aurorae

Rebeca Sarai, Marcos Felipe, Mariane Pastor, Thiago Paim, Jaênia

Aug 20, 2021

USER GUIDE

CHAPTER

ONE

INSTALL

Aurorae can be installed from PyPI with tools like `pip`:

```
$ pip install aurorae
```

1.1 Requirements

1.1.1 Libraries

- Python: >3
- *openpyxl*: >3.0.7
- *pydantic*: >1.8.2

CHAPTER

TWO

INTRODUCTION

2.1 What is a CNAB240

CNAB240 is a standard defined by FEBRABAN and implemented by Brazilian banks for the exchange of information between companies and banks. Based on the information necessary for the implementation of each type of service, the standard defines a set of records/fields that must compose the exchange file.

The standard covers the following types of services:

- **Payment via account credit (Pagamento através de crédito em conta, cheque, OP, DOC ou pagamento com autenticação)**
- Payment of bills of exchange (Pagamento de títulos de Cobrança)
- Tax payments (Pagamento de Tributos)
- Billing Securities (Títulos em Cobrança)
- Electronic Payment Slip (Boleto de Pagamento Eletrônico)
- Payer's Claim (Alegação do Pagador)
- Checking Account Statement for Bank Reconciliation (Extrato de Conta Corrente para Conciliação Bancária)
- Direct debit (Débito em Conta Corrente)
- Vendor (Vendor)
- Custody of Checks (Custódia de Cheques)
- Statement for Cash Management (Extrato para Gestão de Caixa)
- Loan with Payroll Consignment (Empréstimo com Consignação em Folha de Pagamento)
- Buy (Compror)

Aurorae only supports Payment via account credit.

2.2 CNAB versions

The versions of the file are identified through a code with the following composition: VV.R

- VV: Version number
- R: Release number

Aurorae only supports the version 10.7 of CNAB240.

2.3 File Structure

The exchange file is composed of a file header record, one or more service/product batches and a file trailer record.

```
File Header
    Batch
        Batch Header
        Initial Batch Records
        Segment Detail Records
        Final Batch Records
        Batch Trailer
Filte Trailer
```

A single file can contain multiple batches of different Services. Each of the items described above represent one line of the file with the size of 240 bytes. Individuals fields for each line are aligned according to the type of the field:

- Numeric Fields (Num) = Always on the right and padded with zeros on the left.
- Alphanumeric Fields (Alpha) = Always on the left and filled with blanks on the right.

2.3.1 Payment via account credit

This service batch is composed of a batch header record, one or more detail records, and a batch trailer record, besides, the same batch can be used to send information or to receive information. The types of detail segments for this batch are:

```
Shipping
    A (mandatory)
    B (mandatory)
    C (optional)

Return
    A (mandatory)
    B (optional)
    C (optional)
```

Aurorae doesn't support sending the detail segment C yet.

2.4 Summary

- Aurorae only supports Payment via account credit
- Aurorae only supports the version 10.7 of CNAB240
- Aurorae doesn't support sending the detail segment C

CHAPTER

THREE

USAGE

This guide will teach you how to generate CNAB240 files for bulk payments and extend the library for different formats.

3.1 Before starting

Aurora uses Python type hinting for data validation and generation of fixed-width CNAB 240 files. First, the library receives a spreadsheet that must match the Pydantic model Spreadsheet. A general handler parses the initial data to an intermediary representation used by the CNAB240 module to generate files. The library supports different types of inputs through the creation of new providers; check the spreadsheet provider for an example.

The library is organized to hide the complex logic of the file generation and expose a simple model for data handling. As a consequence, we have three areas:

- Spreadsheet models: each provider has their class, which maps perfectly the source data
- Payroll models: represents the default format used to generate the CNAB files
- CNAB models: responsible for transforming the payroll into a CNAB file, contains all the complexity of the file.

3.2 Data format

The supported format is an xlsx spreadsheet with three sheets: Company, Employees, and Payments; each sheet has specific fields that match precisely the ones defined on Spreadsheet. The information of the spreadsheet includes:

```
Company:
    - bank information
    - details of the company
    - address information

Employee
    - bank information
    - address information
    - personal information

Payment:
    - employee
    - details of the payment
```

A sample file can be found here.

3.3 Generating the file

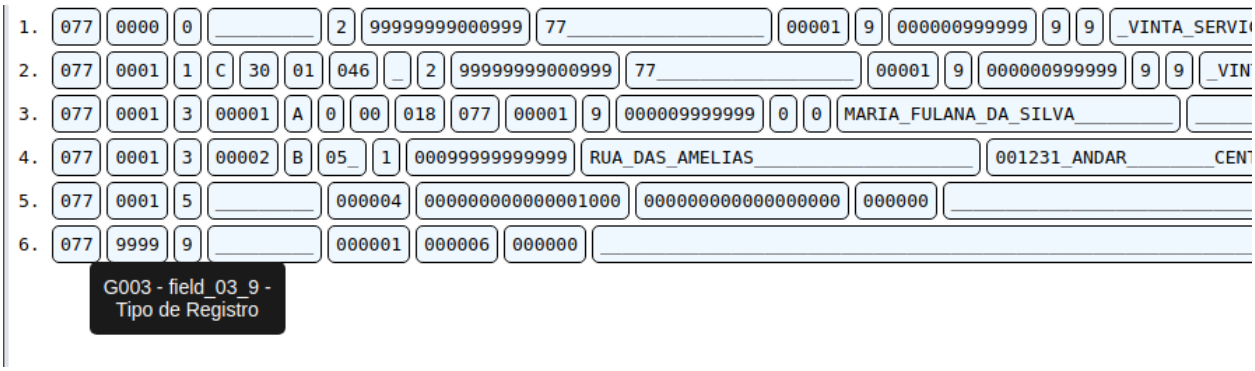
After installing aurorae with pip, the commands `generate_cnab` and `generate_cnab_sample` are available. To check CLI options, run:

```
$ generate_cnab --help
```

The mandatory parameter is the input file, you can also pass an output file name through `--output_filename`.

3.4 Validating the file

A debug file is also generated when using the command line. The file consists on an HTML file with the fields highlighted and the details specified:



A sample debug file can be found here.

3.5 Adding new data formats

This library was designed to support multiple data formats (for details check our ADR). If you are trying to support new formats, you need to:

1. Create a new *provider*
2. Replicate your new format as pydantic models (like the ones on Spreadsheet)
3. Create the *_mapping* on your pydantic models to our standard *Payroll* model (like the ones on Spreadsheet)
4. Replicate the handler behavior using your newly created class

Feel free to open a Pull Request with this new format.

3.6 Example

The library comes with a built-in configuration to generate a sample cnab:

```
$ generate_cnab_sample
```

CHAPTER

FOUR

CONTRIBUTING

Contributions are welcome, and they are greatly appreciated! Every little bit helps, and credit will always be given.

You can contribute in many ways:

4.1 Types of Contributions

4.1.1 Report Bugs

Report bugs at https://github.com/vintasoftware/aurorae/issues

Before reporting a bug, please double-check the requirements: https://github.com/vintasoftware/aurorae/blob/main/README.md#requirements

If you think you really found a bug, please create a GitHub issue and use the "Bug report" template.

4.1.2 Fix Bugs

Look through the GitHub issues for bugs. Anything tagged with "bug" and "help wanted" is open to whoever wants to implement it. Please comment on the issue saying you're working in a solution.

4.1.3 Implement Features

Look through the GitHub issues for features. Anything tagged with "enhancement" and "help wanted" is open to whoever wants to implement it. Please comment on the issue saying you're working on a solution.

4.1.4 Write Documentation

aurorae could always use more documentation, whether as part of the official docs, in docstrings, or even on the web in blog posts, articles, and such.

4.1.5 Submit Feedback

If you have a suggestion, concern, or want to propose a feature, please create a GitHub issue and use the "New feature" template.

4.2 Get Started!

Ready to contribute? Please read our Code of Conduct: https://github.com/vintasoftware/aurorae/blob/main/CODE_OF_CONDUCT.md

Now, here's how to set up *aurorae* for local development.

1. Fork the *aurorae* repo on GitHub.
2. Clone your fork locally:

```
$ git clone git@github.com:your_name_here/aurorae.git
```

3. Install Poetry: https://python-poetry.org/docs/#installation
4. Install your local copy into a virtualenv. Assuming you have virtualenvwrapper installed, this is how you set up your fork for local development:

```
$ mkvirtualenv aurorae
$ cd aurorae/
$ poetry install
```

5. Create a branch for local development:

```
$ git checkout -b name-of-your-bugfix-or-feature
```

   Now you can make your changes locally.

6. When you're done making changes, check that your changes pass the tests:

```
$ poetry run pytest
```

7. Install pre-commit

   $ poetry run pre-commit install

8. Commit your changes and push your branch to GitHub:

```
$ git add .
$ git commit -m "Your detailed description of your changes."
$ git push origin name-of-your-bugfix-or-feature
```

9. Submit a Pull Request through the GitHub website.

4.3 Pull Request Guidelines

Before you submit a Pull Request, check that it meets these guidelines:

1. The Pull Request should include tests.
2. If the Pull Request adds functionality, the docs should be updated.
3. The CI should pass.

CHAPTER

FIVE

RELEASE PROCESS

For maintainers only:

- Update `CHANGELOG.md` with the version changes
- Update `docs/` as required
- Update `README.md` as required
- Commit changes
- Run `bump2version <major|minor|patch>` to update the version number (pick one of the options)
  - Version number on `pyproject.toml` will be updated automatically
  - You can specify the `--new_version` flag in case you wish to manually set the newest version (if not provided, it will be done automatically based on the chosen option)
  - The command also creates a git tag
- Run `poetry build` to package the new version artifacts
- Run `poetry publish` to publish the packages

CHAPTER

SIX

AUTHORS

6.1 Development Lead

aurorae is maintained by Vinta Software. Lead maintainers:

- Mariane Pastor (Vinta Software) <mariane.pastor@vinta.com.br>
- Rebeca Sarai (Vinta Software) <rebeca@vinta.com.br>
- Thiago Paim (Vinta Software) <thiago.paim@vinta.com.br>
- Jaênia Sousa (Vinta Software) <jaenia@vinta.com.br>
- Marcos Felipe <@marcosflp>

6.2 Contributors

None yet. Why not be the first?

CHAPTER

SEVEN

CHANGELOG

All notable changes to this project will be documented in this file.

The format is based on Keep a Changelog, and this project adheres to Semantic Versioning.

7.1 [Unreleased]

...

7.2 [0.0.2] - 2021-08-20

7.2.1 Added

- Add log messages when generating files
- Move files to be saved on current folder instead of `generated_files` folder
- Review and improve names of CNAB types

7.3 [0.0.1] - 2021-08-20

7.3.1 First release