tariochbctools Documentation

Release 0.34.0.post1.dev2

Patrick Ruckstuhl

Apr 29, 2024

CONTENTS

Some importers, plugins and price fetchers for the double-entry bookkeeping software Beancount.

CHAPTER

ONE

CONTENTS

1.1 Installation

As this is released on PyPI you can simply install it with

```
pip install tariochbctools
```

1.2 Importers

The importers normally all work very well together with Smart Importer and are also usable in Fava.

1.2.1 Bitstamp

Import transactions from Bitstamp

Create a file called (or ending with) bitstamp.yaml in your import location (e.g. downloads folder).

```
username: "12345"
key: "MyKey"
secret: "MySecret"
account: 'Assets:Bitstamp'
otherExpensesAccount: 'Expenses:Fee'
capGainAccount: 'Income:Capitalgain'
monthCutoff: 3
currencies:
  - eur
  - btc
```

```
from tariochbctools.importers.bitst import importer as bitstimp

CONFIG = [bitstimp.Importer()]
```

1.2.2 QuickFile

Import from QuickFile using their API services. Supports a range of (mostly UK) banks.

Requires a QuickFile account (any pricing plan, including free) but with a paid Automated Bank Feed subscription (small annual fee).

It is assumed you already have automated bank feeds configured within QuickFile for the accounts of interest and are able to browse transactions within the QuickFile dashboard.

```
from tariochbctools.importers.quickfile import importer as qfimp

CONFIG = [qfimp.Importer()]
```

Create a file called `quickfile.yaml` in your import location (e.g. download folder).

```
account_number: "YOUR_ACCOUNT_NUMBER"
api_key: YOUR_API_KEY
app_id: YOUR_APP_ID
from_date: 2020-12-13
to_date: 2020-12-20
accounts:
    1200: Assets:Other
    1201: Assets:Savings
transaction_count: 200
```

from_date and to_date are both optional

To obtain an API key you must create an app in the *Account Settings* | *3rd Party Integration* | *API* section of your account dashboard.

The only permissions it needs to have is "Invoices.Bank_Search"

your api_key is for your account, you can find it on "Settings - My Apps" or in the quickfile sandbox

Accounts are indexed in the config by their `nominal code` (typically: ~1200) visible in each account's settings. Only accounts listed in the config will be queried.

1.2.3 Revolut

Import CSV from Revolut

```
from tariochbctools.importers.revolut import importer as revolutimp

CONFIG = [revolutimp.Importer("/Revolut-CHF.*\.csv", "Assets:Revolut:CHF", "CHF")]
```

1.2.4 Wise (formerly Transferwise)

Import from Wise using their api.

First, generate a personal API token by logging on and going to settings. Next, you need to generate a public/private key pair and then upload the public key part to your account. To generate the keys, execute (e.g. in your `.ssh` folder)

```
openssl genrsa -out wise.pem
openssl rsa -pubout -in wise.pem -out wise_public.pem
openssl pkey -in wise.pem -traditional > wise_traditional.pem
```

The final command makes a traditional private key for compatibility with the python rsa library. This may stop being necessary at some point. See *this page https://github.com/sybrenstuvel/python-rsa/issues/80* for details.

Now upload the *public* key part to your Wise account.

You can then create an import config for beancount, or add Wise to your existing one.

```
from tariochbctools.importers.transferwise import importer as twimp

CONFIG = [twimp.Importer()]
```

Create a file called (or ending with) transferwise.yaml in your import location (e.g. download folder).

```
token: <your api token>
baseAccount: <Assets:Transferwise:>
privateKeyPath: /path/to/wise_traditional.pem
```

Optionally, you can provide a dictionary of account names mapped by currency. In this case you must provide a name for every currency in your Wise account, otherwise the import will fail.

```
token: <your api token>
baseAccount:
  SEK: "Assets:MySwedishWiseAccount"
  GBP: "Assets:MyUKWiseAccount"
privateKeyPath: /path/to/wise_traditional.pem
```

1.2.5 TrueLayer

Import from TrueLayer using their api services. e.g. supports Revolut. You need to create a dev account and see their documentation about how to get a refresh token.

```
from tariochbctools.importers.truelayer import importer as tlimp

CONFIG = [tlimp.Importer()]
```

Create a file called (or ending with) truelayer.yaml in your import location (e.g. download folder).

```
account: <Assets:MyBank>
client_id: <CLIENT ID>
client_secret: <CLIENT SECRET>
refresh_token: <REFRESH TOKEN>
```

Instead of a single `account`, the configuration may include a *mapping* from TrueLayer account IDs to beancount accounts. e.g.:

```
accounts:
  1aacb3110398ec5a2334fb0ffc2fface: Assets:Revolut:GBP
  ec34db160c61d468dc1cedde8bedb1f1: Liabilities:Visa
```

If it is present, transactions for *only these accounts* will be imported.

1.2.6 Nordigen

Import from Nordigen using their api services. e.g. supports Revolut. You need to create a free account and create a token. I've included a small cli to allow to hook up to different banks with nordigen. If you're country is not supported you can play around with other countries e.g. CH is not allowed but things like revolut still work. You can also create multiple links and they will all be listed in the end.

```
nordigen-conf list_banks --secret_id YOURSECRET_ID --secret_key YOURSECRET_KEY --country␣
↪DE
nordigen-conf create_link --secret_id YOURSECRET_ID --secret_key YOURSECRET_KEY --bank␣
↪REVOLUT_REVOGB21 --reference myref
nordigen-conf list_accounts --secret_id YOURSECRET_ID --secret_key YOURSECRET_KEY
nordigen-conf delete_link --secret_id YOURSECRET_ID --secret_key YOURSECRET_KEY --
↪reference myref
```

```
from tariochbctools.importers.nordigen import importer as nordimp

CONFIG = [nordimp.Importer()]
```

Create a file called (or ending with) nordigen.yaml in your import location (e.g. download folder).

```
secret_id: <YOURSECRET_ID>
secret_key: <YOURSECRET_KEY>

accounts:
  - id: <ACCOUNT-ID>
    asset_account: "Assets:MyAccount:CHF"
```

1.2.7 ZKB

Import mt940 from Zürcher Kantonalbank

```
from tariochbctools.importers.zkb import importer as zkbimp

CONFIG = [zkbimp.ZkbImporter("/\d+\.mt940", "Assets:ZKB")]
```

1.2.8 Raiffeisen CH

Import mt940 from Raiffeisen Schweiz

```
from tariochbctools.importers.raiffeisench import importer as raiffeisenimp

CONFIG = [
    raiffeisenimp.RaiffeisenCHImporter("/Konto_CH\d+_\d+\.mt940", "Assets:Raiffeisen")
]
```

1.2.9 Interactivebrokers

Import dividends and buys from Interactive Brokers

Create a file called (or ending with) ibkr.yaml in your import location (e.g. downloads folder).

```
token: <flex web query token>
queryId: <flex query id>
baseCcy: CHF
```

```
from tariochbctools.importers.ibkr import importer as ibkrimp

CONFIG = [ibkrimp.Importer()]
```

1.2.10 ZAK

Import PDF from Bank Cler ZAK

```
from tariochbctools.importers.zak import importer as zakimp

CONFIG = [zakimp.Importer(r"Kontoauszug.*\.pdf", "Assets:ZAK:CHF")]
```

1.2.11 mt940

Import Swift mt940 files.

1.2.12 Schedule

Generate scheduled transactions.

Define a file called (or ending with) schedule.yaml in your import location (e.g. downloads folder). That describes the schedule transactions. They will be added each month at the end of the month.

```
transactions:
  - narration: 'Save'
    postings:
        - account: 'Assets:Normal'
          amount: '-10'
          currency: CHF
        - account: 'Assets:Saving'
```

```
from tariochbctools.importers.schedule import importer as scheduleimp

CONFIG = [scheduleimp.Importer()]
```

1.2.13 Cembra Mastercard Montly Statement

Import Monthly Statement PDF from Cembra Money Bank (e.g. Cumulus Mastercard). Requires the dependencies for camelot to be installed. See https://camelot-py.readthedocs.io/en/master/user/install-deps.html#install-deps

```
from tariochbctools.importers.cembrastatement import importer as cembrastatementimp

CONFIG = [cembrastatementimp.Importer("\d+.pdf", "Liabilities:Cembra:Mastercard")]
```

1.2.14 Blockchain

Import transactions from Blockchain

Create a file called (or ending with) blockchain.yaml in your import location (e.g. downloads folder).

```
base_ccy: CHF
addresses:
  - address: 'SOMEADDRESS'
    currency: 'BTC'
    narration: 'Some Narration'
    asset_account: 'Assets:MyCrypto:BTC'
  - address: 'SOMEOTHERADDRESS'
    currency: 'LTC'
    narration: 'Some Narration'
    asset_account: 'Assets:MyCrypto:LTC'
```

```
from tariochbctools.importers.blockchain import importer as bcimp

CONFIG = [bcimp.Importer()]
```

1.2.15 Mail Adapter

Instead of expecting files to be in a local directory. Connect per imap to a mail account and search for attachments to import using other importers.

Create a file called mail.yaml in your import location (e.g. downloads folder).

```
host: "imap.example.tld"
user: "myuser"
password: "mypassword"
folder: "INBOX"
targetFolder: "Archive"
```

The targetFolder is optional, if present, mails that had attachments which were valid, will be moved to this folder.

```
from tariochbctools.importers.general.mailAdapterImporter import MailAdapterImporter

CONFIG = [MailAdapterImporter([MyImporter1(), MyImporter2()])]
```

1.2.16 Neon

Import CSV from Neon

```
from tariochbctools.importers.neon import importer as neonimp

CONFIG = [neonimp.Importer("\d\d\d\d_account_statements\.csv", "Assets:Neon:CHF")]
```

1.2.17 Viseca One

Import PDF from Viseca One

```
from tariochbctools.importers.viseca import importer as visecaimp

CONFIG = [visecaimp.Importer(r"Kontoauszug.*\.pdf", "Assets:Viseca:CHF")]
```

1.2.18 BCGE

Import mt940 from BCGE

```
from tariochbctools.importers.bcge import importer as bcge

CONFIG = [bcge.BCGEImporter("/\d+\.mt940", "Assets:BCGE")]
```

1.2.19 Swisscard cards

Import Swisscard's *Cashback Cards <https://www.cashback-cards.ch/>* transactions from a CSV export.__

```
from tariochbctools.importers.swisscard import importer as swisscard

CONFIG = [swisscard.SwisscardImporter("swisscard/.*\.csv", "Liabilities:Cashback")]
```

1.3 Plugins

1.3.1 generate_base_ccy_prices

Dynamically generates prices to the base ccy by applying the fx rate to the base ccy for non base ccy prices

```
plugin "tariochbctools.plugins.generate_base_ccy_prices" "CHF"
```

1.3.2 check_portfolio_sum

For ledger files that contain multiple "portfolios", the plugin verifies that on each transaction, all the "portfolios" have the same weight. Portfolio is the second part of the account name. e.g. Asset:**Peter**:Bank1

```
plugin "tariochbctools.plugins.check_portfolio_sum"
```

1.4 Price fetchers

See the official Beanprice for a lot of price fetchers. Also most of the price fetchers which used to be in this repository have been migrated there.

1.4.1 Interactivebrokers

Fetches prices from Interactivebrokers Only works if you have open positions with the symbols. Requires the environment variables `IBKR_TOKEN` to be set with your flex query token and `IBKR_QUERY_ID` with a flex query that contains the open positions.

```
2019-01-01 commodity VWRL
  price: "CHF:tariochbctools.plugins.prices.ibkr/VWRL"
```

1.5 License

MIT License

1.6 tariochbctools

1.6.1 tariochbctools package

Subpackages

tariochbctools.importers package

Subpackages

tariochbctools.importers.bcge package

Submodules

tariochbctools.importers.bcge.importer module

class tariochbctools.importers.bcge.importer.**BCGEImporter**(*regexps*, *account*)

Bases: *Importer*

prepare_narration(*trxdata*)

prepare_payee(*trxdata*)

tariochbctools.importers.bcge.importer.**strip_newline**(*string*)

Module contents

tariochbctools.importers.bitst package

Submodules

tariochbctools.importers.bitst.importer module

class tariochbctools.importers.bitst.importer.**Importer**

Bases: ImporterProtocol

An importer for Bitstamp.

`extract`(*file*, *existing_entries*)

Extract transactions from a file.

If the importer would like to flag a returned transaction as a known duplicate, it may opt to set the special flag "__duplicate__" to True, and the transaction should be treated as a duplicate by the extraction code. This is a way to let the importer use particular information about previously imported transactions in order to flag them as duplicates. For example, if an importer has a way to get a persistent unique id for each of the imported transactions. (See this discussion for context: https://groups.google.com/d/msg/beancount/0iV-ipBJb8g/-uk4wsH2AgAJ)

Parameters

- **file** – A cache.FileMemo instance.
- **existing_entries** – An optional list of existing directives loaded from the ledger which is intended to contain the extracted entries. This is only provided if the user provides them via a flag in the extractor program.

Returns
A list of new, imported directives (usually mostly Transactions) extracted from the file.

`fetchSingle`(*trx*)

`file_account`(*file*)

Return an account associated with the given file.

Note: If you don't implement this method you won't be able to move the files into its preservation hierarchy; the bean-file command won't work.

Also, normally the returned account is not a function of the input file–just of the importer–but it is provided anyhow.

Parameters
file – A cache.FileMemo instance.

Returns
The name of the account that corresponds to this importer.

`identify`(*file*)

Return true if this importer matches the given file.

Parameters
file – A cache.FileMemo instance.

Returns
A boolean, true if this importer can handle this file.

Module contents

tariochbctools.importers.blockchain package

Submodules

tariochbctools.importers.blockchain.importer module

class tariochbctools.importers.blockchain.importer.**Importer**

Bases: ImporterProtocol

An importer for Blockchain data.

extract(*file*, *existing_entries*)

Extract transactions from a file.

If the importer would like to flag a returned transaction as a known duplicate, it may opt to set the special flag "__duplicate__" to True, and the transaction should be treated as a duplicate by the extraction code. This is a way to let the importer use particular information about previously imported transactions in order to flag them as duplicates. For example, if an importer has a way to get a persistent unique id for each of the imported transactions. (See this discussion for context: https://groups.google.com/d/msg/beancount/0iV-ipBJb8g/-uk4wsH2AgAJ)

Parameters

- **file** – A cache.FileMemo instance.
- **existing_entries** – An optional list of existing directives loaded from the ledger which is intended to contain the extracted entries. This is only provided if the user provides them via a flag in the extractor program.

Returns

A list of new, imported directives (usually mostly Transactions) extracted from the file.

file_account(*file*)

Return an account associated with the given file.

Note: If you don't implement this method you won't be able to move the files into its preservation hierarchy; the bean-file command won't work.

Also, normally the returned account is not a function of the input file–just of the importer–but it is provided anyhow.

Parameters

file – A cache.FileMemo instance.

Returns

The name of the account that corresponds to this importer.

identify(*file*)

Return true if this importer matches the given file.

Parameters

file – A cache.FileMemo instance.

Returns

A boolean, true if this importer can handle this file.

Module contents

tariochbctools.importers.cembrastatement package

Submodules

tariochbctools.importers.cembrastatement.importer module

class tariochbctools.importers.cembrastatement.importer.**Importer**(*regexps*, *account*)

Bases: IdentifyMixin, ImporterProtocol

An importer for Cembra Card Statement PDF files.

cleanDecimal(*formattedNumber*)

createBalanceEntry(*file*, *date*, *amt*)

createEntry(*file*, *date*, *amt*, *text*)

extract(*file*, *existing_entries*)

Extract transactions from a file.

If the importer would like to flag a returned transaction as a known duplicate, it may opt to set the special flag "__duplicate__" to True, and the transaction should be treated as a duplicate by the extraction code. This is a way to let the importer use particular information about previously imported transactions in order to flag them as duplicates. For example, if an importer has a way to get a persistent unique id for each of the imported transactions. (See this discussion for context: https://groups.google.com/d/msg/beancount/0iV-ipBJb8g/-uk4wsH2AgAJ)

Parameters

- **file** – A cache.FileMemo instance.
- **existing_entries** – An optional list of existing directives loaded from the ledger which is intended to contain the extracted entries. This is only provided if the user provides them via a flag in the extractor program.

Returns

A list of new, imported directives (usually mostly Transactions) extracted from the file.

file_account(*file*)

Return an account associated with the given file.

Note: If you don't implement this method you won't be able to move the files into its preservation hierarchy; the bean-file command won't work.

Also, normally the returned account is not a function of the input file–just of the importer–but it is provided anyhow.

Parameters

file – A cache.FileMemo instance.

Returns

The name of the account that corresponds to this importer.

getAmount(*debit*, *credit*)

Module contents

tariochbctools.importers.general package

Submodules

tariochbctools.importers.general.mailAdapterImporter module

class tariochbctools.importers.general.mailAdapterImporter.**MailAdapterImporter**(*importers*)

Bases: ImporterProtocol

An importer adapter that fetches file from mails and then calls another importer.

extract(*file*, *existing_entries*)

Extract transactions from a file.

If the importer would like to flag a returned transaction as a known duplicate, it may opt to set the special flag "__duplicate__" to True, and the transaction should be treated as a duplicate by the extraction code. This is a way to let the importer use particular information about previously imported transactions in order to flag them as duplicates. For example, if an importer has a way to get a persistent unique id for each of the imported transactions. (See this discussion for context: https://groups.google.com/d/msg/beancount/0iV-ipBJb8g/-uk4wsH2AgAJ)

Parameters

- **file** – A cache.FileMemo instance.
- **existing_entries** – An optional list of existing directives loaded from the ledger which is intended to contain the extracted entries. This is only provided if the user provides them via a flag in the extractor program.

Returns

A list of new, imported directives (usually mostly Transactions) extracted from the file.

file_account(*file*)

Return an account associated with the given file.

Note: If you don't implement this method you won't be able to move the files into its preservation hierarchy; the bean-file command won't work.

Also, normally the returned account is not a function of the input file–just of the importer–but it is provided anyhow.

Parameters

file – A cache.FileMemo instance.

Returns

The name of the account that corresponds to this importer.

identify(*file*)

Return true if this importer matches the given file.

Parameters

file – A cache.FileMemo instance.

Returns

A boolean, true if this importer can handle this file.

tariochbctools.importers.general.mt940importer module

class tariochbctools.importers.general.mt940importer.**Importer**(*regexps*, *account*)

Bases: IdentifyMixin, ImporterProtocol

An importer for MT940 files.

extract(*file*, *existing_entries*)

Extract transactions from a file.

If the importer would like to flag a returned transaction as a known duplicate, it may opt to set the special flag "__duplicate__" to True, and the transaction should be treated as a duplicate by the extraction code. This is a way to let the importer use particular information about previously imported transactions in order to flag them as duplicates. For example, if an importer has a way to get a persistent unique id for each of the imported transactions. (See this discussion for context: https://groups.google.com/d/msg/beancount/0iV-ipBJb8g/-uk4wsH2AgAJ)

Parameters

- **file** – A cache.FileMemo instance.
- **existing_entries** – An optional list of existing directives loaded from the ledger which is intended to contain the extracted entries. This is only provided if the user provides them via a flag in the extractor program.

Returns
A list of new, imported directives (usually mostly Transactions) extracted from the file.

file_account(*file*)

Return an account associated with the given file.

Note: If you don't implement this method you won't be able to move the files into its preservation hierarchy; the bean-file command won't work.

Also, normally the returned account is not a function of the input file–just of the importer–but it is provided anyhow.

Parameters
file – A cache.FileMemo instance.

Returns
The name of the account that corresponds to this importer.

identify(*file*)

Return true if this importer matches the given file.

Parameters
file – A cache.FileMemo instance.

Returns
A boolean, true if this importer can handle this file.

prepare_narration(*trxdata*)

prepare_payee(*trxdata*)

tariochbctools.importers.general.priceLookup module

class tariochbctools.importers.general.priceLookup.**PriceLookup**(*existing_entries*, *baseCcy: str*)

Bases: object

fetchPrice(*instrument: str*, *date: date*)

fetchPriceAmount(*instrument: str*, *date: date*)

Module contents

tariochbctools.importers.ibkr package

Submodules

tariochbctools.importers.ibkr.importer module

class tariochbctools.importers.ibkr.importer.**Importer**

Bases: ImporterProtocol

An importer for Interactive Broker using the flex query service.

createBuy(*date: date*, *account: str*, *asset: str*, *quantity: Decimal*, *currency: str*, *price: Decimal*, *commission: Amount*, *netCash: Amount*, *baseCcy: str*, *fxRateToBase: Decimal*)

createDividen(*payout: Decimal*, *withholding: Decimal*, *asset: str*, *currency: str*, *date: date*, *priceLookup: PriceLookup*, *description: str*, *account: str*)

extract(*file*, *existing_entries*)

Extract transactions from a file.

If the importer would like to flag a returned transaction as a known duplicate, it may opt to set the special flag "__duplicate__" to True, and the transaction should be treated as a duplicate by the extraction code. This is a way to let the importer use particular information about previously imported transactions in order to flag them as duplicates. For example, if an importer has a way to get a persistent unique id for each of the imported transactions. (See this discussion for context: https://groups.google.com/d/msg/beancount/0iV-ipBJb8g/-uk4wsH2AgAJ)

Parameters

- **file** – A cache.FileMemo instance.
- **existing_entries** – An optional list of existing directives loaded from the ledger which is intended to contain the extracted entries. This is only provided if the user provides them via a flag in the extractor program.

Returns
A list of new, imported directives (usually mostly Transactions) extracted from the file.

file_account(*file*)

Return an account associated with the given file.

Note: If you don't implement this method you won't be able to move the files into its preservation hierarchy; the bean-file command won't work.

Also, normally the returned account is not a function of the input file–just of the importer–but it is provided anyhow.

Parameters
file – A cache.FileMemo instance.

Returns
The name of the account that corresponds to this importer.

getAssetAccount(*account: str*, *asset: str*)

getFeeAccount(*account: str*)

getIncomeAccount(*account: str*)

getLiquidityAccount(*account: str*, *currency: str*)

getReceivableAccount(*account: str*)

identify(*file*)

Return true if this importer matches the given file.

Parameters
file – A cache.FileMemo instance.

Returns
A boolean, true if this importer can handle this file.

matches(*trx*, *t*, *account*)

Module contents

tariochbctools.importers.neon package

Submodules

tariochbctools.importers.neon.importer module

class tariochbctools.importers.neon.importer.**Importer**(*regexps*, *account*)

Bases: IdentifyMixin, ImporterProtocol

An importer for Neon CSV files.

extract(*file*, *existing_entries*)

Extract transactions from a file.

If the importer would like to flag a returned transaction as a known duplicate, it may opt to set the special flag "__duplicate__" to True, and the transaction should be treated as a duplicate by the extraction code. This is a way to let the importer use particular information about previously imported transactions in order to flag them as duplicates. For example, if an importer has a way to get a persistent unique id for each of the imported transactions. (See this discussion for context: https://groups.google.com/d/msg/beancount/0iV-ipBJb8g/-uk4wsH2AgAJ)

Parameters

- **file** – A cache.FileMemo instance.
- **existing_entries** – An optional list of existing directives loaded from the ledger which is intended to contain the extracted entries. This is only provided if the user provides them via a flag in the extractor program.

Returns
A list of new, imported directives (usually mostly Transactions) extracted from the file.

file_account(*file*)

Return an account associated with the given file.

Note: If you don't implement this method you won't be able to move the files into its preservation hierarchy; the bean-file command won't work.

Also, normally the returned account is not a function of the input file–just of the importer–but it is provided anyhow.

Parameters
file – A cache.FileMemo instance.

Returns
The name of the account that corresponds to this importer.

name()

Return a unique id/name for this importer.

Returns
A string which uniquely identifies this importer.

Module contents

tariochbctools.importers.nordigen package

Submodules

tariochbctools.importers.nordigen.importer module

exception tariochbctools.importers.nordigen.importer.**HttpServiceException**

Bases: Exception

class tariochbctools.importers.nordigen.importer.**Importer**

Bases: ImporterProtocol

An importer for Nordigen API (e.g. for Revolut).

extract(*file*, *existing_entries*)

Extract transactions from a file.

If the importer would like to flag a returned transaction as a known duplicate, it may opt to set the special flag "__duplicate__" to True, and the transaction should be treated as a duplicate by the extraction code. This is a way to let the importer use particular information about previously imported transactions in order to flag them as duplicates. For example, if an importer has a way to get a persistent unique id for each of the imported transactions. (See this discussion for context: https://groups.google.com/d/msg/beancount/0iV-ipBJb8g/-uk4wsH2AgAJ)

Parameters

- **file** – A cache.FileMemo instance.
- **existing_entries** – An optional list of existing directives loaded from the ledger which is intended to contain the extracted entries. This is only provided if the user provides them via a flag in the extractor program.

Returns
A list of new, imported directives (usually mostly Transactions) extracted from the file.

file_account(*file*)

Return an account associated with the given file.

Note: If you don't implement this method you won't be able to move the files into its preservation hierarchy; the bean-file command won't work.

Also, normally the returned account is not a function of the input file–just of the importer–but it is provided anyhow.

Parameters
file – A cache.FileMemo instance.

Returns
The name of the account that corresponds to this importer.

`identify`(*file*)

Return true if this importer matches the given file.

Parameters
file – A cache.FileMemo instance.

Returns
A boolean, true if this importer can handle this file.

tariochbctools.importers.nordigen.nordigen_config module

tariochbctools.importers.nordigen.nordigen_config.**build_header**(*token*)

tariochbctools.importers.nordigen.nordigen_config.**check_result**(*result*)

tariochbctools.importers.nordigen.nordigen_config.**create_link**(*token*, *reference*, *bank*)

tariochbctools.importers.nordigen.nordigen_config.**delete_link**(*token*, *reference*)

tariochbctools.importers.nordigen.nordigen_config.**get_token**(*secret_id*, *secret_key*)

tariochbctools.importers.nordigen.nordigen_config.**list_accounts**(*token*)

tariochbctools.importers.nordigen.nordigen_config.**list_bank**(*token*, *country*)

tariochbctools.importers.nordigen.nordigen_config.**main**(*args*)

tariochbctools.importers.nordigen.nordigen_config.**parse_args**(*args*)

tariochbctools.importers.nordigen.nordigen_config.**run**()
Entry point for console_scripts

Module contents

tariochbctools.importers.postfinance package

Submodules

tariochbctools.importers.postfinance.importer module

class tariochbctools.importers.postfinance.importer.**Importer**(*regexps*, *account*, *currency='CHF'*)
Bases: `IdentifyMixin`, `ImporterProtocol`

An importer for PostFinance CSV.

extract(*file*, *existing_entries*)

Extract transactions from a file.

If the importer would like to flag a returned transaction as a known duplicate, it may opt to set the special flag "__duplicate__" to True, and the transaction should be treated as a duplicate by the extraction code. This is a way to let the importer use particular information about previously imported transactions in order to flag them as duplicates. For example, if an importer has a way to get a persistent unique id for each of the imported transactions. (See this discussion for context: https://groups.google.com/d/msg/beancount/0iV-ipBJb8g/-uk4wsH2AgAJ)

Parameters

- **file** – A cache.FileMemo instance.
- **existing_entries** – An optional list of existing directives loaded from the ledger which is intended to contain the extracted entries. This is only provided if the user provides them via a flag in the extractor program.

Returns

A list of new, imported directives (usually mostly Transactions) extracted from the file.

file_account(*file*)

Return an account associated with the given file.

Note: If you don't implement this method you won't be able to move the files into its preservation hierarchy; the bean-file command won't work.

Also, normally the returned account is not a function of the input file–just of the importer–but it is provided anyhow.

Parameters

file – A cache.FileMemo instance.

Returns

The name of the account that corresponds to this importer.

Module contents

tariochbctools.importers.quickfile package

Submodules

tariochbctools.importers.quickfile.importer module

class tariochbctools.importers.quickfile.importer.**Importer**

Bases: ImporterProtocol

An importer for QuickFile

extract(*file*, *existing_entries=None*)

Extract transactions from a file.

If the importer would like to flag a returned transaction as a known duplicate, it may opt to set the special flag "__duplicate__" to True, and the transaction should be treated as a duplicate by the extraction code. This is a way to let the importer use particular information about previously imported transactions in order to flag them as duplicates. For example, if an importer has a way to get a persistent unique id for each of the imported transactions. (See this discussion for context: https://groups.google.com/d/msg/beancount/0iV-ipBJb8g/-uk4wsH2AgAJ)

Parameters

- **file** – A cache.FileMemo instance.
- **existing_entries** – An optional list of existing directives loaded from the ledger which is intended to contain the extracted entries. This is only provided if the user provides them via a flag in the extractor program.

Returns
A list of new, imported directives (usually mostly Transactions) extracted from the file.

file_account(*file*)

Return an account associated with the given file.

Note: If you don't implement this method you won't be able to move the files into its preservation hierarchy; the bean-file command won't work.

Also, normally the returned account is not a function of the input file–just of the importer–but it is provided anyhow.

Parameters
file – A cache.FileMemo instance.

Returns
The name of the account that corresponds to this importer.

identify(*file*)

Return true if this importer matches the given file.

Parameters
file – A cache.FileMemo instance.

Returns
A boolean, true if this importer can handle this file.

class tariochbctools.importers.quickfile.importer.**QuickFile**(*account_number*, *api_key*, *app_id*)

Bases: object

Encapsulate QuickFile API protocol and data types

API_VERSION_SLUG = '1_2'

DOMAIN = 'quickfile.co.uk'

static auth_md5(*account_number*, *api_key*, *submission_number*)

bank_search(*account_number*, *transaction_count*, *from_date=None*, *to_date=None*)

request_header()

class tariochbctools.importers.quickfile.importer.**QuickFileBankSearch**(*MetaData*, *Transactions*)

Bases: NamedTuple

MetaData: *QuickFileResponseMetaData*

Alias for field number 0

Transactions: Dict[str, List[*QuickFileTransaction*]]

Alias for field number 1

class tariochbctools.importers.quickfile.importer.**QuickFileResponseMetaData**(*RecordsetCount*, *ReturnCount*, *BankName*, *BankType*, *AccountNo*, *SortCode*, *Currency*, *CurrentBalance*)

Bases: NamedTuple

AccountNo: str
Alias for field number 4

BankName: str
Alias for field number 2

BankType: str
Alias for field number 3

Currency: str
Alias for field number 6

CurrentBalance: str
Alias for field number 7

RecordsetCount: int
Alias for field number 0

ReturnCount: int
Alias for field number 1

SortCode: str
Alias for field number 5

class tariochbctools.importers.quickfile.importer.**QuickFileTransaction**(*TransactionDate: str*, *Reference: str*, *Amount: str*, *TagStatus: str*, *TransactionId: str*)

Bases: NamedTuple

Transaction data from QuickFile transaction API

Amount: str
Alias for field number 2

Reference: str
Alias for field number 1

TagStatus: str
Alias for field number 3

TransactionDate: str
Alias for field number 0

TransactionId: str
Alias for field number 4

to_beancount_transaction(*local_account*, *currency*, *invert_sign=False*)

Module contents

tariochbctools.importers.raiffeisench namespace

Submodules

tariochbctools.importers.raiffeisench.importer module

class tariochbctools.importers.raiffeisench.importer.**RaiffeisenCHImporter**(*regexps*, *account*)

Bases: *Importer*

An importer for MT940 from Raiffeisen CH

prepare_narration(*trxdata*)

prepare_payee(*trxdata*)

tariochbctools.importers.revolut package

Submodules

tariochbctools.importers.revolut.importer module

class tariochbctools.importers.revolut.importer.**Importer**(*regexps*, *account*, *currency*)

Bases: IdentifyMixin, ImporterProtocol

An importer for Revolut CSV files.

extract(*file*, *existing_entries*)

Extract transactions from a file.

If the importer would like to flag a returned transaction as a known duplicate, it may opt to set the special flag "__duplicate__" to True, and the transaction should be treated as a duplicate by the extraction code. This is a way to let the importer use particular information about previously imported transactions in order to flag them as duplicates. For example, if an importer has a way to get a persistent unique id for each of the imported transactions. (See this discussion for context: https://groups.google.com/d/msg/beancount/0iV-ipBJb8g/-uk4wsH2AgAJ)

Parameters

- **file** – A cache.FileMemo instance.
- **existing_entries** – An optional list of existing directives loaded from the ledger which is intended to contain the extracted entries. This is only provided if the user provides them via a flag in the extractor program.

Returns

A list of new, imported directives (usually mostly Transactions) extracted from the file.

file_account(*file*)

Return an account associated with the given file.

Note: If you don't implement this method you won't be able to move the files into its preservation hierarchy; the bean-file command won't work.

Also, normally the returned account is not a function of the input file–just of the importer–but it is provided anyhow.

Parameters
file – A cache.FileMemo instance.

Returns
The name of the account that corresponds to this importer.

name()
Return a unique id/name for this importer.

Returns
A string which uniquely identifies this importer.

Module contents

tariochbctools.importers.schedule package

Submodules

tariochbctools.importers.schedule.importer module

class tariochbctools.importers.schedule.importer.**Importer**
Bases: ImporterProtocol

An importer for Scheduled/Recurring Transactions.

createForDate(*trx*, *date*)

extract(*file*, *existing_entries*)
Extract transactions from a file.

If the importer would like to flag a returned transaction as a known duplicate, it may opt to set the special flag "__duplicate__" to True, and the transaction should be treated as a duplicate by the extraction code. This is a way to let the importer use particular information about previously imported transactions in order to flag them as duplicates. For example, if an importer has a way to get a persistent unique id for each of the imported transactions. (See this discussion for context: https://groups.google.com/d/msg/beancount/0iV-ipBJb8g/-uk4wsH2AgAJ)

Parameters

- **file** – A cache.FileMemo instance.
- **existing_entries** – An optional list of existing directives loaded from the ledger which is intended to contain the extracted entries. This is only provided if the user provides them via a flag in the extractor program.

Returns
A list of new, imported directives (usually mostly Transactions) extracted from the file.

file_account(*file*)
Return an account associated with the given file.

Note: If you don't implement this method you won't be able to move the files into its preservation hierarchy; the bean-file command won't work.

Also, normally the returned account is not a function of the input file–just of the importer–but it is provided anyhow.

Parameters
file – A cache.FileMemo instance.

Returns
The name of the account that corresponds to this importer.

`identify`(*file*)

Return true if this importer matches the given file.

Parameters
file – A cache.FileMemo instance.

Returns
A boolean, true if this importer can handle this file.

Module contents

tariochbctools.importers.swisscard package

Submodules

tariochbctools.importers.swisscard.importer module

`class tariochbctools.importers.swisscard.importer.SwisscardImporter`(*regexps*, *account*)

Bases: `IdentifyMixin`, `ImporterProtocol`

An importer for Swisscard's cashback CSV files.

`extract`(*file*, *existing_entries*)

Extract transactions from a file.

If the importer would like to flag a returned transaction as a known duplicate, it may opt to set the special flag "__duplicate__" to True, and the transaction should be treated as a duplicate by the extraction code. This is a way to let the importer use particular information about previously imported transactions in order to flag them as duplicates. For example, if an importer has a way to get a persistent unique id for each of the imported transactions. (See this discussion for context: https://groups.google.com/d/msg/beancount/0iV-ipBJb8g/-uk4wsH2AgAJ)

Parameters

- **file** – A cache.FileMemo instance.
- **existing_entries** – An optional list of existing directives loaded from the ledger which is intended to contain the extracted entries. This is only provided if the user provides them via a flag in the extractor program.

Returns
A list of new, imported directives (usually mostly Transactions) extracted from the file.

`file_account`(*file*)

Return an account associated with the given file.

Note: If you don't implement this method you won't be able to move the files into its preservation hierarchy; the bean-file command won't work.

Also, normally the returned account is not a function of the input file–just of the importer–but it is provided anyhow.

Parameters
file – A cache.FileMemo instance.

Returns
The name of the account that corresponds to this importer.

name()

Return a unique id/name for this importer.

Returns
A string which uniquely identifies this importer.

Module contents

tariochbctools.importers.transferwise package

Submodules

tariochbctools.importers.transferwise.importer module

class tariochbctools.importers.transferwise.importer.**Importer**(**args*, ***kwargs*)

Bases: ImporterProtocol

An importer for Transferwise using the API.

extract(*file*, *existing_entries*)

Extract transactions from a file.

If the importer would like to flag a returned transaction as a known duplicate, it may opt to set the special flag "__duplicate__" to True, and the transaction should be treated as a duplicate by the extraction code. This is a way to let the importer use particular information about previously imported transactions in order to flag them as duplicates. For example, if an importer has a way to get a persistent unique id for each of the imported transactions. (See this discussion for context: https://groups.google.com/d/msg/beancount/0iV-ipBJb8g/-uk4wsH2AgAJ)

Parameters

- **file** – A cache.FileMemo instance.
- **existing_entries** – An optional list of existing directives loaded from the ledger which is intended to contain the extracted entries. This is only provided if the user provides them via a flag in the extractor program.

Returns
A list of new, imported directives (usually mostly Transactions) extracted from the file.

file_account(*file*)

Return an account associated with the given file.

Note: If you don't implement this method you won't be able to move the files into its preservation hierarchy; the bean-file command won't work.

Also, normally the returned account is not a function of the input file–just of the importer–but it is provided anyhow.

Parameters
file – A cache.FileMemo instance.

Returns
The name of the account that corresponds to this importer.

`identify`(*file*)

Return true if this importer matches the given file.

Parameters
file – A cache.FileMemo instance.

Returns
A boolean, true if this importer can handle this file.

Module contents

tariochbctools.importers.truelayer package

Submodules

tariochbctools.importers.truelayer.importer module

`class tariochbctools.importers.truelayer.importer.Importer`

Bases: `ImporterProtocol`

An importer for Truelayer API (e.g. for Revolut).

`extract`(*file*, *existing_entries=None*)

Extract transactions from a file.

If the importer would like to flag a returned transaction as a known duplicate, it may opt to set the special flag "__duplicate__" to True, and the transaction should be treated as a duplicate by the extraction code. This is a way to let the importer use particular information about previously imported transactions in order to flag them as duplicates. For example, if an importer has a way to get a persistent unique id for each of the imported transactions. (See this discussion for context: https://groups.google.com/d/msg/beancount/0iV-ipBJb8g/-uk4wsH2AgAJ)

Parameters

- **file** – A cache.FileMemo instance.
- **existing_entries** – An optional list of existing directives loaded from the ledger which is intended to contain the extracted entries. This is only provided if the user provides them via a flag in the extractor program.

Returns
A list of new, imported directives (usually mostly Transactions) extracted from the file.

`file_account`(*file*)

Return an account associated with the given file.

Note: If you don't implement this method you won't be able to move the files into its preservation hierarchy; the bean-file command won't work.

Also, normally the returned account is not a function of the input file–just of the importer–but it is provided anyhow.

Parameters
file – A cache.FileMemo instance.

Returns
The name of the account that corresponds to this importer.

`identify`(*file*)

Return true if this importer matches the given file.

Parameters
file – A cache.FileMemo instance.

Returns
A boolean, true if this importer can handle this file.

Module contents

tariochbctools.importers.viseca package

Submodules

tariochbctools.importers.viseca.importer module

`class tariochbctools.importers.viseca.importer.Importer`(*regexps*, *account*)

Bases: `IdentifyMixin`, `ImporterProtocol`

An importer for Viseca One Card Statement PDF files.

`createEntry`(*file*, *date*, *entryAmount*, *text*)

`extract`(*file*, *existing_entries*)

Extract transactions from a file.

If the importer would like to flag a returned transaction as a known duplicate, it may opt to set the special flag "__duplicate__" to True, and the transaction should be treated as a duplicate by the extraction code. This is a way to let the importer use particular information about previously imported transactions in order to flag them as duplicates. For example, if an importer has a way to get a persistent unique id for each of the imported transactions. (See this discussion for context: https://groups.google.com/d/msg/beancount/0iV-ipBJb8g/-uk4wsH2AgAJ)

Parameters

- **file** – A cache.FileMemo instance.
- **existing_entries** – An optional list of existing directives loaded from the ledger which is intended to contain the extracted entries. This is only provided if the user provides them via a flag in the extractor program.

Returns
A list of new, imported directives (usually mostly Transactions) extracted from the file.

`file_account`(*file*)

Return an account associated with the given file.

Note: If you don't implement this method you won't be able to move the files into its preservation hierarchy; the bean-file command won't work.

Also, normally the returned account is not a function of the input file–just of the importer–but it is provided anyhow.

Parameters
file – A cache.FileMemo instance.

Returns
The name of the account that corresponds to this importer.

Module contents

tariochbctools.importers.zak package

Submodules

tariochbctools.importers.zak.importer module

class tariochbctools.importers.zak.importer.**Importer**(*regexps*, *account*)

Bases: IdentifyMixin, ImporterProtocol

An importer for Bank Cler ZAK PDF files files.

cleanNumber(*number*)

createBalanceEntry(*file*, *date*, *amt*)

createEntry(*file*, *date*, *amt*, *text*)

extract(*file*, *existing_entries*)

Extract transactions from a file.

If the importer would like to flag a returned transaction as a known duplicate, it may opt to set the special flag "__duplicate__" to True, and the transaction should be treated as a duplicate by the extraction code. This is a way to let the importer use particular information about previously imported transactions in order to flag them as duplicates. For example, if an importer has a way to get a persistent unique id for each of the imported transactions. (See this discussion for context: https://groups.google.com/d/msg/beancount/0iV-ipBJb8g/-uk4wsH2AgAJ)

Parameters

- **file** – A cache.FileMemo instance.
- **existing_entries** – An optional list of existing directives loaded from the ledger which is intended to contain the extracted entries. This is only provided if the user provides them via a flag in the extractor program.

Returns
A list of new, imported directives (usually mostly Transactions) extracted from the file.

file_account(*file*)

Return an account associated with the given file.

Note: If you don't implement this method you won't be able to move the files into its preservation hierarchy; the bean-file command won't work.

Also, normally the returned account is not a function of the input file–just of the importer–but it is provided anyhow.

Parameters
file – A cache.FileMemo instance.

Returns
The name of the account that corresponds to this importer.

Module contents

tariochbctools.importers.zkb package

Submodules

tariochbctools.importers.zkb.importer module

class tariochbctools.importers.zkb.importer.**ZkbImporter**(*regexps*, *account*)

Bases: *Importer*

prepare_narration(*trxdata*)

prepare_payee(*trxdata*)

Module contents

Module contents

tariochbctools.plugins package

Subpackages

tariochbctools.plugins.prices package

Submodules

tariochbctools.plugins.prices.ibkr module

class tariochbctools.plugins.prices.ibkr.**Source**

Bases: Source

get_historical_price(*ticker*, *time*)

Return the historical price found for the symbol at the given date.

This could be the price of the close of the day, for instance. We assume that there is some single price representative of the day.

Parameters

- **ticker** – A string, the ticker to be fetched by the source. This ticker may include structure, such as the exchange code. Also note that this ticker is source-specified, and is not necessarily the same value as the commodity symbol used in the Beancount file.
- **time** – The timestamp at which to query for the price. This is a timezone-aware timestamp you can convert to any timezone. For past dates we query for a time that is equivalent to 4pm in the user's timezone.

Returns
A SourcePrice instance. If the price could not be fetched, None is returned and another source should be consulted. There is never any guarantee that a price source will be able to fetch its value; client code must be able to handle this. Also note that the price's returned time must be timezone-aware.

`get_latest_price`(*ticker: str*)

Fetch the current latest price. The date may differ.

This routine attempts to fetch the most recent available price, and returns the actual date of the quoted price, which may differ from the date this call is made at. {1cfa25e37fc1}

Parameters
ticker – A string, the ticker to be fetched by the source. This ticker may include structure, such as the exchange code. Also note that this ticker is source-specified, and is not necessarily the same value as the commodity symbol used in the Beancount file.

Returns
A SourcePrice instance. If the price could not be fetched, None is returned and another source should be consulted. There is never any guarantee that a price source will be able to fetch its value; client code must be able to handle this. Also note that the price's returned time must be timezone-aware.

Module contents

Submodules

tariochbctools.plugins.check_portfolio_sum module

A plugin that verifies that on each transaction, all the "portfolios" have the same weight.

`class tariochbctools.plugins.check_portfolio_sum.DifferentWeightPerPortfolio`(*source*, *message*, *entry*)

Bases: `tuple`

entry
Alias for field number 2

message
Alias for field number 1

source
Alias for field number 0

`class tariochbctools.plugins.check_portfolio_sum.NonZeroWeightPerPortfolio`(*source*, *message*, *entry*)

Bases: `tuple`

entry
Alias for field number 2

message
Alias for field number 1

source
Alias for field number 0

tariochbctools.plugins.check_portfolio_sum.**check**(*entries*, *options_map*)

tariochbctools.plugins.generate_base_ccy_prices module

A plugin that inserts an additional price to the base rate by applying fx rate to a price.

tariochbctools.plugins.generate_base_ccy_prices.**generate**(*entries*, *options_map*, *baseCcy*)

Module contents

Beancount plugins.

Module contents

CHAPTER

TWO

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

t

INDEX

F

L

M

N

P

Q

R

S

T

Z