ANSEL Codecs Documentation

Release 0.1.1

David Haney

Jan 02, 2019

Contents:

CHAPTER 1

ANSEL Codecs

Codecs for reading/writing documents in the ANSEL character set.

- Free software: MIT license
- Documentation: https://python-ansel.readthedocs.io.

1.1 Features

- Adds support for new encodings ANSEL (ANSI/NISO Z39.47) and GEDCOM.
- Re-orders combining characters for consistency with the ANSEL specification.

1.2 Credits

This package was created with Cookiecutter and the audreyr/cookiecutter-pypackage project template.

CHAPTER 2

Installation

2.1 Stable release

To install ANSEL Codecs, run this command in your terminal:

```
$ pip install ansel
```

This is the preferred method to install ANSEL Codecs, as it will always install the most recent stable release.

If you don't have pip installed, this Python installation guide can guide you through the process.

2.2 From sources

The sources for ANSEL Codecs can be downloaded from the Github repo.

You can either clone the public repository:

```
$ git clone git://github.com/haney/python-ansel
```

Or download the tarball:

```
$ curl  -OL https://github.com/haney/python-ansel/tarball/master
```

Once you have a copy of the source, you can install it with:

```
$ python setup.py install
```

CHAPTER 3

Usage

To use ANSEL Codecs in a project

```
import ansel

ansel.register()
```

The `register` function registers each of the encodings supported by the ansel module. Once registered, they can be used with any of the functions of the `codecs` module or other functions that rely on codecs, for example:

```
with open(filename, "r", encodings="ansel") as fp:
    fp.read()
```

Will open the file `filename` for read with the "ansel" encoding.

3.1 Encodings

The following encodings are provided and registered with the `codecs` module:

Codec	Description
ansel	American National Standard for Extended Latin Alphabet Coded Character Set for Bibliographic Use (ANSEL).
gedcom	GEDCOM extensions to ANSEL.

3.2 Limitations

Pythons `open()` uses the `codecs.IncrementalEncoder` interface, however it doesn't invoke `codecs.IncrementalEncoder.encode()` with *final=True*. This prevents the final character written from being emitted to the stream. For example:

```
parts = ["P", "a", "\u030A", "l"]
with open("tmpfile", "w", encoding="ansel") as fp:
    for part in parts:
        fp.write(part)
```

will write the bytes:

0x50 P	0xEA	0x61 a

Note that the last character, 'l', does not appear in the byte sequence.

Related functions like `codecs.open()` have similar issues. They don't rely on the `codecs.IncrementalEncoder()`, and instead use the `codecs.encode()` function. Since each write is considered atomic, combining characters split across multiple write calls are not handled correctly:

```
with codecs.open("tmpfile", "w", encoding="ansel") as fp:
    for part in parts:
        fp.write(part)
```

will write the bytes:

0x50 P	0x61 a	0xEA	0x6C l

Note that while all of the bytes were written, the combining character follows the character it modifies. In ANSEL, the combining character should be before the character it modifies.

To avoid these issues, manually encoding and writing the parts is recommended. For example:

```
with codecs.open("tmpfile", "wb") as fp:
    for part in codecs.iterencode(parts, encoding="ansel"):
        fp.write(part)
```

will write the bytes:

0x50 P	0xEA	0x61 a	0x6C l

This version writes the correct byte sequence.

CHAPTER 4

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/haney/python-ansel/issues.

If you are reporting a bug, please include:

- Your operating system name and version.
- Any details about your local setup that might be helpful in troubleshooting.
- Detailed steps to reproduce the bug.

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.

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.

4.1.4 Write Documentation

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

4.1.5 Submit Feedback

The best way to send feedback is to file an issue at https://github.com/haney/python-ansel/issues.

If you are proposing a feature:

- Explain in detail how it would work.
- Keep the scope as narrow as possible, to make it easier to implement.
- Remember that this is a volunteer-driven project, and that contributions are welcome :)

4.2 Get Started!

Ready to contribute? Here's how to set up *ansel* for local development.

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

```
$ git clone git@github.com:your_name_here/python-ansel.git
```

3. Install your local copy into a virtualenv. Assuming you have virtualenvwrapper installed, this is how you set up your fork for local development:

```
$ mkvirtualenv ansel
$ cd ansel/
$ python setup.py develop
```

4. Create a branch for local development:

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

   Now you can make your changes locally.

5. When you're done making changes, check that your changes pass flake8 and the tests, including testing other Python versions with tox:

```
$ flake8 ansel tests
$ python setup.py test or py.test
$ tox
```

   To get flake8 and tox, just pip install them into your virtualenv.

6. 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
```

7. 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. Put your new functionality into a function with a docstring, and add the feature to the list in README.rst.
3. The pull request should work for Python 2.7, 3.4, 3.5 and 3.6, and for PyPy. Check https://travis-ci.org/haney/python-ansel/pull_requests and make sure that the tests pass for all supported Python versions.

4.4 Tips

To run a subset of tests:

```
$ py.test tests.test_ansel
```

4.5 Deploying

A reminder for the maintainers on how to deploy. Make sure all your changes are committed (including an entry in HISTORY.rst). Then run:

```
$ bumpversion patch # possible: major / minor / patch
$ git push
$ git push --tags
```

Travis will then deploy to PyPI if tests pass.

CHAPTER 5

Credits

5.1 Development Lead

- David Haney (@haney)

5.2 Contributors

None yet. Why not be the first?

CHAPTER 6

History

6.1 0.1.1 (2018-12-31)

- Fix packaging error that prevented subpackage from being included in distribution.

6.2 0.1.0 (2018-12-30)

- First release on PyPI.

CHAPTER 7

Indices and tables

- genindex
- modindex
- search