklybudget.flaskapp.views.projects.ProjectsView* method), 69

get () (*biweeklybudget.flaskapp.views.reconcile.OfxUnreconciledAjaxView* method), 70

get () (*biweeklybudget.flaskapp.views.reconcile.ReconcileView* method), 70

get () (*biweeklybudget.flaskapp.views.reconcile.TransUnreconciledAjaxView* method), 70

get () (*biweeklybudget.flaskapp.views.reconcile.TxnReconcileAjaxView* method), 70

get () (*biweeklybudget.flaskapp.views.scheduled.OneScheduledAjaxView* method), 71

get () (*biweeklybudget.flaskapp.views.scheduled.ScheduledAjaxView* method), 71

get () (*biweeklybudget.flaskapp.views.scheduled.ScheduledTransView* method), 72

get () (*biweeklybudget.flaskapp.views.scheduled.ScheduledView* method), 72

get () (*biweeklybudget.flaskapp.views.searchableajaxview.SearchableAjaxView* method), 73

get () (*biweeklybudget.flaskapp.views.transactions.OneTransactionAjaxView* method), 73

get () (*biweeklybudget.flaskapp.views.transactions.OneTransactionView* method), 73

get () (*biweeklybudget.flaskapp.views.transactions.TransactionsAjaxView* method), 74

get () (*biweeklybudget.flaskapp.views.transactions.TransactionsView* method), 74

get () (*biweeklybudget.flaskapp.views.utils.DateTestJS* method), 74

get\_accounts () (*biweeklybudget.ofxapi.local.OfxApiLocal* method), 96

get\_accounts () (*biweeklybudget.ofxapi.remote.OfxApiRemote* method), 97

get\_browser () (*biweeklybudget.screenscraper.ScreenScraper* method), 116

get\_notifications () (*biweeklybudget.flaskapp.notifications.NotificationsController* static method), 78

get\_ofx () (*biweeklybudget.ofxgetter.OfxGetter* method), 114

getObjectValueKey () (built-in function), 135

handle () (*biweeklybudget.get.flaskapp.cli\_commands.CustomLoggingWSGIRequestHandler* method), 75

handle\_account\_re\_change () (in module *biweeklybudget.db\_event\_handlers*), 104

handle\_before\_flush () (in module *biweeklybudget.db\_event\_handlers*), 104

handle\_budget\_trans\_amount\_change () (in module *biweeklybudget.db\_event\_handlers*), 105

handle\_new\_or\_deleted\_budget\_transaction () (in module *biweeklybudget.db\_event\_handlers*), 105

handle\_ofx\_transaction\_new\_or\_change () (in module *biweeklybudget.db\_event\_handlers*), 105

handleForm () (built-in function), 128

handleFormError () (built-in function), 128

handleFormSubmitted () (built-in function), 128

handleInlineForm () (built-in function), 129

handleInlineFormError () (built-in function), 129

handleInlineFormSubmitted () (built-in function), 129

handleProjectAdded () (built-in function), 131

helpView (class in *biweeklybudget.get.flaskapp.views.help*), 62

highestBalanceFirstMethod (class in *biweeklybudget.interest*), 107

highestInterestRateFirstMethod (class in *biweeklybudget.interest*), 107

id (*biweeklybudget.models.account.Account* attribute), 79

id (*biweeklybudget.models.account\_balance.AccountBalance* attribute), 82

id (*biweeklybudget.models.budget\_model.Budget* attribute), 83

id (*biweeklybudget.models.budget\_transaction.BudgetTransaction* attribute), 83

id (*biweeklybudget.models.fuel.FuelFill* attribute), 85

id (*biweeklybudget.models.fuel.Vehicle* attribute), 85

id (*biweeklybudget.models.ofx\_statement.OFXStatement* attribute), 86

id (*biweeklybudget.models.projects.BoMItem* attribute), 89

id (*biweeklybudget.models.projects.Project* attribute), 90

id (*biweeklybudget.models.reconcile\_rule.ReconcileRule* attribute), 91

id (*biweeklybudget.models.scheduled\_transaction.ScheduledTransaction* attribute), 91

- `id` (*biweeklybudget.models.transaction.Transaction* attribute), 92
- `id` (*biweeklybudget.models.txn\_reconcile.TxnReconcile* attribute), 93
- `ignoreOfxTrans()` (built-in function), 131
- `ignoreOfxTransDivForm()` (built-in function), 132
- `in_directory()` (in module *biweeklybudget.utils*), 120
- `IndexView` (class in *biweeklybudget.flaskapp.views.index*), 62
- `init_db()` (in module *biweeklybudget.db*), 104
- `init_event_listeners()` (in module *biweeklybudget.db\_event\_handlers*), 105
- `institution_id` (*biweeklybudget.models.plaid\_items.PlaidItem* attribute), 89
- `institution_name` (*biweeklybudget.models.plaid\_items.PlaidItem* attribute), 89
- `interest` (*biweeklybudget.interest.CCStatement* attribute), 106
- `INTEREST_CALCULATION_NAMES` (in module *biweeklybudget.interest*), 108
- `interest_class_name` (*biweeklybudget.models.account.Account* attribute), 79
- `InterestHelper` (class in *biweeklybudget.interest*), 108
- `Investment` (*biweeklybudget.models.account.AcctType* attribute), 81
- `is_active` (*biweeklybudget.models.account.Account* attribute), 79
- `is_active` (*biweeklybudget.models.budget\_model.Budget* attribute), 83
- `is_active` (*biweeklybudget.models.fuel.Vehicle* attribute), 85
- `is_active` (*biweeklybudget.models.projects.BoMItem* attribute), 89
- `is_active` (*biweeklybudget.models.projects.Project* attribute), 90
- `is_active` (*biweeklybudget.models.reconcile\_rule.ReconcileRule* attribute), 91
- `is_active` (*biweeklybudget.models.scheduled\_transaction.ScheduledTransaction* attribute), 91
- `is_budget_source` (*biweeklybudget.models.account.Account* attribute), 79
- `is_in_past` (*biweeklybudget.models.biweeklypayperiod.BiweeklyPayPeriod* attribute), 102
- `is_income` (*biweeklybudget.models.budget\_model.Budget* attribute), 83
- `is_interest_charge` (*biweeklybudget.models.ofx\_transaction.OFXTransaction* attribute), 87
- `is_interest_payment` (*biweeklybudget.models.ofx\_transaction.OFXTransaction* attribute), 87
- `is_json` (*biweeklybudget.models.dbsetting.DBSetting* attribute), 84
- `is_late_fee` (*biweeklybudget.models.ofx\_transaction.OFXTransaction* attribute), 87
- `is_other_fee` (*biweeklybudget.models.ofx\_transaction.OFXTransaction* attribute), 87
- `is_payment` (*biweeklybudget.models.ofx\_transaction.OFXTransaction* attribute), 87
- `is_periodic` (*biweeklybudget.models.budget\_model.Budget* attribute), 83
- `is_stale` (*biweeklybudget.models.account.Account* attribute), 80
- `is_stale_data()` (in module *biweeklybudget.flaskapp.jinja\_tests*), 77
- `isFunction()` (built-in function), 129
- `isodate_filter()` (in module *biweeklybudget.flaskapp.filters*), 76
- `isoformat()` (built-in function), 124
- `item_id` (*biweeklybudget.models.plaid\_accounts.PlaidAccount* attribute), 89
- `item_id` (*biweeklybudget.models.plaid\_items.PlaidItem* attribute), 89
- ## J
- `jquery_finished()` (*biweeklybudget.screenscraper.ScreenScraper* method), 116
- ## L
- `last_interest_charge` (*biweeklybudget.models.account.Account* attribute), 80
- `last_updated` (*biweeklybudget.models.plaid\_items.PlaidItem* attribute), 89
- `ledger` (*biweeklybudget.models.account\_balance.AccountBalance* attribute), 82
- `ledger_bal` (*biweeklybudget.models.ofx\_statement.OFXStatement* attribute), 86

|                               |   |         |  |
|-------------------------------|---|---------|--|
| ledger_bal_as_of              | ( <i>biweeklybudget.models.ofx_statement.OFXStatement</i> attribute), 86                  | methods | ( <i>biweeklybudget.flaskapp.views.accounts.AccountTxfrFormHandler</i> attribute), 55          |
| ledger_date                   | ( <i>biweeklybudget.models.account_balance.AccountBalance</i> attribute), 82              | methods | ( <i>biweeklybudget.flaskapp.views.accounts.OneAccountView</i> attribute), 55                  |
| level_after                   | ( <i>biweeklybudget.models.fuel.FuelFill</i> attribute), 85                               | methods | ( <i>biweeklybudget.flaskapp.views.budgets.BudgetAjax</i> attribute), 56                       |
| level_before                  | ( <i>biweeklybudget.models.fuel.FuelFill</i> attribute), 85                               | methods | ( <i>biweeklybudget.flaskapp.views.budgets.BudgetFormHandler</i> attribute), 56                |
| line_cost                     | ( <i>biweeklybudget.models.projects.BoMItem</i> attribute), 89                            | methods | ( <i>biweeklybudget.flaskapp.views.budgets.BudgetFormHandler</i> attribute), 56                |
| load_cookies()                | ( <i>biweeklybudget.screenscraper.ScreenScraper</i> method), 116                          | methods | ( <i>biweeklybudget.flaskapp.views.budgets.BudgetSpendingChartView</i> attribute), 56          |
| loadSettings()                | ( <i>built-in function</i> ), 123   | methods | ( <i>biweeklybudget.flaskapp.views.budgets.BudgetsView</i> attribute), 57                      |
| LOCALE_NAME                   | (in module <i>biweeklybudget.settings</i> ), 117  | methods | ( <i>biweeklybudget.flaskapp.views.budgets.BudgetTxfrFormHandler</i> attribute), 56            |
| log_request()                 | ( <i>biweeklybudget.flaskapp.cli_commands.CustomLoggingWSGIRequestHandler</i> method), 75 | methods | ( <i>biweeklybudget.flaskapp.views.budgets.BudgetTxfrFormHandler</i> attribute), 56            |
| LowestBalanceFirstMethod      | (class in <i>biweeklybudget.interest</i> ), 108   | methods | ( <i>biweeklybudget.flaskapp.views.budgets.OneBudgetView</i> attribute), 57                    |
| LowestInterestRateFirstMethod | (class in <i>biweeklybudget.interest</i> ), 109   | methods | ( <i>biweeklybudget.flaskapp.views.credit_payoffs.AccountOfxAjax</i> attribute), 57            |
| <b>M</b>                      |   |         |  |
| MagicJSONEncoder              | (class in <i>biweeklybudget.flaskapp.jsonencoder</i> ), 77                                | methods | ( <i>biweeklybudget.flaskapp.views.credit_payoffs.AccountOfxFoxFormHandler</i> attribute), 57  |
| main()                        | (in module <i>biweeklybudget.backfill_ofx</i> ), 98                                       | methods | ( <i>biweeklybudget.flaskapp.views.credit_payoffs.CreditPayoffsView</i> attribute), 58         |
| main()                        | (in module <i>biweeklybudget.initdb</i> ), 106  | methods | ( <i>biweeklybudget.flaskapp.views.credit_payoffs.PayoffSettingsFormHandler</i> attribute), 58 |
| main()                        | (in module <i>biweeklybudget.load_data</i> ), 113   | methods | ( <i>biweeklybudget.flaskapp.views.example.ExampleView</i> attribute), 58                      |
| main()                        | (in module <i>biweeklybudget.ofxgetter</i> ), 114   | methods | ( <i>biweeklybudget.flaskapp.views.formhandlerview.FormHandlerView</i> attribute), 59          |
| main()                        | (in module <i>biweeklybudget.wishlist2project</i> ), 121                                  | methods | ( <i>biweeklybudget.flaskapp.views.fuel.FuelAjax</i> attribute), 61                            |
| makeTransFromOfx()            | ( <i>built-in function</i> ), 132   | methods | ( <i>biweeklybudget.flaskapp.views.fuel.FuelLogFormHandler</i> attribute), 61                  |
| makeTransSaveCallback()       | ( <i>built-in function</i> ), 132   | methods | ( <i>biweeklybudget.flaskapp.views.fuel.FuelMPGChartView</i> attribute), 61                    |
| mask                          | ( <i>biweeklybudget.models.plaid_accounts.PlaidAccount</i> attribute), 89                 | methods | ( <i>biweeklybudget.flaskapp.views.fuel.FuelPriceChartView</i> attribute), 61                  |
| max_total_for_period()        | ( <i>biweeklybudget.interest._PayoffMethod</i> method), 112                               | methods |  |
| mcc                           | ( <i>biweeklybudget.models.ofx_transaction.OFXTransaction</i> attribute), 87              | methods |  |
| memo                          | ( <i>biweeklybudget.models.ofx_transaction.OFXTransaction</i> attribute), 87              | methods |  |
| methods                       | ( <i>biweeklybudget.flaskapp.views.accounts.AccountAjax</i> attribute), 54                | methods |  |
| methods                       | ( <i>biweeklybudget.flaskapp.views.accounts.AccountFormHandler</i> attribute), 55         | methods |  |
| methods                       | ( <i>biweeklybudget.flaskapp.views.accounts.AccountsView</i> attribute), 55               | methods |  |

|  |   |
|--|---|
| methods ( <i>biweeklybudget.flaskapp.views.fuel.FuelView</i> attribute), 61                        | <i>get.flaskapp.views.plaid.PlaidLinkToken</i> attribute), 66                                 |
| methods ( <i>biweeklybudget.flaskapp.views.fuel.VehicleAjax</i> attribute), 61                     | methods ( <i>biweeklybudget.flaskapp.views.plaid.PlaidRefreshAccounts</i> attribute), 66      |
| methods ( <i>biweeklybudget.flaskapp.views.fuel.VehicleFormHandler</i> attribute), 62              | methods ( <i>biweeklybudget.flaskapp.views.plaid.PlaidUpdate</i> attribute), 67               |
| methods ( <i>biweeklybudget.flaskapp.views.help.HelpView</i> attribute), 62                        | methods ( <i>biweeklybudget.flaskapp.views.plaid.PlaidUpdateItemInfo</i> attribute), 67       |
| methods ( <i>biweeklybudget.flaskapp.views.index.AcctBalanceChartView</i> attribute), 62           | methods ( <i>biweeklybudget.flaskapp.views.projects.BoMItemAjax</i> attribute), 67            |
| methods ( <i>biweeklybudget.flaskapp.views.index.IndexView</i> attribute), 62                      | methods ( <i>biweeklybudget.flaskapp.views.projects.BoMItemFormHandler</i> attribute), 67     |
| methods ( <i>biweeklybudget.flaskapp.views.ofx.OfxAccounts</i> attribute), 63                      | methods ( <i>biweeklybudget.flaskapp.views.projects.BoMItemsAjax</i> attribute), 68           |
| methods ( <i>biweeklybudget.flaskapp.views.ofx.OfxAjax</i> attribute), 63                          | methods ( <i>biweeklybudget.flaskapp.views.projects.BoMItemView</i> attribute), 68            |
| methods ( <i>biweeklybudget.flaskapp.views.ofx.OfxStatementPost</i> attribute), 63                 | methods ( <i>biweeklybudget.flaskapp.views.projects.ProjectAjax</i> attribute), 68            |
| methods ( <i>biweeklybudget.flaskapp.views.ofx.OfxTransAjax</i> attribute), 64                     | methods ( <i>biweeklybudget.flaskapp.views.projects.ProjectsAjax</i> attribute), 69           |
| methods ( <i>biweeklybudget.flaskapp.views.ofx.OfxTransView</i> attribute), 64                     | methods ( <i>biweeklybudget.flaskapp.views.projects.ProjectsFormHandler</i> attribute), 69    |
| methods ( <i>biweeklybudget.flaskapp.views.ofx.OfxView</i> attribute), 64                          | methods ( <i>biweeklybudget.flaskapp.views.projects.ProjectsView</i> attribute), 69           |
| methods ( <i>biweeklybudget.flaskapp.views.payperiods.PayPeriodView</i> attribute), 65             | methods ( <i>biweeklybudget.flaskapp.views.reconcile.OfxUnreconciledAjax</i> attribute), 70   |
| methods ( <i>biweeklybudget.flaskapp.views.payperiods.PayPeriodView</i> attribute), 64             | methods ( <i>biweeklybudget.flaskapp.views.reconcile.ReconcileAjax</i> attribute), 70         |
| methods ( <i>biweeklybudget.flaskapp.views.payperiods.PeriodForDateView</i> attribute), 65         | methods ( <i>biweeklybudget.flaskapp.views.reconcile.ReconcileView</i> attribute), 70         |
| methods ( <i>biweeklybudget.flaskapp.views.payperiods.SchedToTransFormHandler</i> attribute), 65   | methods ( <i>biweeklybudget.flaskapp.views.reconcile.TransUnreconciledAjax</i> attribute), 70 |
| methods ( <i>biweeklybudget.flaskapp.views.payperiods.SkipSchedTransFormHandler</i> attribute), 65 | methods ( <i>biweeklybudget.flaskapp.views.reconcile.TxnReconcileAjax</i> attribute), 70      |
| methods ( <i>biweeklybudget.flaskapp.views.plaid.PlaidHandleLink</i> attribute), 66                | methods ( <i>biweeklybudget.flaskapp.views.scheduled.OneScheduledAjax</i> attribute), 71      |
| methods ( <i>biweeklybudget.flaskapp.views.plaid.PlaidJs</i> attribute), 66                        | methods ( <i>biweeklybudget</i> attribute), 71  |
| methods ( <i>biweeklybudget</i> attribute), 66   | methods ( <i>biweeklybudget</i> attribute), 71  |

|                           |   |      |  |
|---------------------------|---|------|--|
|                           | <code>get.flaskapp.views.scheduled.SchedTransFormHandler</code>                       | name | <code>(biweeklybudget.models.account.Account attribute)</code> , 71  |
| methods                   | <code>(biweeklybudget.get.flaskapp.views.scheduled.ScheduledAjax</code>               | name | <code>(biweeklybudget.models.budget_model.Budget attribute)</code> , 80  |
|                           | <code>attribute)</code> , 71  |      |  |
| methods                   | <code>(biweeklybudget.get.flaskapp.views.scheduled.ScheduledTransView</code>          | name | <code>(biweeklybudget.models.dbsetting.DBSetting attribute)</code> , 84  |
|                           | <code>attribute)</code> , 72  |      |  |
| methods                   | <code>(biweeklybudget.get.flaskapp.views.scheduled.ScheduledView</code>               | name | <code>(biweeklybudget.models.fuel.Vehicle attribute)</code> , 85   |
|                           | <code>attribute)</code> , 72  | name | <code>(biweeklybudget.models.ofx_transaction.OFXTransaction attribute)</code> , 87                                   |
| methods                   | <code>(biweeklybudget.get.flaskapp.views.searchableajaxview.SearchableAjaxView</code> | name | <code>(biweeklybudget.models.plaid_accounts.PlaidAccount attribute)</code> , 89                                      |
|                           | <code>attribute)</code> , 73  | name | <code>(biweeklybudget.models.projects.BoMItem attribute)</code> , 90   |
| methods                   | <code>(biweeklybudget.get.flaskapp.views.transactions.OneTransactionAjax</code>       | name | <code>(biweeklybudget.models.projects.Project attribute)</code> , 90   |
|                           | <code>attribute)</code> , 73  | name | <code>(biweeklybudget.models.reconcile_rule.ReconcileRule attribute)</code> , 91                                     |
| methods                   | <code>(biweeklybudget.get.flaskapp.views.transactions.OneTransactionView</code>       |      | <code>negate_ofx_amounts (biweeklybudget.get.models.account.Account attribute)</code> , 80                           |
|                           | <code>attribute)</code> , 73  |      | <code>next (biweeklybudget.biweeklypayperiod.BiweeklyPayPeriod attribute)</code> , 102                               |
| methods                   | <code>(biweeklybudget.get.flaskapp.views.transactions.TransactionFormHandler</code>   |      | <code>next_period (biweeklybudget.interest._BillingPeriod attribute)</code> , 111                                    |
|                           | <code>attribute)</code> , 73  |      | <code>next_with_transactions() (biweeklybudget.get.interest.CCStatement method)</code> , 106                         |
| methods                   | <code>(biweeklybudget.get.flaskapp.views.transactions.TransactionsAjax</code>         |      | <code>nextIndex() (built-in function)</code> , 123   |
|                           | <code>attribute)</code> , 74  |      | <code>NoInterestChargedError</code> , 82   |
| methods                   | <code>(biweeklybudget.get.flaskapp.views.transactions.TransactionsView</code>         |      | <code>note (biweeklybudget.models.txn_reconcile.TxnReconcile attribute)</code> , 94                                  |
|                           | <code>attribute)</code> , 74  |      | <code>notes (biweeklybudget.models.fuel.FuelFill attribute)</code> , 85  |
| methods                   | <code>(biweeklybudget.get.flaskapp.views.utils.DateTestJS</code>                      |      | <code>notes (biweeklybudget.models.ofx_transaction.OFXTransaction attribute)</code> , 88                             |
|                           | <code>attribute)</code> , 74  |      | <code>notes (biweeklybudget.models.projects.BoMItem attribute)</code> , 90   |
| min_payment_class_name    | <code>(biweeklybudget.get.models.account.Account attribute)</code> , 80               |      | <code>notes (biweeklybudget.models.projects.Project attribute)</code> , 90   |
| MIN_PAYMENT_FORMULA_NAMES | <i>(in module biweeklybudget.interest)</i> , 109                                      |      | <code>notes (biweeklybudget.models.scheduled_transaction.ScheduledTransaction attribute)</code> , 91                 |
| min_payments              | <code>(biweeklybudget.get.interest.InterestHelper attribute)</code> , 108             |      | <code>notes (biweeklybudget.models.transaction.Transaction attribute)</code> , 93                                    |
| minimum_payment           | <code>(biweeklybudget.get.interest.CCStatement attribute)</code> , 106                |      | <code>notifications() (in module biweeklybudget.flaskapp.context_processors)</code> , 75                             |
| MinPaymentAmEx            | <i>(class in biweeklybudget.interest)</i> , 109                                       |      | <code>NotificationsController (class in biweeklybudget.flaskapp.notifications)</code> , 78                           |
| MinPaymentCiti            | <i>(class in biweeklybudget.interest)</i> , 109                                       |      | <code>num_per_period (biweeklybudget.get.models.scheduled_transaction.ScheduledTransaction attribute)</code> , 91    |
| MinPaymentDiscover        | <i>(class in biweeklybudget.interest)</i> , 110                                       |      | <code>num_stale_accounts() (biweeklybudget.flaskapp.notifications.NotificationsController static method)</code> , 78 |
| MinPaymentMethod          | <i>(class in biweeklybudget.interest)</i> , 110                                       |      | <code>num_unreconciled_ofx() (biweeklybudget.flaskapp.notifications.NotificationsController</code>                   |
| ModelAsDict               | <i>(class in biweeklybudget.models.base)</i> , 82                                     |      |  |
| monthsyears()             | <i>(in module biweeklybudget.flaskapp.filters)</i> , 76                               |      |  |

*static method*), 78

## O

odometer\_miles (biweeklybudget.models.fuel.FuelFill attribute), 85

ofx\_account\_id (biweeklybudget.models.txn\_reconcile.TxnReconcile attribute), 94

ofx\_cat\_memo\_to\_name (biweeklybudget.models.account.Account attribute), 80

ofx\_fitid (biweeklybudget.models.txn\_reconcile.TxnReconcile attribute), 94

ofx\_statement (biweeklybudget.models.account.Account attribute), 80

ofx\_trans (biweeklybudget.models.ofx\_statement.OFXStatement attribute), 86

ofx\_trans (biweeklybudget.models.txn\_reconcile.TxnReconcile attribute), 94

OfxAccounts (class in biweeklybudget.flaskapp.views.ofx), 62

OfxAjax (class in biweeklybudget.flaskapp.views.ofx), 63

OfxApiLocal (class in biweeklybudget.ofxapi.local), 95

OfxApiRemote (class in biweeklybudget.ofxapi.remote), 96

OfxBackfiller (class in biweeklybudget.backfill\_ofx), 97

OfxGetter (class in biweeklybudget.ofxgetter), 113

ofxgetter\_config (biweeklybudget.models.account.Account attribute), 80

ofxgetter\_config\_json (biweeklybudget.models.account.Account attribute), 80

OFXStatement (class in biweeklybudget.models.ofx\_statement), 86

OfxStatementPost (class in biweeklybudget.flaskapp.views.ofx), 63

OFXTransaction (class in biweeklybudget.models.ofx\_transaction), 87

OfxTransAjax (class in biweeklybudget.flaskapp.views.ofx), 64

ofxTransModal() (built-in function), 130

OfxTransView (class in biweeklybudget.flaskapp.views.ofx), 64

OfxUnreconciledAjax (class in biweeklybudget.flaskapp.views.reconcile), 70

OfxView (class in biweeklybudget.flaskapp.views.ofx), 64

omit\_from\_graphs (biweeklybudget.models.budget\_model.Budget attribute), 83

OneAccountView (class in biweeklybudget.flaskapp.views.accounts), 55

OneBudgetView (class in biweeklybudget.flaskapp.views.budgets), 57

OneScheduledAjax (class in biweeklybudget.flaskapp.views.scheduled), 70

OneTransactionAjax (class in biweeklybudget.flaskapp.views.transactions), 73

OneTransactionView (class in biweeklybudget.flaskapp.views.transactions), 73

Other (biweeklybudget.models.account.AcctType attribute), 81

overall\_date (biweeklybudget.models.account\_balance.AccountBalance attribute), 82

overall\_sums (biweeklybudget.biweeklypayerperiod.BiweeklyPayPeriod attribute), 102

## P

params\_from\_ofxparser\_transaction() (biweeklybudget.models.ofx\_transaction.OFXTransaction static method), 88

parse\_args() (in module biweeklybudget.backfill\_ofx), 98

parse\_args() (in module biweeklybudget.initdb), 106

parse\_args() (in module biweeklybudget.load\_data), 113

parse\_args() (in module biweeklybudget.ofxgetter), 114

parse\_args() (in module biweeklybudget.wishlist2project), 121

pay() (biweeklybudget.interest.CCStatement method), 107

PAY\_PERIOD\_START\_DATE (in module biweeklybudget.settings), 117

PAY\_PERIOD\_START\_DATE (in module biweeklybudget.settings\_example), 118

payment\_date (biweeklybudget.interest.\_BillingPeriod attribute), 111

PAYOFF\_METHOD\_NAMES (in module biweeklybudget.interest), 110

PayoffSettingsFormHandler (class in biweeklybudget.flaskapp.views.credit\_payoffs), 58

PayPeriodsView (class in biweeklybudget.flaskapp.views.payperiods), 65

PayPeriodView (class in biweeklybudget.flaskapp.views.payperiods), 64

period\_for\_date() (biweeklybudget.biweeklypayerperiod.BiweeklyPayPeriod static method), 102

period\_interval (biweeklybudget.biweeklypayerperiod.BiweeklyPayPeriod attribute), 102

[period\\_length](#) (*biweeklybudget.get.biweeklypayperiod.BiweeklyPayPeriod* attribute), 102  
[period\\_panel\\_color\\_filter\(\)](#) (in module *biweeklybudget.flaskapp.filters*), 76  
[PeriodForDateView](#) (class in *biweeklybudget.flaskapp.views.payperiods*), 65  
[plaid\\_account](#) (*biweeklybudget.get.models.account.Account* attribute), 80  
[plaid\\_account\\_id](#) (*biweeklybudget.get.models.account.Account* attribute), 80  
[plaid\\_client\(\)](#) (in module *biweeklybudget.utils*), 120  
[PLAID\\_CLIENT\\_ID](#) (in module *biweeklybudget.get.settings*), 117  
[PLAID\\_CLIENT\\_ID](#) (in module *biweeklybudget.get.settings\_example*), 118  
[plaid\\_configured](#) (*biweeklybudget.get.models.account.Account* attribute), 80  
[PLAID\\_COUNTRY\\_CODES](#) (in module *biweeklybudget.get.settings*), 117  
[PLAID\\_COUNTRY\\_CODES](#) (in module *biweeklybudget.get.settings\_example*), 118  
[PLAID\\_ENV](#) (in module *biweeklybudget.settings*), 117  
[PLAID\\_ENV](#) (in module *biweeklybudget.get.settings\_example*), 118  
[plaid\\_item](#) (*biweeklybudget.get.models.plaid\_accounts.PlaidAccount* attribute), 89  
[plaid\\_item\\_id](#) (*biweeklybudget.get.models.account.Account* attribute), 80  
[PLAID\\_PRODUCTS](#) (in module *biweeklybudget.get.settings*), 118  
[PLAID\\_PRODUCTS](#) (in module *biweeklybudget.get.settings\_example*), 118  
[PLAID\\_SECRET](#) (in module *biweeklybudget.settings*), 118  
[PLAID\\_SECRET](#) (in module *biweeklybudget.get.settings\_example*), 119  
[PLAID\\_USER\\_ID](#) (in module *biweeklybudget.settings*), 118  
[PLAID\\_USER\\_ID](#) (in module *biweeklybudget.get.settings\_example*), 119  
[PlaidAccount](#) (class in *biweeklybudget.get.models.plaid\_accounts*), 88  
[PlaidHandleLink](#) (class in *biweeklybudget.flaskapp.views.plaid*), 66  
[PlaidItem](#) (class in *biweeklybudget.get.models.plaid\_items*), 89  
[PlaidJs](#) (class in *biweeklybudget.flaskapp.views.plaid*), 66  
[plaidLink\(\)](#) (built-in function), 131  
[PlaidLinkToken](#) (class in *biweeklybudget.flaskapp.views.plaid*), 66  
[plaidRefresh\(\)](#) (built-in function), 131  
[PlaidRefreshAccounts](#) (class in *biweeklybudget.flaskapp.views.plaid*), 66  
[PlaidUpdate](#) (class in *biweeklybudget.flaskapp.views.plaid*), 66  
[plaidUpdate\(\)](#) (built-in function), 131  
[PlaidUpdateItemInfo](#) (class in *biweeklybudget.flaskapp.views.plaid*), 67  
[PlaidUpdater](#) (class in *biweeklybudget.get.plaid\_updater*), 114  
[PlaidUpdateResult](#) (class in *biweeklybudget.get.plaid\_updater*), 114  
[planned\\_budget](#) (*biweeklybudget.get.models.transaction.Transaction* attribute), 93  
[planned\\_budget\\_id](#) (*biweeklybudget.get.models.transaction.Transaction* attribute), 93  
[planned\\_transactions](#) (*biweeklybudget.get.models.budget\_model.Budget* attribute), 83  
[pluralize\\_filter\(\)](#) (in module *biweeklybudget.flaskapp.filters*), 76  
[post\(\)](#) (*biweeklybudget.flaskapp.views.credit\_payoffs.PayoffSettingsFormHandler* method), 58  
[post\(\)](#) (*biweeklybudget.flaskapp.views.formhandlerview.FormHandlerView* method), 60  
[post\(\)](#) (*biweeklybudget.flaskapp.views.ofx.OfxStatementPost* method), 63  
[post\(\)](#) (*biweeklybudget.flaskapp.views.plaid.PlaidHandleLink* method), 66  
[post\(\)](#) (*biweeklybudget.flaskapp.views.plaid.PlaidLinkToken* method), 66  
[post\(\)](#) (*biweeklybudget.flaskapp.views.plaid.PlaidRefreshAccounts* method), 66  
[post\(\)](#) (*biweeklybudget.flaskapp.views.plaid.PlaidUpdate* method), 67  
[post\(\)](#) (*biweeklybudget.flaskapp.views.plaid.PlaidUpdateItemInfo* method), 67  
[post\(\)](#) (*biweeklybudget.flaskapp.views.reconcile.ReconcileAjax* method), 70  
[pp\\_sum\(\)](#) (*biweeklybudget.flaskapp.notifications.NotificationsController* static method), 78  
[prev\\_period](#) (*biweeklybudget.interest.\_BillingPeriod*

- attribute), 111
- previous (biweeklybudget.get.biweeklypayperiod.BiweeklyPayPeriod attribute), 103
- prime\_rate (biweeklybudget.get.prime\_rate.PrimeRateCalculator attribute), 115
- prime\_rate\_margin (biweeklybudget.get.models.account.Account attribute), 80
- PrimeRateCalculator (class in biweeklybudget.get.prime\_rate), 115
- principal (biweeklybudget.interest.CCStatement attribute), 107
- project (biweeklybudget.models.projects.BoMItem attribute), 90
- Project (class in biweeklybudget.models.projects), 90
- project\_id (biweeklybudget.get.models.projects.BoMItem attribute), 90
- ProjectAjax (class in biweeklybudget.flaskapp.views.projects), 68
- ProjectsAjax (class in biweeklybudget.flaskapp.views.projects), 68
- ProjectsFormHandler (class in biweeklybudget.flaskapp.views.projects), 69
- ProjectsView (class in biweeklybudget.flaskapp.views.projects), 69
- ## Q
- quantity (biweeklybudget.models.projects.BoMItem attribute), 90
- query\_profile\_after() (in module biweeklybudget.db\_event\_handlers), 105
- query\_profile\_before() (in module biweeklybudget.db\_event\_handlers), 105
- ## R
- re\_interest\_charge (biweeklybudget.get.models.account.Account attribute), 81
- re\_interest\_paid (biweeklybudget.get.models.account.Account attribute), 81
- re\_late\_fee (biweeklybudget.get.models.account.Account attribute), 81
- re\_other\_fee (biweeklybudget.get.models.account.Account attribute), 81
- re\_payment (biweeklybudget.models.account.Account attribute), 81
- read() (biweeklybudget.vault.Vault method), 120
- recalcPayoffs() (built-in function), 124
- reconcile (biweeklybudget.get.models.ofx\_transaction.OFXTransaction attribute), 88
- reconcile (biweeklybudget.get.models.transaction.Transaction attribute), 93
- RECONCILE\_BEGIN\_DATE (in module biweeklybudget.get.settings), 118
- RECONCILE\_BEGIN\_DATE (in module biweeklybudget.get.settings\_example), 119
- reconcile\_id (biweeklybudget.get.models.ofx\_transaction.OFXTransaction attribute), 88
- reconcile\_trans (biweeklybudget.get.models.account.Account attribute), 81
- ReconcileAjax (class in biweeklybudget.flaskapp.views.reconcile), 70
- reconciled\_at (biweeklybudget.get.models.txn\_reconcile.TxnReconcile attribute), 94
- reconcileDoUnreconcile() (built-in function), 132
- reconcileDoUnreconcileNoOfx() (built-in function), 132
- reconcileDoUnreconcileNoTrans() (built-in function), 132
- reconcileGetOFX() (built-in function), 132
- reconcileGetTransactions() (built-in function), 133
- reconcileHandleSubmit() (built-in function), 133
- reconcileOfxDiv() (built-in function), 133
- reconcileOfxNoTrans() (built-in function), 133
- ReconcileRule (class in biweeklybudget.get.models.reconcile\_rule), 91
- reconcileShowOFX() (built-in function), 133
- reconcileShowTransactions() (built-in function), 133
- reconcileTransactions() (built-in function), 134
- reconcileTransDiv() (built-in function), 133
- reconcileTransDroppableAccept() (built-in function), 133
- reconcileTransHandleDropEvent() (built-in function), 133
- reconcileTransNoOfx() (built-in function), 134
- ReconcileView (class in biweeklybudget.flaskapp.views.reconcile), 70
- recurrence\_str (biweeklybudget.get.models.scheduled\_transaction.ScheduledTransaction attribute), 91
- reddollars\_filter() (in module biweeklybudget.flaskapp.filters), 77
- reloadProject() (built-in function), 122
- remaining\_cost (biweeklybudget.get.models.projects.Project attribute), 90
- removeIncrease() (built-in function), 124
- removeOnetime() (built-in function), 124
- reported\_miles (biweeklybud-

- `get.models.fuel.FuelFill` attribute), 85
  - `reported_mpg` (*biweeklybudget.models.fuel.FuelFill* attribute), 85
  - `routing_number` (*biweeklybudget.models.ofx\_statement.OFXStatement* attribute), 86
  - `rule` (*biweeklybudget.models.txn\_reconcile.TxnReconcile* attribute), 94
  - `rule_id` (*biweeklybudget.models.txn\_reconcile.TxnReconcile* attribute), 94
  - `run()` (*biweeklybudget.backfill\_ofx.OfxBackfiller* method), 98
  - `run()` (*biweeklybudget.wishlist2project.WishlistToProject* method), 121
- S**
- `save_cookies()` (*biweeklybudget.screenscraper.ScreenScraper* method), 116
  - `schedModal()` (*built-in function*), 135
  - `schedModalDivFillAndShow()` (*built-in function*), 135
  - `schedModalDivForm()` (*built-in function*), 135
  - `schedModalDivHandleType()` (*built-in function*), 135
  - `SchedToTransFormHandler` (*class in biweeklybudget.flaskapp.views.payperiods*), 65
  - `schedToTransModal()` (*built-in function*), 130
  - `schedToTransModalDivFillAndShow()` (*built-in function*), 130
  - `schedToTransModalDivForm()` (*built-in function*), 130
  - `SchedTransFormHandler` (*class in biweeklybudget.flaskapp.views.scheduled*), 71
  - `schedule_type` (*biweeklybudget.models.scheduled\_transaction.ScheduledTransaction* attribute), 92
  - `scheduled_trans` (*biweeklybudget.models.transaction.Transaction* attribute), 93
  - `scheduled_trans_id` (*biweeklybudget.models.transaction.Transaction* attribute), 93
  - `scheduled_transactions` (*biweeklybudget.models.budget\_model.Budget* attribute), 83
  - `ScheduledAjax` (*class in biweeklybudget.flaskapp.views.scheduled*), 71
  - `ScheduledTransaction` (*class in biweeklybudget.models.scheduled\_transaction*), 91
  - `ScheduledTransView` (*class in biweeklybudget.flaskapp.views.scheduled*), 72
  - `ScheduledView` (*class in biweeklybudget.flaskapp.views.scheduled*), 72
  - `ScreenScraper` (*class in biweeklybudget.screenscraper*), 116
  - `SearchableAjaxView` (*class in biweeklybudget.flaskapp.views.searchableajaxview*), 72
  - `SecretMissingException`, 120
  - `selectBudget()` (*built-in function*), 135
  - `send_response()` (*biweeklybudget.flaskapp.cli\_commands.CustomLoggingWSGIRequestHandler* method), 75
  - `serializeForm()` (*built-in function*), 129
  - `serializeForms()` (*built-in function*), 124
  - `set_balance()` (*biweeklybudget.models.account.Account* method), 81
  - `set_budget_amounts()` (*biweeklybudget.models.transaction.Transaction* method), 93
  - `set_log_debug()` (*in module biweeklybudget.cliutils*), 103
  - `set_log_info()` (*in module biweeklybudget.cliutils*), 103
  - `set_log_level_format()` (*in module biweeklybudget.cliutils*), 103
  - `set_ofxgetter_config()` (*biweeklybudget.models.account.Account* method), 81
  - `set_url_rules()` (*in module biweeklybudget.flaskapp.views.plaid*), 67
  - `set_url_rules()` (*in module biweeklybudget.flaskapp.views.utils*), 74
  - `setChanged()` (*built-in function*), 124
  - `settings()` (*in module biweeklybudget.flaskapp.context\_processors*), 75
  - `show_in_ui` (*biweeklybudget.interest.FixedPaymentMethod* attribute), 107
  - `show_in_ui` (*biweeklybudget.interest.HighestBalanceFirstMethod* attribute), 107
  - `show_in_ui` (*biweeklybudget.interest.HighestInterestRateFirstMethod* attribute), 108
  - `show_in_ui` (*biweeklybudget.interest.LowestBalanceFirstMethod* attribute), 109
  - `show_in_ui` (*biweeklybudget.interest.LowestInterestRateFirstMethod* attribute), 109
  - `show_in_ui` (*biweeklybudget.interest.MinPaymentMethod* attribute), 110
  - `shutdown_session()` (*in module biweeklybudget.flaskapp.app*), 75
  - `sic` (*biweeklybudget.models.ofx\_transaction.OFXTransaction*

- attribute*), 88
- SimpleInterest (*class in biweeklybudget.interest*), 110
- SkipSchedTransFormHandler (*class in biweeklybudget.flaskapp.views.payperiods*), 65
- skipSchedTransModal() (*built-in function*), 130
- skipSchedTransModalDivFillAndShow() (*built-in function*), 131
- skipSchedTransModalDivForm() (*built-in function*), 131
- STALE\_DATA\_TIMEDELTA (*in module biweeklybudget.settings*), 118
- STALE\_DATA\_TIMEDELTA (*in module biweeklybudget.settings\_example*), 119
- standing\_budgets\_sum() (*biweeklybudget.flaskapp.notifications.NotificationsController static method*), 78
- start\_date (*biweeklybudget.biweeklypayperiod.BiweeklyPayPeriod attribute*), 103
- start\_date (*biweeklybudget.interest.\_BillingPeriod attribute*), 111
- start\_date (*biweeklybudget.interest.CCStatement attribute*), 107
- starting\_balance (*biweeklybudget.models.budget\_model.Budget attribute*), 83
- statement (*biweeklybudget.models.ofx\_transaction.OFXTransaction attribute*), 88
- statement\_id (*biweeklybudget.models.ofx\_transaction.OFXTransaction attribute*), 88
- STATEMENTS\_SAVE\_PATH (*in module biweeklybudget.settings*), 118
- STATEMENTS\_SAVE\_PATH (*in module biweeklybudget.settings\_example*), 119
- subclass\_dict() (*in module biweeklybudget.interest*), 113
- submit() (*biweeklybudget.flaskapp.views.accounts.AccountFormHandler method*), 55
- submit() (*biweeklybudget.flaskapp.views.accounts.AccountTxfrFormHandler method*), 55
- submit() (*biweeklybudget.flaskapp.views.budgets.BudgetFormHandler method*), 56
- submit() (*biweeklybudget.flaskapp.views.budgets.BudgetTxfrFormHandler method*), 56
- submit() (*biweeklybudget.flaskapp.views.credit\_payoffs.AccountOfxFormHandler method*), 57
- submit() (*biweeklybudget.get.flaskapp.views.formhandlerview.FormHandlerView method*), 60
- submit() (*biweeklybudget.get.flaskapp.views.fuel.FuelLogFormHandler method*), 61
- submit() (*biweeklybudget.get.flaskapp.views.fuel.VehicleFormHandler method*), 62
- submit() (*biweeklybudget.get.flaskapp.views.payperiods.SchedToTransFormHandler method*), 65
- submit() (*biweeklybudget.get.flaskapp.views.payperiods.SkipSchedTransFormHandler method*), 65
- submit() (*biweeklybudget.get.flaskapp.views.projects.BoMItemFormHandler method*), 67
- submit() (*biweeklybudget.get.flaskapp.views.projects.ProjectsFormHandler method*), 69
- submit() (*biweeklybudget.get.flaskapp.views.scheduled.SchedTransFormHandler method*), 71
- submit() (*biweeklybudget.get.flaskapp.views.transactions.TransactionFormHandler method*), 73
- suffix\_for\_period() (*biweeklybudget.get.flaskapp.views.payperiods.PayPeriodView method*), 64
- ## T
- template\_paths() (*in module biweeklybudget.flaskapp.cli\_commands*), 75
- TOKEN\_PATH (*in module biweeklybudget.settings*), 118
- TOKEN\_PATH (*in module biweeklybudget.settings\_example*), 119
- total\_cost (*biweeklybudget.models.fuel.FuelFill attribute*), 85
- total\_cost (*biweeklybudget.models.projects.Project attribute*), 90
- trans\_id (*biweeklybudget.models.budget\_transaction.BudgetTransaction attribute*), 84
- trans\_type (*biweeklybudget.models.ofx\_transaction.OFXTransaction attribute*), 88
- transaction (*biweeklybudget.models.budget\_transaction.BudgetTransaction attribute*), 84
- transaction (*biweeklybudget.models.txn\_reconcile.TxnReconcile attribute*), 94

Transaction (class in *biweeklybudget.models.transaction*), 92

TransactionFormHandler (class in *biweeklybudget.flaskapp.views.transactions*), 73

transactions (*biweeklybudget.models.scheduled\_transaction.ScheduledTransaction* attribute), 92

transactions\_list (*biweeklybudget.biweeklypayperiod.BiweeklyPayPeriod* attribute), 103

TransactionsAjax (class in *biweeklybudget.flaskapp.views.transactions*), 73

TransactionsView (class in *biweeklybudget.flaskapp.views.transactions*), 74

transfer (*biweeklybudget.models.transaction.Transaction* attribute), 93

transfer\_id (*biweeklybudget.models.transaction.Transaction* attribute), 93

transferrable\_types (*biweeklybudget.models.account.AcctType* attribute), 82

transModal() (built-in function), 135

transModalAddSplitBudget() (built-in function), 136

transModalBudgetSplitRowHtml() (built-in function), 136

transModalDivFillAndShow() (built-in function), 136

transModalDivForm() (built-in function), 136

transModalFormSerialize() (built-in function), 136

transModalHandleSplit() (built-in function), 136

transModalOfxFillAndShow() (built-in function), 134

transModalSplitBudgetChanged() (built-in function), 136

transNoOfx() (built-in function), 134

transNoOfxDivForm() (built-in function), 134

TransUnreconciledAjax (class in *biweeklybudget.flaskapp.views.reconcile*), 70

txn\_id (*biweeklybudget.models.txn\_reconcile.TxnReconcile* attribute), 94

TxnReconcile (class in *biweeklybudget.models.txn\_reconcile*), 93

TxnReconcileAjax (class in *biweeklybudget.flaskapp.views.reconcile*), 70

txnReconcileModal() (built-in function), 134

txnReconcileModalDiv() (built-in function), 134

type (*biweeklybudget.models.ofx\_statement.OFXStatement* attribute), 86

## U

unit\_cost (*biweeklybudget.models.projects.BoMItem* attribute), 90

unreconciled (*biweeklybudget.models.account.Account* attribute), 81

unreconciled() (*biweeklybudget.models.ofx\_transaction.OFXTransaction* static method), 88

unreconciled() (*biweeklybudget.models.transaction.Transaction* static method), 93

unreconciled\_sum (*biweeklybudget.models.account.Account* attribute), 81

update() (*biweeklybudget.plaid\_updater.PlaidUpdater* method), 115

update\_is\_fields() (*biweeklybudget.models.ofx\_transaction.OFXTransaction* method), 88

update\_statement\_ofx() (*biweeklybudget.ofxapi.local.OfxApiLocal* method), 96

update\_statement\_ofx() (*biweeklybudget.ofxapi.remote.OfxApiRemote* method), 97

updateReconcileTrans() (built-in function), 134

upsert\_record() (in module *biweeklybudget.db*), 104

url (*biweeklybudget.models.projects.BoMItem* attribute), 90

## V

validate() (*biweeklybudget.flaskapp.views.accounts.AccountFormHandler* method), 55

validate() (*biweeklybudget.flaskapp.views.accounts.AccountTxfrFormHandler* method), 55

validate() (*biweeklybudget.flaskapp.views.budgets.BudgetFormHandler* method), 56

validate() (*biweeklybudget.flaskapp.views.budgets.BudgetTxfrFormHandler* method), 56

validate() (*biweeklybudget.flaskapp.views.credit\_payoffs.AccountOfxFormHandler* method), 57

validate() (*biweeklybudget.flaskapp.views.formhandlerview.FormHandlerView* method), 60

validate() (*biweeklybudget.flaskapp.views.fuel.FuelLogFormHandler* method), 61

validate() (*biweeklybudget.flaskapp.views.fuel.VehicleFormHandler* method), 61

method), 62

validate() (biweeklybudget.flaskapp.views.payperiods.SchedToTransFormHandler method), 65

validate() (biweeklybudget.flaskapp.views.payperiods.SkipSchedTransFormHandler method), 65

validate() (biweeklybudget.flaskapp.views.projects.BoMItemFormHandler method), 67

validate() (biweeklybudget.flaskapp.views.projects.ProjectsFormHandler method), 69

validate() (biweeklybudget.flaskapp.views.scheduled.SchedTransFormHandler method), 71

validate() (biweeklybudget.flaskapp.views.transactions.TransactionFormHandler method), 73

validate\_day\_of\_month() (biweeklybudget.models.scheduled\_transaction.ScheduledTransaction method), 92

validate\_gallons() (biweeklybudget.models.fuel.FuelFill method), 85

validate\_num\_per\_period() (biweeklybudget.models.scheduled\_transaction.ScheduledTransaction method), 92

validate\_odometer\_miles() (biweeklybudget.models.fuel.FuelFill method), 85

validateTransModalSplits() (built-in function), 136

value (biweeklybudget.models.dbsetting.DBSetting attribute), 84

Vault (class in biweeklybudget.vault), 120

VAULT\_ADDR (in module biweeklybudget.settings), 118

VAULT\_ADDR (in module biweeklybudget.settings\_example), 119

vault\_creds\_path (biweeklybudget.models.account.Account attribute), 81

vehicle (biweeklybudget.models.fuel.FuelFill attribute), 85

Vehicle (class in biweeklybudget.models.fuel), 85

vehicle\_id (biweeklybudget.models.fuel.FuelFill attribute), 85

VehicleAjax (class in biweeklybudget.flaskapp.views.fuel), 61

VehicleFormHandler (class in biweeklybudget.flaskapp.views.fuel), 61

vehicleModal() (built-in function), 130

vehicleModalDivFillAndShow() (built-in function), 130

vehicleModalDivForm() (built-in function), 130

## W

## X