Beancount I/O Plugins Proposal

Author: Tim Sweña
Created: 2026-03-06
Last updated: Mar 7, 2026

Edit history:

Date	Summary of changes
2026-03-06	Initial draft.
2026-03-07	Updated to use a .beanrc file instead of environment variables.

Overview

I/O Plugins intercept and transform the raw file input/output (I/O) stream *before* Beancount's standard parser sees the content. They act as a pre-parser layer, allowing the processing of non-standard Beancount source files, like encrypted or compressed files, or content embedded in other formats (like Markdown).

As the Beancount community develops, it could even be used to create translation layers between different enhancements to the Beancount format.

Examples

Because plugins must run *before* the configuration within the Beancount file itself can be read, users would primarily interact with I/O Plugins through environment variables. This could be extended to local rc/configuration/env/toml files in the future.

```
$ cat .beanrc
[beancount.plugins.io]
plugins="beancount.plugins.io.literate"
```

```
$ bean-query my_journal.md balance Expenses
```

Separate multiple plugins using commas. These are executed in order.

```
$ cat .beanrc
[beancount.plugins.io]
plugins="beancount.plugins.io.literate,mypackage.translate_hledger_to_
beancount"

$ bean-query my_journal.md balance Expenses
```

Motivation

Users need I/O Plugins to process Beancount data stored in formats or locations that the standard parser cannot directly handle:

Use Case	Problem Solved	Example Plugin
Security/Compression	Reading files that are encrypted (e.g., GPG) or compressed (`.gz`).	`beancount.plugins.io.encryption`
Source Flexibility	Treating custom data streams as Beancount files.	Allowing older hledger / ledger files to be imported without having to rewrite the old files.
Literate Programming	Extracting Beancount code blocks embedded within documents (e.g., Markdown).	`beancount.plugins.io.literate`

Design

1. **Built-in Plugins (Automatic):**

    - Handling encrypted files is rewritten to be configured as a **built-in** plugin that runs automatically.
2. **User-Defined Plugins (Configuration):**
    - To use a custom plugin (e.g., a literate programming extractor), the user would create a .beanrc file next to their main beancount ledger before running Beancount.

Plugin interface

This proposal uses the same convention as other plugins: a Python module with an entry point defined via the __plugins__ attribute.

An I/O plugin function should return an iterable of tuples (or yield tuples if it is a generator), where the tuple has 3 parts:

1. An io.IOBase object (usually the original file or an io.BytesIO object)
2. A filename (this can be transformed to indicate sub-elements of the file for easier tracing of errors through multiple plugins).
3. A list of errors (possibly empty).

Example plugin

```
__plugins__ = ["shout"]

def shout(file_io, filename):
    contents = file_io.read()
    if isinstance(contents, bytes):
        contents = contents.decode('utf-8')

    output = io.StringIO()
    for line in contents.split("\n"):
        output.write(f"{line.upper()}\n")
    return [
        (
            io.BytesIO(output.getvalue().encode('utf-8')),
            f"{filename}:{block}",
            errors
```

```
        )
    ]
```

Comparison to other plugins

The key distinction is the stage of processing at which the plugin operates:

Feature	I/O Plugins (Proposed)	Directive Plugins (Existing)	Importers (Existing)
Input/Timing	Operates on the **raw file stream** (`io.IOBase`) *before* parsing.	Operates on **parsed directives** (`data.Directive`) *after* parsing.	Operates on **external financial files** (e.g., CSV, OFX) to *generate* directives.
Purpose	Transform, decrypt, decompress, or split the source file itself.	Modify the list of entries/directives (e.g., adding metadata, enforcing rules).	Convert third-party data into Beancount entries.
Configuration	A .beanrc config file or hardcoded (built-in).	Options directive within the Beancount file.	Configuration within the importer file.
Output	One or more transformed I/O streams and new reporting filenames.	A modified list of Beancount entries and errors.	A list of generated Beancount entries.

Resources

- https://github.com/beancount/beancount/issues/388#issuecomment-4015245718 -- Outdated documentation that inspired this feature.
- https://github.com/tswast/beancount/pull/1 -- prototype implementation

Alternatives considered

Use an environment variable to set plugins

Instead of introducing a .beanrc file, we could use an environment variable (e.g. export `BEANCOUNT_PLUGINS_IO='my_plugin_module,another_one'`) to configure the I/O plugins.

Note: If this is desired, this could still be used in combination with the .beanrc file. For example, twine looks in the following places for configuration:

1. Environment variables
2. Local .pypirc, set with the --config option.
3. Home directory .pypirc

Pros:

- No need for additional files.

Cons:

- Environment variables are easy to forget to set. They require additional tooling like direnv to set automatically per project.
- Environment variables aren't included in the repository (unless using additional tooling, like direnv). This means you could end up with a ledger you don't know how to parse because of some custom plugin that's not enabled.