latexcodec Documentation

Release 2.0.0

Matthias C. M. Troffaes

Jan 26, 2021

Contents

Release 2.0.0

Date Jan 26, 2021

CHAPTER 1

Contents

1.1 Getting Started

1.1.1 Overview

A lexer and codec to work with LaTeX code in Python.

- Download: http://pypi.python.org/pypi/latexcodec/#downloads
- Documentation: http://latexcodec.readthedocs.org/
- Development: http://github.com/mcmtroffaes/latexcodec/

The codec provides a convenient way of going between text written in LaTeX and unicode. Since it is not a LaTeX compiler, it is more appropriate for short chunks of text, such as a paragraph or the values of a BibTeX entry, and it is not appropriate for a full LaTeX document. In particular, its behavior on the LaTeX commands that do not simply select characters is intended to allow the unicode representation to be understandable by a human reader, but is not canonical and may require hand tuning to produce the desired effect.

The encoder does a best effort to replace unicode characters outside of the range used as LaTeX input (ascii by default) with a LaTeX command that selects the character. More technically, the unicode code point is replaced by a LaTeX command that selects a glyph that reasonably represents the code point. Unicode characters with special uses in LaTeX are replaced by their LaTeX equivalents. For example,

original text	encoded LaTeX
¥	`\yen`
ü	`\"u`
`\N{NO-BREAK SPACE}`	`~`
`~`	`\textasciitilde`
`%`	`\%`
`#`	`\#`
`\textbf{x}`	`\textbf{x}`

The decoder does a best effort to replace LaTeX commands that select characters with the unicode for the character they are selecting. For example,

original LaTeX	decoded unicode
`\yen`	¥
`\"u`	ü
`~`	`\N{NO-BREAK SPACE}`
`\textasciitilde`	~
`\%`	%
`\#`	#
`\textbf{x}`	`\textbf {x}`
#	#

In addition, comments are dropped (including the final newline that marks the end of a comment), paragraphs are canonicalized into double newlines, and other newlines are left as is. Spacing after LaTeX commands is also canonicalized.

For example,

```
hi % bye
there\par world
\textbf     {awesome}
```

is decoded as

```
hi there

world
\textbf {awesome}
```

When decoding, LaTeX commands not directly selecting characters (for example, macros and formatting commands) are passed through unchanged. The same happens for LaTeX commands that select characters but are not yet recognized by the codec. Either case can result in a hybrid unicode string in which some characters are understood as literally the character and others as parts of unexpanded commands. Consequently, at times, backslashes will be left intact for denoting the start of a potentially unrecognized control sequence.

Given the numerous and changing packages providing such LaTeX commands, the codec will never be complete, and new translations of unrecognized unicode or unrecognized LaTeX symbols are always welcome.

1.1.2 Installation

Install the module with `pip install latexcodec`, or from source using `python setup.py install`.

1.1.3 Minimal Example

Simply import the `latexcodec` module to enable `"latex"` to be used as an encoding:

```
import latexcodec
text_latex = b"\\'el\\`eve"
assert text_latex.decode("latex") == u"élève"
text_unicode = u"ångström"
assert text_unicode.encode("latex") == b'\\aa ngstr\\"om'
```

There are also a `ulatex` encoding for text transforms. The simplest way to use this codec goes through the codecs module (as for all text transform codecs on Python):

```
import codecs
import latexcodec
text_latex = u"\\'el\\`eve"
assert codecs.decode(text_latex, "ulatex") == u"élève"
text_unicode = u"ångström"
assert codecs.encode(text_unicode, "ulatex") == u'\\aa ngstr\\"om'
```

By default, the LaTeX input is assumed to be ascii, as per standard LaTeX. However, you can also specify an extra codec as `latex+<encoding>` or `ulatex+<encoding>`, where `<encoding>` describes another encoding. In this case characters will be translated to and from that encoding whenever possible. The following code snippet demonstrates this behaviour:

```
import latexcodec
text_latex = b"\xfe"
assert text_latex.decode("latex+latin1") == u"þ"
assert text_latex.decode("latex+latin2") == u"ţ"
text_unicode = u"ţ"
assert text_unicode.encode("latex+latin1") == b'\\c t'  # ţ is not latin1
assert text_unicode.encode("latex+latin2") == b'\xfe'   # but it is latin2
```

When encoding using the `ulatex` codec, you have the option to pass through characters that cannot be encoded in the desired encoding, by using the `'keep'` error. This can be a useful fallback option if you want to encode as much as possible, whilst still retaining as much as possible of the original code when encoding fails. If instead you want to translate to LaTeX but keep as much of the unicode as possible, use the `ulatex+utf8` codec, which should never fail.

```
import codecs
import latexcodec
text_unicode = u''  # \u2328 = keyboard symbol, currently not translated
try:
    # raises a value error as \u2328 cannot be encoded into latex
    codecs.encode(text_unicode, "ulatex+ascii")
except ValueError:
    pass
assert codecs.encode(text_unicode, "ulatex+ascii", "keep") == u''
assert codecs.encode(text_unicode, "ulatex+utf8") == u''
```

1.1.4 Limitations

- Not all unicode characters are registered. If you find any missing, please report them on the tracker: https://github.com/mcmtroffaes/latexcodec/issues
- Unicode combining characters are currently not handled.
- By design, the codec never removes curly brackets. This is because it is very hard to guess whether brackets are part of a command or not (this would require a full latex parser). Moreover, bibtex uses curly brackets as a guard against case conversion, in which case automatic removal of curly brackets may not be desired at all, even if they are not part of a command. Also see: http://stackoverflow.com/a/19754245/2863746

1.2 API

1.2.1 LaTeX Codec

The *latexcodec.codec* module contains all classes and functions for LaTeX code translation. For practical use, you should only ever need to import the `latexcodec` module, which will automatically register the codec so it can be used by `str.encode()`, `str.decode()`, and any of the functions defined in the `codecs` module such as `codecs.open()` and so on. The other functions and classes are exposed in case someone would want to extend them.

`latexcodec.codec.`**`register`**`()`
Register the *find_latex()* codec search function.

See also:

`codecs.register()`

`latexcodec.codec.`**`find_latex`**`(`*encoding*`)`
Return a `codecs.CodecInfo` instance for the requested LaTeX *encoding*, which must be equal to `latex`, or to `latex+<encoding>` where `<encoding>` describes another encoding.

`class` `latexcodec.codec.`**`LatexIncrementalEncoder`**`(`*errors='strict'*`)`
Bases: *latexcodec.lexer.LatexIncrementalEncoder*

Translating incremental encoder for latex. Maintains a state to determine whether control spaces etc. need to be inserted.

`get_latex_chars``(`*unicode_*, *final=False*`)`
Encode every character. Override to process the unicode in some other way (for example, for character translation).

`get_space_bytes``(`*bytes_*`)`
Inserts space bytes in space eating mode.

`reset``()`
Reset state.

`class` `latexcodec.codec.`**`LatexIncrementalDecoder`**`(`*errors='strict'*`)`
Bases: *latexcodec.lexer.LatexIncrementalDecoder*

Translating incremental decoder for LaTeX.

`get_unicode_tokens``(`*chars*, *final=False*`)`
Decode every token. Override to process the tokens in some other way (for example, for token translation).

`getstate``()`
Get state.

`reset``()`
Reset state.

`setstate``(`*state*`)`
Set state. The *state* must correspond to the return value of a previous *getstate()* call.

`class` `latexcodec.codec.`**`LatexCodec`**
Bases: `codecs.Codec`

`decode``(`*bytes_*, *errors='strict'*`)`
Convert LaTeX bytes to unicode string.

`encode``(`*unicode_*, *errors='strict'*`)`
Convert unicode string to LaTeX bytes.

class latexcodec.codec.**LatexUnicodeTable**(*lexer*)
Tabulates a translation between LaTeX and unicode.

register(*unicode_text*, *latex_text*, *mode='text'*, *package=None*, *decode=True*, *encode=True*)
Register a correspondence between *unicode_text* and *latex_text*.

Parameters

- **unicode_text** (*str*) – A unicode character.
- **latex_text** (*str*) – Its corresponding LaTeX translation.
- **mode** (*str*) – LaTeX mode in which the translation applies (`'text'` or `'math'`).
- **package** (*str*) – LaTeX package requirements (currently ignored).
- **decode** (*bool*) – Whether this translation applies to decoding (default: `True`).
- **encode** (*bool*) – Whether this translation applies to encoding (default: `True`).

register_all()
Register all symbols and their LaTeX equivalents (called by constructor).

1.2.2 LaTeX Lexer

This module contains all classes for lexing LaTeX code, as well as general purpose base classes for incremental LaTeX decoders and encoders, which could be useful in case you are writing your own custom LaTeX codec.

class latexcodec.lexer.**Token**(*name*, *text*)
Token(name, text)

class latexcodec.lexer.**LatexLexer**(*errors='strict'*)
Bases: latexcodec.lexer.RegexpLexer

A very simple lexer for tex/latex.

class latexcodec.lexer.**LatexIncrementalLexer**(*errors='strict'*)
Bases: *latexcodec.lexer.LatexLexer*

A very simple incremental lexer for tex/latex code. Roughly follows the state machine described in Tex By Topic, Chapter 2.

The generated tokens satisfy:

- no newline characters: paragraphs are separated by 'par'
- spaces following control tokens are compressed

get_tokens(*chars*, *final=False*)
Yield tokens while maintaining a state. Also skip whitespace after control words and (some) control symbols. Replaces newlines by spaces and par commands depending on the context.

getstate()
Get state.

reset()
Reset state.

setstate(*state*)
Set state. The *state* must correspond to the return value of a previous *getstate()* call.

class latexcodec.lexer.**LatexIncrementalDecoder**(*errors='strict'*)
Bases: *latexcodec.lexer.LatexIncrementalLexer*

Simple incremental decoder. Transforms lexed LaTeX tokens into unicode.

To customize decoding, subclass and override *get_unicode_tokens()*.

decode(*bytes_*, *final=False*)
Decode LaTeX *bytes_* into a unicode string.

This implementation calls *get_unicode_tokens()* and joins the resulting unicode strings together.

decode_token(*token*)
Returns the decoded token text.

Note: Control words get an extra space added at the back to make sure separation from the next token, so that decoded token sequences can be joined together.

For example, the tokens `u'\hello'` and `u'world'` will correctly result in `u'\hello world'` (remember that LaTeX eats space following control words). If no space were added, this would wrongfully result in `u'\helloworld'`.

---

get_unicode_tokens(*chars*, *final=False*)
Decode every token. Override to process the tokens in some other way (for example, for token translation).

class latexcodec.lexer.**LatexIncrementalEncoder**(*errors='strict'*)
Bases: codecs.IncrementalEncoder

Simple incremental encoder for LaTeX. Transforms unicode into bytes.

To customize decoding, subclass and override `get_latex_bytes()`.

encode(*unicode_*, *final=False*)
Encode the *unicode_* string into LaTeX bytes.

This implementation calls *get_latex_chars()* and joins the resulting bytes together.

flush_unicode_tokens()
Flush the buffer.

get_latex_chars(*unicode_*, *final=False*)
Encode every character. Override to process the unicode in some other way (for example, for character translation).

get_unicode_tokens(*unicode_*, *final=False*)
Split unicode into tokens so that every token starts with a non-combining character.

getstate()
Get state.

reset()
Reset state.

setstate(*state*)
Set state. The *state* must correspond to the return value of a previous *getstate()* call.

1.3 Changes

1.3.1 2.0.0 (14 January 2020)

- Lexer now processes unicode directly, to fix various issues with multibyte encodings. This also simplifies the implementation. Many thanks to davidweichiang for reporting and implementing.

- New detailed description of the package for the readme, to clarify the behaviour and design choices. Many thanks to tschantzmc for contributing this description (see issue #70).
- Minor fix in decoding of LaTeX comments (see issue #72).
- Support Python 3.9 (see issue #75).

1.3.2 1.0.7 (3 May 2019)

- More symbols (THIN SPACE, various accented characters).
- Fix lexer issue with multibyte encodings (thanks to davidweichiang for reporting).

1.3.3 1.0.6 (18 January 2018)

- More symbols (EM SPACE, MINUS SIGN, GREEK PHI SYMBOL, HYPHEN, alternate encodings of Swedish å and Å).

1.3.4 1.0.5 (16 June 2017)

- More maths symbols (naturals, reals, . . .).
- Fix lower case z with accents (reported by AndrewSwann, see issue #51).

1.3.5 1.0.4 (21 September 2016)

- Fix encoding and decoding of percent sign (reported by jgosmann, see issue #48).

1.3.6 1.0.3 (26 March 2016)

- New `'keep'` error for the ulatex encoder to keep unicode characters that cannot be translated (contributed by xuhdev, see pull requestion #45).

1.3.7 1.0.2 (1 March 2016)

- New `ulatex` codec which works as a text transform on unicode strings.
- Fix spacing when translating math (see issue #29, reported by beltiste).
- Performance improvements in latex to unicode translation.
- Support old-style math mode (see pull request #40, contributed by xuhdev).
- Treat tab character as a space character (see discussion in issue #40, raised by xuhdev).

1.3.8 1.0.1 (24 September 2014)

- br"\par" is now decoded using two newlines (see issue #26, reported by Jorrit Wronski).
- Fix encoding and decoding of the ogonek (see issue #24, reported by beltiste).

1.3.9 1.0.0 (5 August 2014)

- Add Python 3.4 support.
- Fix "DZ" decoding (see issue #21, reported and fixed by Philipp Spitzer).

1.3.10 0.3.2 (17 April 2014)

- Fix underscore "_" encoding (see issue #17, reported and fixed by Michael Radziej).

1.3.11 0.3.1 (5 February 2014)

- Drop Python 3.2 support.
- Drop 2to3 and instead use six to support both Python 2 and 3 from a single code base.
- Fix control space "\ " decoding.
- Fix LaTeX encoding of number sign "#" and other special ascii characters (see issues #11 and #13, reported by beltiste).

1.3.12 0.3.0 (19 August 2013)

- Copied lexer and codec from sphinxcontrib-bibtex.
- Initial usage and API documentation.
- Some small bugs fixed.

1.3.13 0.2 (28 September 2012)

- Adding additional codec with brackets around special characters.

1.3.14 0.1 (26 May 2012)

- Initial release.

1.4 Authors

Main authors:

- David Eppstein
  - wrote the original LaTeX codec as a recipe on ActiveState http://code.activestate.com/recipes/252124-latex-codec/
- Peter Tröger
  - wrote the original latexcodec package, which contained a simple but very effective LaTeX encoder
- Matthias Troffaes (matthias.troffaes@gmail.com)
  - wrote the lexer

  - integrated codec with the lexer for a simpler and more robust design
  - various bugfixes

Contributors:

- Michael Radziej
- Philipp Spitzer

1.5 License

Remark

Versions 0.1 and 0.2 of the latexcodec package were written by Peter Tröger, and were released under the Academic Free License 3.0. The current version of the latexcodec package shares no code with those earlier versions.

CHAPTER 2

Indices and tables

- genindex
- modindex
- search

Python Module Index

l

Index

S

T