## Contents

1. **Contributor Guide** ................................................................. 3
2. **Policy Parameter Naming and Placing Conventions** ...................... 6
3. **Tax-Calculator Public API** ..................................................... 8
4. **Micro Data** ........................................................................... 9
5. **License** ................................................................................ 10
6. **Contributors** ........................................................................ 10
Tax-Calculator simulates the USA federal individual income and payroll tax system. In conjunction with micro data that represent the USA population, Tax-Calculator can be used to estimate the aggregate revenue and distributional effects of tax reforms under static analysis assumptions. In conjunction with other modules, Tax-Calculator can be used to estimate reform effects under a range of non-static assumptions. Tax-Calculator is written in Python, an interpreted language that can execute on Windows, Mac, or Linux.

This documentation is about developing Tax-Calculator. If you just want to use Tax-Calculator to conduct tax analysis, read this user documentation.

If you are relying on Tax-Calculator capabilities in your own project, be sure to read the definition of the Tax-Calculator Public API.
1.1 Contributor Guide

The purpose of this guide is to get you to the point where you can make improvements to the Tax-Calculator and share them with the rest of the team.

We keep track of our Tax-Calculator code using Git, but we don’t expect you to be an expert Git user. Where possible, we link to Git and GitHub documentation to help with some of the unfamiliar terminology. Following the next steps will get you up and running and contributing to our model even if you’ve never used anything like Git.

If you have already completed the Setup Python and Setup Git sections, please skip to Workflow.

1.1.1 Setup Python

The Tax-Calculator is written in the Python programming language. Download and install the free Anaconda distribution of Python 3.6 from Anaconda. You must do this even if you already have Python installed on your computer because the Anaconda distribution contains all the additional Python packages that we use to conduct tax calculations (many of which are not included in other Python installations). You can install the Anaconda distribution without having administrative privileges on your computer and the Anaconda distribution will not interfere with any Python installation that came as part of your computer’s operating system.

1.1.2 Setup Git

1. Create a GitHub user account.
2. Install Git on your local machine by following steps 1-4 on Git setup.
3. Tell Git to remember your GitHub password by following steps 1-4 on password setup.
4. Sign in to GitHub and create your own remote repository (repo) of Tax-Calculator by clicking Fork in the upper right corner of the Tax-Calculator’s GitHub page. Select your username when asked “Where should we fork this repository?”
5. From your command line, navigate to the directory on your computer where you would like your local repo to live.

6. Create a local repo by entering at the command line the text after the $.

   $ git clone https://github.com/[github-username]/Tax-Calculator.git

7. From your command line or terminal, navigate to your local Tax-Calculator directory.

8. Make it easier to push your local work to others and pull others’ work to your local machine by entering at the command line:

   $ cd Tax-Calculator
   Tax-Calculator$ git remote add upstream https://github.com/open-source-economics/Tax-Calculator.git

9. Create a conda environment with all of the necessary packages to execute the source code:

   Tax-Calculator$ conda env create

10. The prior command will create a conda environment called “taxcalc-dev”. Activate this environment as follows:

    Tax-Calculator$ source activate taxcalc-dev

    If you are working on Windows, use the following from the command line:

    Tax-Calculator$ activate taxcalc-dev

    Note: never conda install the taxcalc package in the taxcalc-dev environment because the taxcalc source code and the installed package will conflict.

11. To check that everything is working properly, run the following at the command line from the Tax-Calculator directory:

    Tax-Calculator$ cd taxcalc
    Tax-Calculator/taxcalc$ py.test -m "not requires_pufcsv and not pre_release" -n4

    If you do have a copy of the puf.csv file used by Tax-Calculator, then on the second line above omit the ‘not requires_pufcsv and’ expression so as to execute ‘py.test -m “not pre_release” -n4’.

    If all the tests pass, you’re good to go. If they don’t pass, enter the following updates at the command line and then try running the tests again:

    Tax-Calculator$ conda update conda
    Tax-Calculator$ conda env update

    For more detail on Tax-Calculator testing procedures, read the testing documentation. If the tests still don’t pass, please contact us.

If you’ve made it this far, you’ve successfully made a remote copy (a fork) of central Tax-Calculator repo. That remote repo is hosted on GitHub.com. You’ve also created a local repo (a clone) that lives on your machine and only you can see; you will make your changes to the Tax-Calculator by editing the files in the Tax-Calculator directory on your machine and then submitting those changes to your local repo. As a new contributor, you will push your changes from your local repo to your remote repo when you’re ready to share that work with the team.

---

\(1\) The dollar sign is the end of the command prompt on a Mac. If you’re on Windows, this is usually the right angle bracket (>). No matter the symbol, you don’t need to type it (or anything to its left, which shows the current working directory) at the command line before you enter a command; the prompt symbol and preceding characters should already be there.
Don’t be alarmed if the above paragraph is confusing. The following section introduces some standard Git practices and guides you through the contribution process.

1.1.3 Workflow

The following text describes a typical workflow for changing Tax-Calculator. Different workflows may be necessary in some situations, in which case other contributors are here to help.

1. Before you edit the Tax-Calculator source code on your machine, make sure you have the latest version of the central Tax-Calculator repository by executing the following four Git commands:

   a. Tell Git to switch to the master branch in your local repo. Navigate to your local Tax-Calculator directory and enter the following text at the command line:

   ```
   Tax-Calculator$ git checkout master
   ```

   b. Download all of the content from the central Tax-Calculator repo:

   ```
   Tax-Calculator$ git fetch upstream
   ```

   c. Update your local master branch to contain the latest content of the central master branch using merge. This step ensures that you are working with the latest version of the Tax-Calculator:

   ```
   Tax-Calculator$ git merge upstream/master
   ```

   d. Push the updated master branch in your local repo to your GitHub repo:

   ```
   Tax-Calculator$ git push origin master
   ```

2. Create a new branch on your local machine. Think of your branches as a way to organize your projects. If you want to work on this documentation, for example, create a separate branch for that work. If you want to change the maximum child care tax credit in the Tax-Calculator, create a different branch for that project:

   ```
   Tax-Calculator$ git checkout -b [new-branch-name]
   ```

3. If your changes involve creating a new tax policy parameter, be sure to read about Tax-Calculator parameter naming conventions.

4. As you make changes, frequently check that your changes do not introduce bugs or degrade the accuracy of the Tax-Calculator. To do this, run the following command from the command line from inside the Tax-Calculator/taxcalc directory:

   ```
   Tax-Calculator/taxcalc$ py.test -m "not requires_pufcsv and not pre_release" -n4
   ```

   If the tests do not pass, try to fix the issue by using the information provided by the error message. If this isn’t possible or doesn’t work, we are here to help.

5. Now you’re ready to commit your changes to your local repo using the code below. The first line of code tells Git to track a file. Use “git status” to find all the files you’ve edited, and “git add” each of the files that you’d like Git to track. As a rule, do not add large files. If you’d like to add a file that is larger than 25 MB, please contact the other contributors and ask how to proceed. The second line of code commits your changes to your local repo and allows you to create a commit message; this should be a short description of your changes.

   ```
   Tip: Committing often is a good idea as Git keeps a record of your changes. This means that you can always revert to a previous version of your work if you need to. Do this to commit:
   ```
6. Periodically, make sure that the branch you created in step 2 is in sync with the changes other contributors are making to the central master branch by fetching upstream and merging upstream/master into your branch:

```
Tax-Calculator$ git fetch upstream
Tax-Calculator$ git merge upstream/master
```

You may need to resolve conflicts that arise when another contributor changed the same section of code that you are changing. Feel free to ask other contributors for guidance if this happens to you. If you do need to fix a merge conflict, re-run the test suite afterwards (step 4.)

7. When you are ready for other team members to review your code, make your final commit and push your local branch to your remote repo:

```
Tax-Calculator$ git push origin [new-branch-name]
```

8. From the GitHub.com user interface, open a pull request.

9. When you open a GitHub pull request, a code coverage report will be automatically generated. If your branch adds new code that is not tested, the code coverage percent will decline and the number of untested statements (“misses” in the report) will increase. If this happens, you need to add to your branch one or more tests of your newly added code. Add tests so that the number of untested statements is the same as it is on the master branch.

You should now read the more detailed pull-request workflow document.

### 1.1.4 Simple Usage

For examples of Tax-Calculator usage, see the Cookbook of Tested Recipes for Python Programming with Tax-Calculator, which contains Python scripts that can be executed by the Python command-line interpreter or imported into a Python notebook for interactive execution.

### 1.2 Policy Parameter Naming and Placing Conventions

Policy parameter names have two components; the first component is an abbreviation for the parameter’s tax category and the second component is an abbreviation for the role the parameter plays in those tax category rules. In some cases, parameter names have a subcategory that conveys additional information about the role the parameter plays (see Examples below).

#### 1.2.1 Tax Categories

Tax categories are uppercase:

- **ACTC**: Additional Child Tax Credit
- **ALD**: Above-the-Line Deduction
- **AMEDT**: Additional Medicare Tax
- **AMT**: Alternative Minimum Tax
- **CDCC**: Child and Dependent Care Credit
- **CG**: Capital Gain
CTC: Child Tax Credit
EITC: Earned Income Tax Credit
ETC: Education Tax Credit
FICA: Federal Income Contributions Act
ID: Itemized Deduction
II: Individual Income (including personal exemptions and tax brackets)
KT: Kiddie Tax
LLC: Lifetime Learning Credit
NIIT: Net Investment Income Tax
PT: Pass-Through Income
SS: Social Security
STD: Standard Deduction

### 1.2.2 Parameter Role

Abbreviates for the role the parameter plays in the tax rules are usually lowercase:

- c: ceiling (or use Max especially for integer variables)
- e: end
- ec: exclusion
- em: exemption
- f: floor (or use Min especially for integer variables)
- hc: haircut
- p: phaseout
- rt: rate (always expressed as a decimal, rather than a percentage, rate)
- s: start
- t: tax
- thd: threshold

Combine abbreviations to create more complex roles: frt = floor rate.

### 1.2.3 Examples

- `_AMT_em`: Alternative Minimum Tax exemption amount
- `_ID_ps`: Itemized Deduction phaseout Adjusted Gross Income start (Pease)
- `_AMT_brk1`: Alternative Minimum Tax first rate bracket top
- `_SS_Earnings_c`: Maximum taxable earnings for Social Security
- `_AMT_Child_em`: Child Alternative Minimum Tax exemption additional income base
- `_ETC_pe_Married`: Education Tax Credit phaseout ends (Married)
_EITC_MinEligAge_: Earned Income Tax Credit minimum eligibility age for those with no EITC-eligible children

### 1.2.4 Placing New Parameters in `current_law_policy.json`

All new policy parameters should be added to the `current_law_policy.json` file in a location that is near conceptually similar parameters. Be sure to specify the `section_1` and `section_2` values of each new parameter so that it appears in an appropriate place on the TaxBrain input page. If the new parameter is not supposed to appear on the TaxBrain input page, set the value of `section_1` and `section_2` to an empty string.

### 1.3 Tax-Calculator Public API

The Tax-Calculator's core capabilities are in the Python package called taxcalc, the source code for which is located in the Tax-Calculator/taxcalc directory tree.

Here we provide a high-level view of the public API of the taxcalc conda package with links to the source code. This high-level view is organized around the modules in the taxcalc package. Below is a list of the taxcalc package modules (in alphabetical order) with documentation about how to call each public class method and function. There is also a link to the source code for each documented member. However, it may be more convenient to access this list interactively via the alphabetical genindex of taxcalc members.

Developers who want to use Tax-Calculator capabilities in their own projects should restrict themselves to using this public API. All other Tax-Calculator members are private and subject to change without advance notice.
1.4 Micro Data

The Tax-Calculator simulates federal individual income and payroll taxes for each tax filing unit in a sample. For revenue estimation, that sample must be representative of the US population.

1.4.1 Description

This project commonly relies on a micro dataset that closely reproduces the multivariate distribution of income, deduction and credit items in 2009, extrapolated to 2015-2026 levels in accordance with CBO forecasts available in Spring 2016. It is intended to match similar but confidential data used by the Congressional Joint Committee on Taxation. The underlying dataset must be purchased from the Statistics of Income division of the Internal Revenue Service. Additional information on non-filers is taken from the March 2013 Current Population Survey.

The open source TaxData model implements the extrapolation and statistical match.

1.4.2 Documentation

For additional documentation on the core variables used by the Tax Calculator, refer to the general description booklet.
1.5 License

This project is in the public domain within the United States.

Additionally, we waive copyright and related rights in the work worldwide through the CC0 1.0 Universal public

domain dedication.

CC0 1.0 Universal Summary

This is a human-readable summary of the Legal Code (read the full text).

No Copyright

The person who associated a work with this deed has dedicated the work to the public domain by waiving all of his

or her rights to the work worldwide under copyright law, including all related and neighboring rights, to the extent

allowed by law.

You can copy, modify, distribute and perform the work, even for commercial purposes, all without asking permission.

Other Information

In no way are the patent or trademark rights of any person affected by CC0, nor are the rights that other persons may

have in the work or in how the work is used, such as publicity or privacy rights.

Unless expressly stated otherwise, the person who associated a work with this deed makes no warranties about the

work, and disclaims liability for all uses of the work, to the fullest extent permitted by applicable law. When using or

citing the work, you should not imply endorsement by the author or the affirmer.

1.6 Contributors

Many people have contributed to the Tax-Calculator, and we hope this list continues to grow.

1.6.1 Origins of Tax-Calculator

Tax-Calculator is a descendant of TAXCALC and TAXSIM, programs written by Dan Feenberg (NBER) and Inna

Shapiro (NBER).

1.6.2 Contributors to Tax-Calculator

Contributors include the following individuals who have made direct technical contributions to Tax-Calculator as well

as many others have provided valuable feedback and review.

- Dan Feenberg
- Sameer Sarkar
- Matt Jensen
- Ilia Kurenkov
- T.J. Alumbaugh
- Jake Lyons
- Amy Xu
- John O’Hare
- Regan Kuchan
• Maria Messick
• Martin Holmer
• Peter Steinberg
• Sean Wang
• Anderson Frailey
• Cody Kallen
• Xiyu Wang
• Hank Doupe
• Zach Risher
• Adele Hunter
• Salim Furth
• Richard Evans