texttables Documentation

Release 1.0.1

Taylor C. Richberger

Sep 27, 2017

Contents

This is a simple python module for reading and writing ASCII text tables. It attempts to have an interface as similar to Python's official `csv` module as possible. It supports fixed-size tables (where column sizes are pre-decided) for reading and writing (including with a dictionary). It supports dynamic-sized tables (where each column's width is deduced to be the largest element in that column) for writing only, including dict writing.

There are less obvious uses to this module, such as being able to use a sort of TSV that is width-delimited rather than character-delimited.

There are a few limitations to this module. The most obvious are that it enforces some specific rules to the tables. Cells may not span multiple rows or columns (and the table therefore must represent a strict integral grid), meaning that this can only parse a subset of allowed RST tables. Corners must all be identical, including in the header and all borders (if present). This module only parses text tables. It will not assist in parsing HTML, LaTeX, or any other kind of markup.

There is a small unit test suite that attempts to catch obvious issues. Pull requests to any of this module are welcome, as long as the license remains the same.

The sources are available in the GitHub Repository.

CHAPTER 1

Fixed Readers

texttables.fixed.reader

class texttables.fixed.**reader**(*file*, *widths*, *dialect=None*, *fieldnames=None*, ***fmtparams*)

Fixed-table table reader, reading tables with predefined column-sizes. The *texttables.Dialect* class is used to configure how this reads tables. This is an iterable, returning rows from the table as tuples.

Iteration can raise a *texttables.ValidationError* if an invalid table is read.

Parameters

- **file** – An iterable object, returning a line with each iteration.
- **widths** – An iterable of widths, containing the field sizes of the table. Each width may be prefixed with <, >, =, or ^, for alignment through the Python format specification, though these prefixes will be ignored if they are present.
- **dialect** – A dialect class or object used to define aspects of the table. The stored dialect is always an instance of *texttables.Dialect*, not necessarily the passed-in object. All the attributes of Dialect are grabbed from this object using getattr.
- **fieldnames** – An iterable specifying the field names. If this is absent, field names are pulled from the table. This will change how the table is read. If this parameter is present, the table may not have a header. If this parameter is absent, the table must have a header. Either way, the field names of the table must be delivered to this class in one way, and exactly only one way.
- **fmtparams** – parameters to override the parameters in dialect.

dialect

The *texttables.Dialect* constructed from the passed-in dialect. This is always unique, and is not the same object that is passed in. Assigning to this will also likewise construct a new *texttables.Dialect*, not simply assign the attribute.

fieldnames

The table's fieldnames as a tuple. This will invoke a read on the file if this method has not been called and this object hasn't yet been iterated upon.

Raises *texttables.ValidationError* – if the table does not properly match the dialect

file
The file object that was passed in to the constructor. It is not safe to change this object until you are finished using the class

widths
The widths that were passed into the constructor, as a tuple, with any alignments stripped.

texttables.fixed.DictReader

class texttables.fixed.**DictReader**(*file*, *widths*, *dialect=None*, *fieldnames=None*, ***fmtparams*)
Fixed-table table dictionary reader, reading tables with predefined column-sizes. The *texttables.Dialect* class is used to configure how this reads tables. Tables are read one row at a time. This is a simple convenience frontend to *texttables.fixed.reader*. This is an iterable, returning rows from the table as dictionaries.

All the passed in construction parameters are passed to the *texttables.fixed.reader* constructor literally. All properties also align directly as well.

dialect

fieldnames

file

widths

CHAPTER 2

Fixed Writers

texttables.fixed.writer

class texttables.fixed.**writer**(*file*, *widths*, *dialect=None*, ***fmtparams*)

Fixed-table document writer, writing tables with predefined column-sizes. The *texttables.Dialect* class is used to configure how this writes tables. This works as a context manager, in which case writetop() and writebottom() will be called automatically.

Parameters

- **file** – A writable file object with a write method
- **widths** – An iterable of widths, containing the field sizes of the table. Each width may be prefixed with <, >, =, or ^, for alignment through the Python format specification.
- **dialect** – A dialect class or object used to define aspects of the table. The stored dialect is always an instance of *texttables.Dialect*, not necessarily the passed-in object. All the attributes of Dialect are grabbed from this object using getattr.
- **fmtparams** – parameters to override the parameters in dialect.

dialect

The *texttables.Dialect* constructed from the passed-in dialect. This is always unique, and is not the same object that is passed in. Assigning to this will also likewise construct a new *texttables.Dialect*, not simply assign the attribute.

file

The file object that was passed in to the constructor. It is not safe to change this object until you are finished using the class

widths

The widths that were passed into the constructor, as a tuple.

writebottom()

Write the bottom of the table out to file().

writeheader(*row*)
Write the header out to `file()`.

Parameters **row** – An iterable representing the row to write as a header

writerow(*row*)
Write a single row out to `file()`, respecting any delimiters and header separators necessary.

Parameters **row** – An iterable representing the row to write

writerows(*rows*)
Write a multiple rows out to `file()`, respecting any delimiters and header separators necessary.

Parameters **rows** – An iterable of iterables representing the rows to write

writetop()
Write the top of the table out to `file()`.

texttables.fixed.DictWriter

class texttables.fixed.**DictWriter**(*file*, *fieldnames*, *widths*, *dialect=None*, ***fmtparams*)
Fixed-table document writer, writing tables with predefined column-sizes and names through dictionary rows passed in.

The *texttables.Dialect* class is used to configure how this writes tables. This is a simple convenience frontend to *texttables.fixed.writer*. This works as a context manager, in which case `writetop()` and `writebottom()` will be called automatically.

All the passed in construction parameters are passed to the *texttables.fixed.writer* constructor literally. All properties and most methods also align directly as well.

dialect

fieldnames

file

widths

writebottom()

writeheader()
Write the header based on `fieldnames()`.

writerow(*row*)
Write a single row out to `file()`, respecting any delimiters and header separators necessary.

Parameters **row** – A dictionary representing the row to write

writerows(*rows*)
Write multiple rows out to `file()`, respecting any delimiters and header separators necessary.

Parameters **row** – An iterable of dictionaries representing the rows to write

writetop()

CHAPTER 3

Dynamic Writers

texttables.dynamic.writer

class texttables.dynamic.**writer**(*file*, *alignments=None*, *dialect=None*, ***fmtparams*)

Dynamic-table document writer, writing tables with computed column-sizes. The *texttables.Dialect* class is used to configure how this writes tables. This works as a context manager, in which case `finish()` will be called automatically. This class does not actually write anything out until `finish()` is called (or the context manager is exited) because it needs the information from all rows before it knows how wide to make all the columns.

Parameters

- **file** – A writable file object with a `write` method
- **alignments** – An iterable of alignments. Each alignment may be <, >, =, or ^, for alignment through the Python format specification.
- **dialect** – A dialect class or object used to define aspects of the table. The stored dialect is always an instance of *texttables.Dialect*, not necessarily the passed-in object. All the attributes of Dialect are grabbed from this object using getattr.
- **fmtparams** – parameters to override the parameters in `dialect`.

dialect

The passed-in dialect. This does not behave like the fixed dialects, because it does not actually construct a *texttables.Dialect* until `finish()` is called.

file

The file object that was passed in to the constructor. It is not safe to change this object until you are finished using the class

finish()

Write the top, the bottom, the header (if present), and all rows out with proper delimitation to `file()`, respecting the dialect

rows

Get or set the total rows. This will override all rows passed in with `writerow()` and `writerows()`

writeheader(*header*)
Set the header to be written out

writerow(*row*)
Add a row to the row set to be written out. This does not write anything, and is only named as such for uniformity

Parameters **row** – an iterable representing a row to write

writerows(*rows*)
Add rows to the row set to be written out. This does not write anything, and is only named as such for uniformity

Parameters **rows** – An iterable of iterables representing the rows to write

texttables.dynamic.DictWriter

class texttables.dynamic.**DictWriter**(*file*, *fieldnames*, *alignments=None*, *dialect=None*, ***fmtparams*)
Dynamic-table document writer, writing tables with predefined column-sizes and names through dictionary rows passed in.

The *texttables.Dialect* class is used to configure how this writes tables. This is a simple convenience frontend to *texttables.dynamic.writer*. This works as a context manager, in which case `finish()` will be called automatically.

All the passed in construction parameters are passed to the *texttables.dynamic.writer* constructor literally. All properties and most methods also align directly as well.

dialect

fieldnames

file

writeheader()
Set the header based on `fieldnames()`.

writerow(*row*)
Write a row based on `fieldnames()`.

Parameters **row** – A dictionary representing a row.

writerows(*rows*)
Write rows based on `fieldnames()`.

Parameters **row** – An iterable of dictionaries representing rows.

CHAPTER 4

texttables.Dialect

class texttables.**Dialect**

Class that is mostly subclassed for use in tables. Some attributes might only be used for either a dynamic or fixed table, but not both. Likewise, some attributes might only be used for a reader or writer. This can be instantiated and have the attributes changed instead of subclassing, for one-offs, but subclassing is usually clearer.

bottom_border = None

Border character for non-corners on the bottom side of the bottom cells. None to disable

cell_delimiter = u' '

Delimiter character separating cells from one another. Must exist.

corner_border = u'+'

Border character for corners on each border and on the row and header delimiters. Required when the borders or delimiters are specified.

header_delimiter = None

Delimiter character separating header from rows. None to disable

left_border = None

Border character for non-corners on the left side of each row. None to disable

lineterminator = u'\n'

Line terminator. Used only for writing tables, and ignored on reading

right_border = None

Border character for non-corners on the right side of each row. None to disable

row_delimiter = None

Delimiter character separating rows from one another. None to disable

strict = True

Whether to raise an exception on read errors, such as borders appearing in the wrong order or missing borders.

strip = True

Whether to strip fields on reads. This is usually desired, especially for DictReader types.

top_border = None
Border character for non-corners on the top side of the top cells. None to disable

CHAPTER 5

texttables.ValidationError

class texttables.**ValidationError**
This is raised if *texttables.Dialect.strict* is True and an invalid table is read.

CHAPTER 6

Examples

texttables.fixed.writer

```
>>> from texttables import Dialect
>>> from texttables.fixed import writer
>>> from sys import stdout
>>> with writer(stdout, [10, 10, 10]) as w:
...     w.writeheader(('header 1', 'header 2', 'header 3'))
...     w.writerow(('data 1', 'data 2', 'data 3'))
...     w.writerow(('data 4', 'data 5', 'data 6'))
...
header 1   header 2   header 3
data 1     data 2     data 3
data 4     data 5     data 6
```

texttables.Dialect

```
>>> from texttables import Dialect
>>> from texttables.fixed import writer
>>> from sys import stdout
>>> class dialect(Dialect):
...     header_delimiter = '='
...     row_delimiter = '-'
...     top_border = '#'
...     bottom_border = '_'
...     left_border = '|'
...     cell_delimiter = '|'
...     right_border = '|'
...     corner_border = '+'
...
>>> with writer(stdout, [10, 10, 10], dialect=dialect) as w:
...     w.writeheader(('header 1', 'header 2', 'header 3'))
```

```
...     w.writerow(('data 1', 'data 2', 'data 3'))
...     w.writerow(('data 4', 'data 5', 'data 6'))
...
+##########+##########+##########+
|header 1  |header 2  |header 3  |
+==========+==========+==========+
|data 1    |data 2    |data 3    |
+----------+----------+----------+
|data 4    |data 5    |data 6    |
+__________+__________+__________+
```

texttables.fixed.DictWriter

```
>>> from texttables import Dialect
>>> from texttables.fixed import DictWriter
>>> from sys import stdout
>>> class dialect(Dialect):
...     header_delimiter = '='
...     row_delimiter = '-'
...     top_border = '#'
...     bottom_border = '_'
...     left_border = '|'
...     cell_delimiter = '|'
...     right_border = '|'
...     corner_border = '+'
...
>>> with DictWriter(stdout, ['foo', 'bar', 'baz'], [10, '>10', '^10'],␣
↪dialect=dialect) as w:
...     w.writeheader()
...     w.writerow({'foo': 'data 1', 'bar': 'data 2', 'baz': 'data 3'})
...     w.writerow({'foo': 'data 4', 'bar': 'data 5', 'baz': 'data 6'})
...
+##########+##########+##########+
|foo       |       bar|   baz    |
+==========+==========+==========+
|data 1    |    data 2|  data 3  |
+----------+----------+----------+
|data 4    |    data 5|  data 6  |
+__________+__________+__________+
```

texttables.fixed.reader

```
>>> from texttables import Dialect
>>> from texttables.fixed import reader
>>> from sys import stdout
>>> class dialect(Dialect):
...     header_delimiter = '='
...     row_delimiter = '-'
...     top_border = '#'
...     bottom_border = '_'
...     left_border = '|'
...     cell_delimiter = '|'
```

```
...     right_border = '|'
...     corner_border = '+'
...
>>> data = (
...     '+##########+##########+##########+\n'
...     '|header 1  |  header 2| header 3 |\n'
...     '+==========+==========+==========+\n'
...     '|data 1    |    data 2|  data 3  |\n'
...     '+----------+----------+----------+\n'
...     '|data 4    |    data 5|  data 6  |\n'
...     '+__________+__________+__________+\n'
...     )
>>> r = reader(data.splitlines(), [10, 10, 10], dialect=dialect)
>>> rows = [row for row in r]
>>> r.fieldnames
('header 1', 'header 2', 'header 3')
>>> rows
[('data 1', 'data 2', 'data 3'), ('data 4', 'data 5', 'data 6')]
```

texttables.fixed.DictReader

```
>>> from texttables import Dialect
>>> from texttables.fixed import DictReader
>>> from sys import stdout
>>> class dialect(Dialect):
...     header_delimiter = '='
...     row_delimiter = '-'
...     top_border = '#'
...     bottom_border = '_'
...     left_border = '|'
...     cell_delimiter = '|'
...     right_border = '|'
...     corner_border = '+'
...
>>> data = (
...     '+##########+##########+##########+\n'
...     '|header 1  |  header 2| header 3 |\n'
...     '+==========+==========+==========+\n'
...     '|data 1    |    data 2|  data 3  |\n'
...     '+----------+----------+----------+\n'
...     '|data 4    |    data 5|  data 6  |\n'
...     '+__________+__________+__________+\n'
...     )
>>> r = DictReader(data.splitlines(), [10, 10, 10], dialect=dialect)
>>> rows = [row for row in r]
>>> rows
[{'header 3': 'data 3', 'header 2': 'data 2', 'header 1': 'data 1'}, {'header 3':
↪'data 6', 'header 2': 'data 5', 'header 1': '
data 4'}]
```

texttables.ValidationError

```
>>> from texttables import Dialect
>>> from texttables.fixed import DictReader
>>> from sys import stdout
>>> class dialect(Dialect):
...     header_delimiter = '='
...     row_delimiter = '-'
...     top_border = '#'
...     bottom_border = '_'
...     left_border = '|'
...     cell_delimiter = '|'
...     right_border = '|'
...     corner_border = '+'
...
>>> data = (
...     '|header 1  |  header 2| header 3 |\n'
...     '+==========+==========+==========+\n'
...     '|data 1    |    data 2|  data 3  |\n'
...     '+----------+----------+----------+\n'
...     '|data 4    |    data 5|  data 6  |\n'
...     '+__________+__________+__________+\n'
...     )
>>> r = DictReader(data.splitlines(), [10, 10, 10], dialect=dialect)
>>> rows = [row for row in r]
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
  File "<stdin>", line 1, in <listcomp>
  File "/home/taylor/Projects/texttables/texttables/fixed/_reader.py", line 273, in __
↪next__
    row = next(self._iter)
  File "/home/taylor/Projects/texttables/texttables/fixed/_reader.py", line 205, in __
↪next__
    fieldnames = self.fieldnames
  File "/home/taylor/Projects/texttables/texttables/fixed/_reader.py", line 135, in␣
↪fieldnames
    raise ValidationError('The first line of the table did not match what the top of␣
↪the table should be')
texttables.errors.ValidationError: The first line of the table did not match what the␣
↪top of the table should be
```

texttables.dynamic.writer

```
>>> from texttables import Dialect
>>> from texttables.dynamic import writer
>>> from sys import stdout
>>> class dialect(Dialect):
...     header_delimiter = '='
...     row_delimiter = '-'
...     top_border = '#'
...     bottom_border = '_'
...     left_border = '|'
...     cell_delimiter = '|'
...     right_border = '|'
...     corner_border = '+'
```

```
...
>>> with writer(stdout, ['', '>', '^'], dialect=dialect) as w:
...     w.writeheader(('header 1', 'header 2', 'header 3'))
...     w.writerows([
...         ('data 1', 'data 2', 'data 3'),
...         ('data 4', 'data 5', 'data 6')])
...
+########+########+########+
|header 1|header 2|header 3|
+========+========+========+
|data 1  |  data 2| data 3 |
+--------+--------+--------+
|data 4  |  data 5| data 6 |
+________+________+________+
```

texttables.dynamic.DictWriter

```
>>> from texttables import Dialect
>>> from texttables.dynamic import DictWriter
>>> from sys import stdout
>>> class dialect(Dialect):
...     header_delimiter = '='
...     corner_border = ' '
...
>>> with DictWriter(stdout, ['foo', 'bar', 'baz'], dialect=dialect) as w:
...     w.writeheader()
...     w.writerows([
...         {'foo': 'data 1', 'bar': 'data 2', 'baz': 'data 3'},
...         {'foo': 'data 4', 'bar': 'data 5', 'baz': 'data 6'}])
...
foo    bar    baz
====== ====== ======
data 1 data 2 data 3
data 4 data 5 data 6
```

RST tables

```
>>> from texttables import Dialect
>>> from texttables.fixed import DictReader
>>> from sys import stdout
>>> data = '''
... +------------------------+------------+----------+----------+
... | Header row, column 1   | Header 2   | Header 3 | Header 4 |
... +========================+============+==========+==========+
... | body row 1, column 1   | column 2   | column 3 | column 4 |
... +------------------------+------------+----------+----------+
... | body row 2             | ...        | ...      |          |
... +------------------------+------------+----------+----------+
... '''.strip()
>>> class dialect(Dialect):
...     header_delimiter = '='
...     corner_border = '+'
```

```
...     top_border = '-'
...     bottom_border = '-'
...     left_border = '|'
...     right_border = '|'
...     cell_delimiter = '|'
...     row_delimiter = '-'
...
>>> [row for row in DictReader(data.splitlines(), [24, 12, 10, 10], dialect=dialect)]
[{'Header 4': 'column 4', 'Header 2': 'column 2', 'Header row, column 1': 'body row 1,
↪ column 1', 'Header 3': 'column 3'}, {'Header 4': '', 'Header 2': '...', 'Header␣
↪row, column 1': 'body row 2', 'Header 3': '...'}]
```

```
>>> from texttables import Dialect
>>> from texttables.fixed import DictReader
>>> from sys import stdout
>>> data = '''
... ===== ===== =======
... A     B     A and B
... ===== ===== =======
... False False False
... True  False False
... False True  False
... True  True  True
... ===== ===== =======
... '''.strip()
>>> class dialect(Dialect):
...     header_delimiter = '='
...     corner_border = ' '
...     top_border = '='
...     bottom_border = '='
...     cell_delimiter = ' '
...
>>> [row for row in DictReader(data.splitlines(), [5, 5, 7], dialect=dialect)]
[{'A and B': 'False', 'A': 'False', 'B': 'False'}, {'A and B': 'False', 'A': 'True',
↪'B': 'False'}, {'A and B': 'False', 'A': 'False', 'B': 'True'}, {'A and B': 'True',
↪'A': 'True', 'B': 'True'}]
```

CHAPTER 7

Indices and tables

- genindex
- search

CHAPTER 8

License

This module is released under the MIT license.

Index