ledger2beancount

Ledger to Beancount converter

Stefano Zacchiroli Martin Michlmayr

June 2019

Contents

Introduction

ledger2beancount is a script to automatically convert Ledger-based textual ledgers to Beancount ones.

Conversion is based on (concrete) syntax, so that information that is not meaningful for accounting reasons but still valuable (e.g., comments, formatting, etc.) can be preserved.

ledger2beancount aims to be compatible with the latest official release of beancount.

Installation

ledger2beancount is a Perl script and relies on the following Perl modules:

- Config::Onion
- Date::Calc
- DateTime::Format::Strptime
- File::BaseDir
- Getopt::Long::Descriptive
- String::Interpolate
- YAML::XS

You can install the required Perl modules with cpanminus:

```
cpanm --installdeps .
```

If you use Debian, you can install the dependencies with this command:

```
sudo apt install libconfig-onion-perl libdate-calc-perl \
    libfile-basedir-perl libyaml-libyaml-perl \
    libgetopt-long-descriptive-perl libdatetime-format-strptime-perl \
    libstring-interpolate-perl
```

Note that String::Interpolate (libstring-interpolate-perl) is not in Debian stable.

ledger2beancount itself consists of one script. You can clone the repository and run the script directly or copy it to `$HOME/bin` or a similar location:

```
git clone https://github.com/zacchiro/ledger2beancount/
./bin/ledger2beancount examples/simple.ledger
```

Arch Linux

ledger2beancount is available on AUR.

Debian

ledger2beancount is available in Debian.

macOS

You can install `cpanm` from Homebrew:

```
brew install cpanminus
```

Microsoft Windows

You can install Strawberry Perl on Windows and use `cpanm` as described above to install the required Perl modules. ledger2beancount is not packaged for Windows but you can clone this Git repository and run the script.

Configuration

ledger2beancount can use a configuration file. It will search for the config file `ledger2beancount.yml` in the current working directory. If that file is not found, it will look for `$HOME/.config/ledger2beancount/config.yml`. You can also pass an alternative config file via `--config/-c`. The file must end in `.yml` or `.yaml`. See the sample config file for the variables you can use.

While the configuration file is optional, you may have to define a number of variables for ledger2beancount to work correctly with your ledger files:

- `ledger_indent` sets the indentation level used in your ledger file (by default `4`).
- `date_format` has to be configured if you don't use the date format `YYYY-MM-DD`.
- `decimal_comma` has to be set to `true` if you use commas as the decimal separator (for example, `10,12 EUR` meaning 10 Euro and 12 cents).
- `commodity_map` defines mappings from ledger to beancount commodities. You have to set this if you use commodity codes like `€` or `£` (to map them to `EUR` and `GBP`, respectively).

Additionally, these options are useful to configure beancount:

- `operating_currencies`: a list of the currencies you frequently use.
- `beancount_header`: a file which is embedded at the beginning of the converted beancount file which can include beancount `option` statements, `plugin` directives, `query` information and more.

Other variables can be set to use various functionality offered by ledger2beancount. Please read the section on features to learn about these variables and refer to the complete list of configuration options at the end of the manual.

Usage

ledger2beancount accepts input from `stdin` or from a file and will write the converted data to `stdout`. You can run ledger2beancount like this on the example provided:

```
ledger2beancount examples/simple.ledger > simple.beancount
```

After you convert your ledger file, you should validate the generated beancount file with `bean-check` and fix all errors:

```
bean-check simple.beancount
```

You should also inspect the generated beancount file to see if it looks correct to you. Please note that ledger2beancount puts notes at the beginning of the generated beancount file if it encounters problems with the conversion.

If you believe that ledger2beancount could have produced a better conversion or if you get an error message from ledger2beancount, please file a bug along with a simple test case.

You can pipe the output of ledger2beancount to beancount's bean-format if you want to use the conversion as an opportunity to reformat your file.

Beancount compatibility

The syntax of beancount is quite stable but it's expected to become slightly less restrictive as some missing features are implemented (such as posting-level tags).

ledger2beancount aims to be compatible with the latest official release of beancount, but some functionality may require an unreleased version of beancount. You can install the latest development version of beancount directly from the beancount repository:

```
pip3 install hg+https://bitbucket.org/blais/beancount/
```

Currently, there are no features that require an unreleased version of beancount.

ledger2beancount is largely compatible with Beancount 2.0. If you use the following features, you need Beancount 2.1:

- UTF-8 letters and digits in account names
- Full-line comments in transactions
- Transaction tags on multiple lines

Features

ledger2beancount supports most of the syntax from ledger. It also offers some features to improve the conversion from ledger to beancount.

If you're new to beancount, we suggest you read this section in parallel to the illustrated ledger file provided. This example ledger file explains differences between ledger and beancount, shows how ledger syntax is converted to beancount and describes how you can use the features described in this section to improve the conversion from ledger to beancount. The illustrated example uses the same subsections as this section, so it's easy to follow in parallel.

You can convert the illustrated ledger file to beancount like this:

```
ledger2beancount --config examples/illustrated.yml examples/illustrated.ledger
```

But please be aware that it doesn't pass `bean-check`. See the comments in the file as to why.

Note on **regular expressions**: many of the features described below require you to specify regular expressions in ledger2beancount configuration file. The expected syntax (and semantics) for all such values is that of Perl regular expressions.

Accounts

ledger2beancount will convert ledger account declarations to beancount `open` statements using the `account_open_date` variable as the opening date. The `note` is used as the `description`.

Unlike ledger, beancount requires declarations for all account names. If an account was not declared in your ledger file but used, ledger2beancount will automatically create an `open` statement in beancount. You can turn this off by setting `automatic_declarations` to `false`. This is useful if you have include files and run ledger2beancount several times since duplicate `open` statements for the same account will result in an error from beancount.

ledger2beancount replaces ledger account names with valid beancount accounts and therefore performs the following transformations automatically:

1. Replaces space and other invalid characters with dash (`Liabilities:Credit Card` becomes `Liabilities:Credit-Card`)
2. Replaces account names starting with lower case letters with upper case letters (`Assets:test` becomes `Assets:Test`)
3. Ensures the first letter is a letter or number by replacing a non-letter first character with an `X`.

While these transformations lead to valid beancount account names, they might not be what you desire. Therefore, you can add account mappings to `account_map` to map the transformed account names to something different. The mapping will work on your ledger account names and on the account names after the transformation.

Unlike ledger, beancount expects all account names to start with one of five account types, also known as root names. The default root names are `Assets`, `Liabilities`, `Equity`, `Expenses`, and `Income`. If you want to use other root names, you can configure them using the beancount options `name_assets`, `name_liabilities`, `name_equity`, `name_expenses`, and `name_income`.

If you use more than five root names, you will have to rename them. ledger2beancount offers the `account_regex` option to mass rename account names. If you use the top-level

root name `Accrued` to track accounts payable and accounts receivable, you can rename them with this `account_regex` config option:

```
account_regex:
  ^Accrued:Accounts Payable:(.*): Liabilities:Accounts-Payable:$1
  ^Accrued:Accounts Receivable:(.*): Assets:Accounts-Receivable:$1
```

Ledger's `apply account` and `alias` directives are supported. The mapping of account names described above is done after these directives.

Amounts

In ledger, amounts can be placed after the amount. This is converted to beancount with the the amount first, followed by the commodity.

If you use commas as the decimal separator (i.e. values like `10,12`, using the ledger option `--decimal-comma`) you have to set the `decimal_comma` option to `true`. Please note that commas are not supported as the decimal separator in beancount at the moment (issue 204) so your amounts are converted not to use comma as the decimal separator.

Commas as separators for thousands (e.g. `1,000,000`) are supported by beancount.

Commodities

Like accounts, ledger2beancount will convert ledger commodity declarations to beancount. The `note` is converted to `name`. As with account names, ledger2beancount will create `commodity` statements for all commodities used in your ledger file (if `automatic_declarations` is `true`).

ledger2beancount will automatically convert commodities to valid beancount commodities. This involves replacing all invalid characters with a dash (a character allowed in beancount commodities but not in ledger commodities), stripping quoted commodities, making the commodity uppercase and limiting it to 24 characters. Furthermore, the first character will be replaced with an `X` if it's not a letter and the same will be done for the last character if it's not a letter or digit. Finally, all beancount commodities currently have to consist of at least two characters (issue 192).

If you require a mapping between ledger and beancount commodities, you can use `commodity_map`. You can use your ledger commodity names or the names after the transformation in the map to perform a mapping to another commodity name.

Commodity symbols (like `$`, `€` and `£`) are supported and converted to their respective commodity codes (like `USD`, `EUR`, `GBP`). Update `commodity_map` if you use other symbols.

Flags

ledger2beancount supports both transaction flags (transaction state) and account flags (state flags).

Dates

ledger supports a wide range of date formats whereas beancount requires all dates in the format `YYYY-MM-DD` (ISO 8601). The variable `date_format` has to be set if you don't use ISO 8601 for the dates in your ledger file. `date_format` uses the same format as the ledger options `--input-date-format` and `--date-format` (see `man 1 date`).

Ledger allows dates without a year if the year is declared using the `Y`, `year` and `apply year` directives. If `date_format_no_year` is set, ledger2beancount can convert such dates to `YYYY-MM-DD`.

Posting-level dates are recognized by ledger2beancount and stored as metadata according to the `postdate_tag` (`date` by default) but this has no effect in beancount. There is a proposal to support this functionality in a different way, but this is not implemented in beancount yet.

While ledger2beancount itself doesn't read your ledger config file, the script `ledger2beancount-ledger-config` can be used to parse your ledger config file (`~/.ledgerrc`) or your ledger file (ledger files may contain ledger options) to output the correct config option for ledger2beancount.

Auxiliary dates

Beancount currently doesn't support ledger's auxiliary dates (or effective dates; also known as date2 in hledger) (but there is a proposal to support this functionality in a different way), so these are stored as metadata according to the `auxdate_tag` variable. Unset the variable if you don't want auxiliary dates to be stored as metadata. Account and posting-level auxiliary dates are supported.

Transaction codes

Beancount doesn't support ledger's transaction codes. These are therefore stored as metatags if `code_tag` is set.

Narration

The ledger payee information, which is generally used as free-form text to describe the transaction, is stored in beancount's narration field and properly quoted.

Payees

Ledger has limited support for payees. A `payee` metadata key can be set but this also overrides the free-form text to describe the transaction. Payees can also be declared explicitly in ledger but this is not required by beancount, so such declarations are ignored (they are preserved as comments).

hledger allows the separation of payee and narration using the pipe character (`payee | narration`). This is supported by ledger2beancount if the `hledger` option is enabled.

Since ledger has limited support for payees, ledger2beancount offers several features to determine the payee from the transaction itself.

You can set `payee_split` and define a list of regular expressions which allow you to split ledger's payee field into payee and narration. You have to use regular expressions with the named capture groups `payee` and `narration`. For example, given the ledger transaction header

```
2018-03-18 * Supermarket (Tesco)
```

and the configuration

```
payee_split:
  - (?<narration>.*?)\s+\((?<payee>Tesco)\)
```

ledger2beancount will create this beancount transaction header:

```
2018-03-18 * "Tesco" "Supermarket"
```

In other words, `payee_split` allows you to split the ledger payee into payee and narration in beancount. `payee_split` is a list of regular expressions and ledger2beancount stops when a match is found.

Furthermore, you can use `payee_match` to match based on the ledger payee field and assign payees according to the match. This variable is a list consisting of regular expressions and the corresponding payees. For example, if your ledger contains a transaction like:

```
2018-03-18 * Oyster card top-up
```

you can use

```
payee_match:
  - ^Oyster card top-up: Transport for London
```

to match the line and assign the payee `Transport for London`:

```
2018-03-18 * "Transport for London" "Oyster card top-up"
```

Unlike `payee_split`, the full payee field from ledger is used as the narration in beancount. Again, ledger2beancount stops after the first match. Beancount comes with a plugin called `fix_payees` which offers a similar functionality to `payee_match`: it renames payees based on a set of rules which allow you to match account names, payees and the narration. The difference is that ledger2beancount's `payee_match` will write the matched payee to the beancount file whereas the `fix_payees` plugin leaves your input file intact and assigns the new payee within beancount.

Please note that the `payee_match` is done after `payee_split` and `payee_match` is evaluated even if `payee_split` matched. This allows you to remove some information from the narration using `payee_split` while overriding the found payee using `payee_match`.

The regular expressions from `payee_split` and `payee_match` are evaluated in a case sensitive manner by default. If you want case insensitive matches, you can prefix your pattern with `(?i)`, for example:

```
payee_match:
  - (?i)^Oyster card top-up: Transport for London
```

Finally, metadata describing a payee or payer will be used to set the payee. The tags used for that information can be specified in `payee_tag` and `payer_tag`. Payees identified with these tags will override the payees found with `payee_split` and `payee_match` (although in the case of `payee_split` the narration will be modified as per the regular expression). This allows you to define generic matches using `payee_split` and `payee_match` and override special cases using metadata information.

Metadata

Account and posting metadata are converted to beancount syntax. Metadata keys used in ledger can be converted to different keys in beancount using `metadata_map`. Metadata can also be converted to links (see below).

Beancount is more restrictive than ledger in what it allows as metadata keys. ledger2beancount will automatically convert metadata keys to valid beancount metadata keys. This involves replacing all invalid characters with a dash and making sure

the first character is a lowercase letter (either by lowercasing a letter or adding the prefix `x`).

ledger2beancount also supports typed metadata (i.e. `key::` instead of `key:`) and doesn't quote the values accordingly, but you should make sure the values are valid in beancount.

Tags

Beancount allows tags for transactions but currently doesn't support tags for postings (issue 144). Because of this, posting-level tags are currently stored as metadata with the key `tags`. This should be seen as a workaround because metadata with the key `tags` is not treated the same way by beancount as proper tags.

Ledger's `apply tag` directive is supported. If the string to apply is metadata or a link (according to `link_match`, see below), the information will be added to each transaction between `apply tag` and `end tag`. If it's a tag, beancount's equivalent of `apply tag` is used (`pushtag` and `poptag`).

Note that tags can be defined in ledger using a `tag` directive. This is not required in beancount and there's no equivalent directive so all `tag` directives are skipped.

Links

Beancount differentiates between tags and links whereas ledger doesn't. Links can be used in beancount to link several transactions together. ledger2beancount offers two mechanisms to convert ledger tags and metadata to links.

First, you can define a list of metadata tags in `link_tags` whose values should be converted to beancount links instead of metadata. For example:

```
link_tags:
  - Invoice
```

with the ledger input

```
2018-03-19 * Invoice 4
    ; Invoice:: 4
```

will be converted to

```
2018-03-19 * Invoice 4 ^4
```

instead of

```
2018-03-19 * Invoice 4 #4
```

Tags are case insensitive. Be aware that the metadata must not contain any whitespace.

Since posting-level links are currently not allowed in beancount, they are stored as metadata.

Second, you can define regular expressions in `link_match` to determine that a tag should be rendered as a link instead. For example, if you tag your trips in the format `YYYY-MM-DD-foo`, you could use

```
link_match:
  - ^\d\d\d\d-\d\d-\d\d-
```

to render them as links. So the ledger transaction header

```
2018-02-02 * Train Brussels airport to city
    ; :2018-02-02-brussels-fosdem:debian:
```

would become the following in beancount:

```
2018-02-02 * "Train Brussels airport to city" ^2018-02-02-brussels-fosdem #debian
```

Comments

Comments are supported.

Currently, beancount doesn't accept top-level comments with the `|` marker (issue 282). ledger2beancount changes such comments to use the `;` marker.

Virtual costs

Beancount does not have a concept of virtual costs (issue 248). ledger2beancount therefore treats them as regular costs (or, rather, as regular prices).

Lots

Lot costs and prices are supported, including per-unit and total lot costs. Lot dates and lot notes are converted to beancount.

The behaviour of ledger and beancount is different when it comes to costs. In ledger, the statement

```
Assets:Test           10.00 EUR @ 0.90 GBP
```

creates the lot `10.00 EUR {0.90 GBP}`. In beancount, this is not the case and a cost is only associated if done so explicitly:

```
Assets:Test           10.00 EUR {0.90 GBP}
```

This makes automatic conversion tricky because some statements should be simple conversions without associating a cost whereas it's vital to preserve the cost in other conversions.

Generally, it doesn't make sense to preserve the cost for currency conversion (as opposed to conversions involving commodities like shares and stocks). Since most currency codes consist of 3 characters (`EUR`, `GBP`, `USD`, etc), the script makes a simple conversion (`10.00 EUR @ 0.90 GBP`) if both commodities consist of 3 characters. Otherwise it associates a cost (`1 LU0274208692 {48.67 EUR}`). Since some 3 character symbols might be commodities instead of currencies (e.g. `ETH` and `BTH`), the `currency_is_commodity` variable can be used to treat them as commodities and associate a cost in conversions. Similarly, `commodity_is_currency` can be used to configure commodities that should be treated as currencies in the sense that no cost is retained. This is useful if you, for example, track miles or hotel points that are sometimes redeemed for a cash value. Both of these variables expect beancount commodities, i.e. after transformation and mapping. (Note that beancount itself uses the terms "commodity" and "currency" interchangeably.)

Balance assertions and assignments

Ledger balance assertions are converted to beancount `balance` statements.

Please note that beancount evaluates balance assertions at the beginning of the day whereas ledger evaluates them at the end of the day (up to ledger 3.1.1) or at the end of the transaction (newer versions of ledger). Therefore, we schedule the balance assertion for the day *after* the original transaction. This assumes that there are no other transactions on the same day that change the balance again for this account.

In addition to balance assertions, ledger also supports balance assignments. ledger2beancount can handle some, but not all types of balance assertions. The most simple case is something like:

```
2012-03-10 KFC
    Expenses:Food                $20.00
    Assets:Cash                         = $50.00
```

which can be handled like a balance assertion. However, ledger also allows transactions with two null postings when there's a balance assignment, as in:

```
2012-03-10 KFC
    Expenses:Food                $20.00
    Expenses:Drink
    Assets:Cash                         = $50.00
```

This can't be handled by ledger2beancount. While ledger can calculate how much you spent in `Assets:Cash` and balance it with `Expenses:Drink`, ledger2beancount can't. The transformation of this transaction will lead to two null postings, which `bean-check` will flag as invalid.

Finally, ledger allows transactions solely consisting of two null postings when one has a balance assignment:

```
2012-03-10 Adjustment
    Assets:Cash                         = $500.00
    Equity:Adjustments
```

ledger2beancount will create a beancount `pad` statement, followed by a `balance` statement the following day, to set the correct balance.

Automated transactions

Ledger's automated transactions are not supported in beancount. They are added as comments to the beancount file.

Periodic transactions

Ledger's periodic transactions are not supported in beancount. They are added as comments to the beancount file.

Virtual postings

Ledger's concept of virtual postings does not exist in beancount. Ledger has two types of virtual postings: those in parentheses (`(Budget:Food)`) which don't have to balance and those in brackets (`[Budget:Food]`) which have to balance. The former violate the accounting equation and can't be converted to beancount. The latter can be converted by making them into "real" accounts. ledger2beancount will do this if the `convert_virtual` option is set to `true`. By default, ledger2beancount will simply skip all virtual postings.

If you set `convert_virtual` to `true`, be aware that all account names have to start with one of five assets classes (`Assets`, etc). This is often not the case for virtual postings, so you will have to rename or map these account names.

Inline math

Very simple inline math is supported in postings. Specifically, basic multiplications and divisions are supported, such as shown in the following transactions:

```
2018-03-26 * Simple inline math
    Assets:Test1            1 GBP @ (1/1.14 EUR)
    Assets:Test2                       -0.88 EUR

2018-03-26 * Simple inline math
    Assets:Test1                    (1 * 3 GBP)
    Assets:Test2                         -3 GBP
```

Support for more complex inline math would require substantial changes to the parser.

Implicit conversions

ledger allows implicit conversions under some circumstances, such as in this example:

```
2019-01-29 * Implicit conversion
    Assets:A                 10.00 EUR
    Assets:B                -11.42 USD
```

They are generally a bad idea since they make it very easy to hide problems that are hard to track down. beancount doesn't support implicit conversions.

ledger2beancount supports implicit conversions if there are only two postings in a transaction (the most common case). More complex implicit conversations are not supported.

Fixated prices and costs

ledger allows you to "fix" the cost or price at the time of a transaction, which means the amount will not be revalued subsequently when the price of the commodity changes in the `pricedb`. beancount doesn't have a notion of a fixated price or cost.

However, you can achieve the same result in beancount. ledger2beancount will always convert ledger fixated prices and costs to costs in beancount. This way, the original cost is always attached to the transaction. You can then use `SUM(COST(position))` to get the original value.

hledger syntax

The syntax of hledger is largely compatible with that of ledger. If the `hledger` config option is set to `true`, ledger2beancount will look for some hledger specific features:

1) hledger allows the separation of a transaction's description into payee and note (narration) using the pipe character (`payee | narration`).
2) hledger allows `date:` and `date2:` to specify posting dates in posting comments in addition to ledger's `[date=date2]` syntax.
3) The syntax of tags is different in hledger: `tag1: tag2:, tag3:` in hledger vs `:tag1:tag2:tag3:` in ledger.

Ignoring certain lines

Sometimes it makes sense to exclude certain lines from the conversion. For example, you may not want a specific `include` directive to be added to the beancount file if the file contains ledger-specific definitions or directives with no equivalence in beancount.

ledger2beancount allows you to define a marker in the config file as `ignore_marker`. If this marker is found as a ledger comment on a line, the line will be skipped and not added to the beancount output. For example, given the config setting

```
ignore_marker: NoL2B
```

you could do this:

```
C 1.00 Mb = 1024 Kb ; NoL2B
```

If you want to skip several lines, you can use `$ignore_marker begin` and `$ignore_marker end`. This syntax is also useful for ledger `include` directives, which don't allow a comment on the same line.

```
; NoL2B begin
include ledger-specific-header.ledger
; NoL2B end
```

Since some people use ledger and beancount in parallel using ledger2beancount, it is sometimes useful to put beancount-specific commands in the input file. Of course, they may not be valid in ledger. Therefore, you can put a commented out line in the ledger input, mark it with the `$keep_marker` and ledger2beancount will uncomment the line and put it in the output.

Given the input

```
; 2013-11-03 note Liabilities:CreditCard "Called about fraud" ; L2Bonly
```

ledger2beancount will add the following line to the beancount output:

```
2013-11-03 note Liabilities:CreditCard "Called about fraud"
```

You can also use `$keep_marker begin` and `$keep_marker end` to denote multiple lines that should be included in the output:

```
; L2Bonly begin
; 2014-07-09 event "location" "Paris, France"
; 2018-09-01 event "location" "Bologna, Italy"
; L2Bonly end
```

Unsupported features

Unsupported in beancount

The following features are not supported in beancount and therefore commented out during the conversion from ledger to beancount:

- Automated transactions
- Commodity conversion (`C AMOUNT1 = AMOUNT2`)
- Commodity format (`D AMOUNT`)
- Commodity pricing: ignore pricing (`N SYMBOL`)
- Timeclock support (`I`, `i`, `O`, `o`, `b`, `h`)
- Periodic transactions

Unsupported in ledger2beancount

The following ledger features are currently not supported by ledger2beancount:

- The `define` directive

Contributions are welcome!

Configuration options

Input options

The following options may be needed for ledger2beancount to interpret your ledger files correctly.

date_format The date format used in your ledger file (default: `%Y-%m-%d`).

date_format_no_year The date format for dates without the year when ledger's `Y/year` directive is used (default: `%m-%d`).

ledger_indent Sets the indentation level used in your ledger file (default: `4`).

decimal_comma Parses amounts with the decimal comma (e.g. `10,00 EUR`). Set this option to `true` if you use option `--decimal-comma` in ledger.

hledger Tells ledger2beancount whether to attempt to parse hledger-specific features.

Other options

beancount_indent Sets the indentation level for the generated beancount file (default: `2`).

operating_currencies A list of frequently used currencies. This is used by fava, the web UI for beancount.

automatic_declarations Emit account and commodity declarations. (Default: `true`)

Note: the declarations done in ledger via `account` and `commodity` declarations are always converted. If this option is `true`, declarations are created for those which have not been explicitly declared in ledger but used.

account_open_date The date used to open accounts (default: `1970-01-01`).

commodities_date The date used to create commodities (default: `1970-01-01`).

beancount_header Specifies a file which serves as a beancount "header", i.e. it's put at the beginning of the converted beancount file. You can use such a header to specify options for beancount, such as `option "title"`, define `plugin` directives or beancount `query` information.

ignore_marker Specifies a marker that tells ledger2beancount to ignore a line if the marker is found.

keep_marker Specifies a marker that tells ledger2beancount to take a line from the input that is commented out, uncomment it and display it in the output.

convert_virtual Specifies whether virtual postings should be converted. If set to `true`, virtual postings in brackets will be made into real accounts. (Virtual postings in parentheses are always ignored, regardless of this option.)

account_map Specifies a hash of account names to be mapped to other account names.

account_regex Specifies a hash of regular expressions to replace account names.

commodity_map Specifies a mapping of ledger commodities to beancount commodities.

metadata_map Specifies a mapping of ledger metadata keys to corresponding beancount keys.

payee_tag Specify a metadata tag (after the mapping done by `metadata_map`) used to set the payee.

payer_tag Specify a metadata tag (after the mapping done by `metadata_map`) used to set the payee.

payee_split Specifies a list of regular expressions to split ledger's payee field into payee and narration. You have to use the named capture groups `payee` and `narration`.

payee_match Specifies a list of regular expressions and corresponding payees. The whole ledger payee becomes the narration and the matched payee from the regular expression becomes the payee.

postdate_tag Specifies the metadata tag to be used to store posting dates. (Use the empty string if you don't want the metadata to be added to beancount.)

auxdate_tag Specifies the metadata tag to be used to store auxiliary dates (also known as effective dates; or `date2` in hledger). (Use the empty string if you don't want the metadata to be added to beancount.)

code_tag Specifies the metadata tag to be used to store transaction codes. (Use the empty string if you don't want the metadata to be added to beancount.)

link_match Specifies a list of regular expressions that will cause a tag to be rendered as a link.

link_tags Specifies a list of metadata tags whose values should be converted to beancount links instead of metadata. Tags are case insensitive and values must not contain whitespace.

currency_is_commodity Specifies a list of commodities that should be treated as commodities rather than currencies even though they consist of 3 characters (which is usually a characteristic of a currency). Expects beancount commodities (i.e. after transformation and mapping).

commodity_is_currency Specifies a list of commodities that should be treated as currencies (in the sense that cost is not retained). Expects beancount commodities (i.e. after transformation and mapping).

Bugs and contributions

If you find any bugs in ledger2beancount or believe the conversion from ledger to beancount could be improved, please open an issue. Please include a small test case so we can reproduce the problem.

See the contributing guide for more information on how to contribute to ledger2beancount.