Taxcount

From 40,000 ft

What is Taxcount?

- Open source software that helps bitcoin traders with US federal tax filing
- Imports wallet transaction histories and trade logs from exchanges

Taxcount as a function

- Input your wallet and trade histories for a tax year and some other info. Tags for categorizing transactions, a list of xPub keys, an optional date of residency in a US territory, and so on.

- Taxcount runs a parallel simulation of all transactions across wallets, exchanges, and blockchains. It carefully tracks the cost basis of assets to summarize capital gains and losses. Special attention is paid to fees.

- Outputs a "checkpoint file", a list of "detail CSVs", and a summary CSV.

Checkpoint file

A checkpoint can be thought of as a snapshot of the internal state of the simulation. It allows users to incrementally run their taxes by inputting all data for a calendar year.

Snapshots can be saved for each individual year. Each snapshot is a fully independent representation, they are not deltas.

A tool is included to bootstrap a snapshot for users who have incomplete transaction histories, or are unable to completely trace the origin of coins. Bootstrapping associates a transaction ID with a cost basis.

Detail CSV files

For each wallet history and each exchange ledger provided as input, a CSV will be produced that provides a full breakdown of how coins flow through the system, from acquisition, through the exchange, to releasing the asset to a third party. The files detail how coins are split and merged across multiple transactions.

Merges never end up as an average. When splits are merged, they simply show up as multiple rows in the CSV with the same identifying transaction ID. We call this process "atomization". Large coins can be split into smaller coins or passed through in full.

Useful for debugging and auditing purposes.

Summary CSV file

This is the most important output. Numbers in this CSV can be reported directly on IRS Form 8949, "Sales and other Dispositions of Capital Assets", for 1040 Schedule D, "Capital Gains and Losses".

It records short term gains and long term gains, both for US-sourced gains and territory-sourced gain with election of residency in a US territory.

Annoying details (Federal tax rules are insanely complex)

Spot margin trading and market-or-limit trading cannot be modeled the same way. Gains for market or limit trades is a function of proceeds minus cost basis. Gains for spot margin trades are simply the trade proceeds.

Spot margin fees can be applied as investment interest expenses, which are subject to additional restrictions.

There are two ways to handle fees: With VWAP accounting, the fee changes the cost basis. With atomization, the fee is split from the asset, reducing the size of the asset, which retains its cost basis. The fee is given the same cost basis from the split, and this new fee-basis goes into calculating the summary gains.

Fees with fee-basis cannot be modeled the same way as proceeds with cost-basis.

The simulation TL;DR:

- Implemented in `State::resolve()`.
- Each wallet history is input as a FIFO, each exchange ledger is another FIFO.
- The scheduler operates over 1-minute time-slices of each FIFO. Inputs must be chronologically ordered and normalized.
- Processing each FIFO item is basically just a large switch statement.
- Only moves monotonically forwards with linear time complexity. Never backwards and never needs to recursively search.

Transaction normalization

- Implemented separately for each wallet format as `FIFO::<_>::resolve()`.
- Uses the public blockchain API to "hydrate" each transaction with as much information as possible. Also used as source of truth for ordering and time.
- Combines the xPub inputs to directly identify the transaction as incoming, outgoing, a move between wallets, and other scenarios.
- De-duplicates transactions from multiple wallets by merging matching transaction IDs.

Support crates

- `Fett`: A simple concurrent memoizing HashMap.
    - Basically `Vec<RwLock<Vec<(K, V)>>>`
        - The outer Vec is the list of hash buckets. It is not resizeable. Its size defines the total concurrency available.
        - The inner Vecs are the Key-Value pairs. These can grow dynamically. They are intended to be kept small to reduce iteration overhead while seeking.
    - The map can be deconstructed and reconstructed to resize the hash buckets.
    - Deadlock-free by design.
    - Concurrency test suite with `loom`.

- `Esploda`: A sans-IO implementation of the Esplora protocol, and the Bitcoind JSON-RPC protocol.
    - Examples for `ureq` (sync HTTP client) and `reqwest` (async HTTP client with `tokio`).