Prices in Beancount

Martin Blais, December 2015

http://furius.ca/beancount/doc/prices

Introduction

Processing a Beancount file is, by definition, constrained to the contents of the file itself. In particular, the latest prices of commodities are *never* fetched automatically from the internet. This is by design, so that any run of a report is deterministic and can also be run offline. No surprises.

However, we do need access to price information in order to compute market values of assets. To this end, Beancount provides a Price directive which can be used to fill its in-memory price database by inserting these price points inline in the input file:

```
2015-11-20 price ITOT    95.46 USD
2015-11-20 price LQD    115.63 USD
2015-11-21 price USD   1.33495 CAD
…
```

Of course, you could do this manually, looking up the prices online and writing the directives yourself. But for assets which are traded publicly you can automate it, by invoking some code that will download prices and write out the directives for you.

Beancount comes with some tools to help you do this. This document describes these tools.

The Problem

In the context of maintaining a Beancount file, we have a few particular needs to address.

First, we cannot expect the user to always update the input file in a timely manner. This means that we have to be able to fetch not only the latest prices, but also historical prices, from the past. Beancount provides an interface to provide these prices and a few implementations (fetching from Yahoo! Finance and from Google Finance).

Second, we don't want to store too many prices for holdings which aren't relevant to us at a particular point in time. The list of assets held in a file varies over time. Ideally we would want to include the list of prices for just those assets, while they are held. The Beancount price maintenance tool is able to figure out which commodities it needs to fetch prices for at a particular date.

Third, we don't want to fetch the same price if it already appears in the input file. The tools detect this and skip those prices. There's also a caching mechanism that avoids redundant network calls.

Finally, while we provide some basic implementations of price sources, we cannot provide such codes for all possible online sources and websites. The problem is analogous to that of importing and extracting data from various institutions. In order to address that, we provide a mechanism for **extensibility**, a way that you can implement your own price source fetcher in a Python module and point to it from your input file by specifying the module's name as the source for that commodity.

The “bean-price” Tool

Beancount comes with a “bean-price” command-line tool that integrates the ideas above. By default, this script accepts a list of Beancount input filenames, and fetches prices required to compute latest market values for current positions held in accounts:

```
bean-price /home/joe/finances/joe.beancount
```

It is also possible to provide a list of specific price fetching jobs to run, e.g.,

```
bean-price -e USD:google/TSE:XUS CAD:mysources.morningstar/RBF1005
```

These jobs are run concurrently so it should be fairly fast.

Source Strings

The general format of each of these "job source strings" is

```
<quote-currency>:<module>/[^]<symbol>
```

For example:

```
USD:beancount.prices.sources.google/NASDAQ:AAPL
```

The “quote-currency” is the currency the Commodity is quoted in. For example, shares of Apple are quoted in US dollars.

The "module" is the name of a Python module that contains a Source class which can be instantiated and which connect to a data source to extract price data. These modules are automatically imported and a Source class therein instantiated in order to pull the price from the particular online source they support. This allows you to write your own fetcher codes without having to modify this script. Your code can be placed anywhere on your Python PYTHONPATH and you should not have to modify Beancount itself for this to work.

The “symbol” is a string that is fed to the price fetcher to lookup the currency. For example, Apple shares trade on the Nasdaq, and the corresponding symbol in the Google Finance source is “NASDAQ:AAPL”. Other price sources may have a different symbology, e.g., some may require the asset's CUSIP.

Default implementations of price sources are provided; we provide fetchers for Yahoo! Finance or Google Finance, which cover a large universe of common public investment types (e.g. stock and some mutual funds). As a convenience, the module name is always first searched under the "`beancount.prices.sources`" package, where those implementations live. This is how, for example, in order to use the provided Yahoo! Finance data fetcher you don't have to write all of "`beancount.prices.sources.yahoo/AAPL`" but you can simply use "`yahoo/AAPL`".

Fallback Sources

In practice, fetching prices online often fails. Data sources typically only support some limited number of assets and even then, the support may vary between assets. As an example, Google Finance supports historical prices for stocks, but does not return historical prices for currency instruments (these restrictions may be more related to contractual arrangements between them and the data providers upstream than with technical limitations).

To this extent, a source string may provide multiple sources for the data, separated with commas. For example:

```
USD:google/CURRENCY:GBPUSD,yahoo/GBPUSD
```

Each source is tried in turn, and if one fails to return a valid price, the next source is tried as a fallback. The hope is that at least one of the specified sources will work out.

Inverted Prices

Sometimes, prices are only available for the inverse of an instrument. This is often the case for currencies. For example, the price of Canadian dollars quoted in US dollars is provided by the USD/CAD market, which gives the price of a US dollar in Canadian dollars (the inverse). In order use this, you can prepend "^" to the instrument name to instruct the tool to compute the inverse of the fetched price:

```
USD:google/^CURRENCY:USDCAD
```

If a source price is to be inverted, like this, the precision could be different than what is fetched. For instance, if the price of USD/CAD is 1.32759, for the above directive to price the “CAD” instrument it would output this:

```
2015-10-28 price CAD  0.753244601119 USD
```

By default, inverted rates will be rounded similarly to how other Price directives were rounding those numbers.

As you may now, Beancount's in-memory price database works in both directions (the reciprocals of all rates are stored automatically). So if you prefer to have the output Price entries with swapped currencies instead of inverting the rate number itself, you can use the --swap-inverted option. In the previous example for the price of CAD, it would output this:

```
2015-10-28 price USD   1.32759 CAD
```

Date

By default, the latest prices for the assets are pulled in. You can use an option to fetch prices for a desired date in the past instead:

```
bean-price --date=2015-02-03 …
```

If you are using an input file to specify the list of prices to be fetched, the tool will figure out the list of assets held on the books *at that time* and fetch historical prices for those assets only.

Caching

Prices are automatically cached (if current and latest, prices are cached for only a short period of time, about half an hour). This is convenient when you're having to run the script multiple times in a row for troubleshooting.

You can disable the cache with an option:

```
bean-price --no-cache
```

You can also instruct the script to clear the cache before fetching its prices:

```
bean-price --clear-cache
```

Prices from a Beancount Input File

Generally, one uses a Beancount input file to specify the list of currencies to fetch. In order to do that, you should have Commodity directives in your input file for each of the currencies you mean to fetch prices for, like this:

```
2007-07-20 commodity VEA
  price: "USD:google/NYSEARCA:VEA"
```

The "price" metadata should contain a list of price source strings. For example, a stock product might look like this:

```
2007-07-20 commodity CAD
  price: "USD:google/CURRENCY:USDCAD,yahoo/USDCAD"
```

While a currency may have multiple target currencies it needs to get converted to:

```
1990-01-01 commodity GBP
  price: "USD:yahoo/GBPUSD CAD:yahoo/GBPCAD CHF:yahoo/GBPCHF"
```

Which Assets are Fetched

There are many ways to compute a list of commodities with needed prices from a Beancount input file:

1. **Commodity directives.** The list of all Commodity directives with “price” metadata present in the file. For each of those holdings, the directive is consulted and its "`price`" metadata field is used to specify where to fetch prices from.
2. **Commodities of assets held at cost.** Prices for all the holdings that were seen held at cost at a particular date. Because these holdings are held at cost, we can assume there is a corresponding time-varying price for their commodity.
3. **Converted commodities.** Prices for holdings which were price-converted from some other commodity in the past (i.e., converting some cash in a currency from another).

By default, the list of tickers to be fetched includes only the intersection of these three lists. This is because the most common usage of this script is to fetch missing prices for a particular date, and only the needed ones.

Inactive commodities. You can use the “`--inactive`” option to fetch the entire set of prices from (1), regardless of asset holdings determined in (2) and (3).

Undeclared commodities. Commodities without a corresponding “Commodity” directive will be ignored by default. To include the full list of commodities seen in an input file, use the “`--undeclared`” option.

Clobber. Existing price directives for the same data are excluded by default, since the price is already in the file. You can use “`--clobber`” to ignore existing price directives and avoid filtering out what is fetched.

Finally, you can use “--all” to include inactive and undeclared commodities and allow clobbering existing ones. You probably don't want to use that other than for testing.

If you'd like to do some troubleshooting and print out the list of seen commodities, use the "`--verbose`" option twice, i.e., "`-vv`". You can also just print the list of prices to be fetched with the "`--dry-run`" option, which stops short of actually fetching the missing prices.

Conclusion

Writing Your Own Script

If the workflow defined by this tool does not fit your needs and you would like to cook up your own script, you should not have to start from scratch; you should be able to reuse the existing price fetching code to do that. I'm hoping to provide a few examples of such scripts in the experiments directory. For example, given an existing file it would be convenient to fetch all prices of assets every Friday in order to fill up a history of missing prices. Another example would be to fetch all price directives required in order to correctly compute investment returns in the presence of contributions and distributions.

Contributions

If this workflow suits your needs well and you'd like to contribute some price source fetcher to Beancount, please contact the mailing-list. I'm open to including very general fetchers that have been very carefully unit-tested and used for a while.