Rigidity Documentation

Release 1.3.0

Austin Hartzheim

January 03, 2016

Contents

Rigidity is a simple wrapper to Python's built-in csv module that allows for validation and correction of data being read/written to/from CSV files.

With Rigidity, you can easily construct validation and correction rulesets to be applied automatically while preserving the csv interface. In other words, you can easily upgrade old code to better adhere to new output styles, or allow new code to better parse old files.

CHAPTER 1

Installing

1.1 Installing via PyPI

Rigidity is listed on the Python Package Index. So, you may install it via pip:

```
pip install rigidity
```

Note that Rigidity only supports Python 3, so you may need to modify your pip command if your default Python version differs.

1.2 Installing from Source

If you have downloaded one of the source tarballs via Github, you may install it as follows:

```
tar -xzf rigidity-1.3.0.tar.gz
cd rigidity-1.3.0
sudo python3 setup.py install
```

1.3 Installing from Git

If you want to install the development version, you may clone our git repository:

```
git clone https://github.com/austinhartzheim/rigidity.git
cd rigidity
sudo python3 setup.py install1
```

CHAPTER 2

Examples

This page includes examples of how to use Rigidity in your own projects.

2.1 Correcting Capitalization

Some spreadsheet providers insist on capitalizing all data. But, readability can be greatly enhanced by capitalizing words correctly.

Take the following CSV file as an example:

```
TITLE,AUTHOR
BRAVE NEW WORLD,ALDOUS HUXLEY
NINETEEN EIGHTY-FOUR,GEORGE ORWELL
```

It would be much more readable in the following form, and could even be included directly on a public-facing website:

```
Title,Author
Brave New World,Aldous Huxley
Nineteen Eighty-Four,George Orwell
```

Rigidity's *CapitalizeWords* rule allows for selective capitalization of certain letters. By default, it capitalizes the characters following whitespace. But, we need to capitalize words following hyphens as well (in the case of Nineteen Eighty-Four). Here is how we do it:

```
import csv
import rigidity

reader = csv.reader(open('data.csv'))

rules = [
    [rigidity.rules.Lower(),  # Convert to lower-case first
     rigidity.rules.CapitalizeWords(' -')],  # Selectively capitalize
    [rigidity.rules.Lower(),  # Do the same for the author
     rigidity.rules.CapitalizeWords(' -')]
]
r = rigidity.Rigidity(reader, rules)

for row in r:
    print(', '.join(row))
```

The *CapitalizeWords* rule only performs selective capitalization. So, we need to use the *Lower* rule to convert the entire string to lower-case first. We also tell the rule to capitalize all letters immediately following a space character or a hyphen, which allows us to correctly capitalize "Nineteen Eighty-Four."

2.2 UPC Validation

The following example demonstrates how to validate that a UPC-A code is correct by using the check digit. An additional test is also performed to ensure that the UPC is unique (which prevents accidental duplicates of what should be a unique identifier):

```
import rigidity

rules = [
    [rigidity.rules.UpcA(strict=True),
     rigidity.rules.Unique()]
]
r = rigidity.Rigidity(reader, rules)

for row in r:
    print(row[0])
```

This example assumes that there is only one column in the CSV file - the column with the UPC code.

Activating *strict* on the UpcA rule causes the check bit of the UPC to be validated. If the digit is not valid, an error is raised. This can be deactivated to prevent check digit verification.

CHAPTER 3

Creating Rules

This page details how you can create your own rules for use in Rigidity.

3.1 A Simple Example

Rigidty rules are all contained in classes subclassing `Rule`. The simplest example has only an *apply()* method that validates or modifies data.

For example, a simple rule to check integers might look like this:

```
class Integer(rigidity.rules.Rule):
    def apply(self, value):
        return int(value)
```

When this rule is used, it attempts to convert the data passed in the *value* parameter to an integer. If it is successful, it returns the integer representation of *value*. If it fails, the *ValueError* from the cast propagates, preventing the invalid data from entering or exiting the program.

As a side effect, because the integer representation is returned, all future checks will then be operating on an integer value, regardless of the original type of *value*. This can be useful to automatically convert data as it enters the program rather than having to handle the casting logic later.

3.2 Dropping Rows

It is not always desirable to raise an error when invalid data is discovered. Sometimes the appropriate action is to ignore the offending row. Rigidity rules can cause a row to be ignored by raising the *DropRow* exception.

The following code can be used to verify inventory data, preventing the inclusion of any products that are out of stock:

```
class Inventory(rigidity.rules.Rule):
    def apply(self, value):
        if isinstance(value, str):
            value = int(str)
        if not isinstance(value, int):
            raise ValueError('Inventory was not an integer value.')

        if value < 1:
            raise rigidity.errors.DropRow()
        return value
```

If we use this rule to validate this CSV file:

```
Product,Inventory
T-Shirt,12
Pants,4
Shorts,0
Shoes,-1
Gloves,3
```

We will get the following list of items that are in stock at the store:

```
Product,Inventory
T-Shirt,12
Pants,4
Gloves,3
```

Additionally, if any invalid data is located in the inventory column, an error will be raised to prevent other data from entering the CSV file.

3.3 Bidirectional Validation

Sometimes it is necessary to validate data differently depending on whether it is being read or written. This is why the *Rule* class supports both the *read()* and *write()* methods. Implementing these methods in your rules can allow for greater flexibility of rulesets because the same rules can be used for both reading and writing data.

An example use of this functionality is the built-in *Bytes* rule. This rule assumes that the data being read is raw binary data that is best represented as a Python *bytes* object:

```
class Bytes(Rule):
    '''
    When reading data, encode it as a bytes object using the given
    encoding. When writing data, decode it using the given encoding.
    '''

    def __init__(self, encoding='utf8'):
        self.encoding = encoding

    def read(self, value):
        return value.encode(self.encoding)

    def write(self, value):
        return value.decode(self.encoding)
```

When the data is read from a CSV file, the `read()` method is called, which encodes the data using the selected encoding type and returns it. When it is time to write the data back into a CSV file, the `write()` method is called to decode the data using the specified encoding scheme and return the value.

This rule could not be implemented as a unidirectional rule because the *csv* module would not know how to decode the bytes object.

CHAPTER 4

Code Documentation

4.1 Errors

This submodule contains exception classes that are used by Rigidity to handle different actions from the rule classes.

exception rigidity.errors.**DropRow**
Bases: rigidity.errors.RigidityException

When a rule raises this error, the row that is being processed is dropped from the output.

4.2 Rigidity

This module contains the wrapper class that can be used to adapt your CSV parsing code to use rigidity. Rigidity is a simple wrapper to the built-in csv module that allows for validation and correction of data being read/written from/to CSV files.

This module allows you to easily construct validation and correction rulesets to be applied automatically while preserving the csv interface. This allows you to easily upgrade old software to use new, strict rules.

class rigidity.**Rigidity**(*csvobj*, *rules=[]*)
A wrapper for CSV readers and writers that allows

skip()
Return a row, skipping validation. This is useful when you want to skip validation of header information.

validate(*row*)

> **Warning:** This method is deprecated and will be removed in a future release; it is included only to support old code. It will not produce consistent results with bi-directional rules. You should use *validate_read()* or *validate_write()* instead.

Validate that the row conforms with the specified rules, correcting invalid rows where the rule is able to do so.

If the row is valid or can be made valid through corrections, this method will return a row that can be written to the CSV file. If the row is invalid and cannot be corrected, then this method will raise an exception.

Parameters **row** – a row object that can be passed to a CSVWriter's writerow() method.

validate_read(*row*)
Validate that the row conforms with the specified rules, correcting invalid rows where the rule is able to do so.

If the row is valid or can be made valid through corrections, this method will return a row that can be written to the CSV file. If the row is invalid and cannot be corrected, then this method will raise an exception.

Parameters **row** – a row object that can be returned from CSVReader's readrow() method.

validate_write(*row*)
Validate that the row conforms with the specified rules, correcting invalid rows where the rule is able to do so.

If the row is valid or can be made valid through corrections, this method will return a row that can be written to the CSV file. If the row is invalid and cannot be corrected, then this method will raise an exception.

Parameters **row** – a row object that can be passed to a CSVWriter's __next__() method.

writeheader()
Plain pass-through to the given CSV object. It is assumed that header information is already valid when the CSV object is constructed.

writerow(*row*)
Validate and correct the data provided in *row* and raise an exception if the validation or correction fails. Then, write the row to the CSV file.

writerows(*rows*)
Validate and correct the data provided in every row and raise an exception if the validation or correction fails.

Note: Behavior in the case that the data is invalid and cannot be repaired is undefined. For example, the implementation may choose to write all valid rows up until the error, or it may choose to only conduct the write operation after all rows have been verified. Do not depend on the presence or absence of any of the rows in *rows* in the event that an exception occurs.

4.3 Rules

This submodule contains the built-in rules that are used for filtering and modifying data.

class rigidity.rules.**Boolean**(*allow_null=False*, *action=1*, *default=None*)
Bases: *rigidity.rules.Rule*

Cast a string as a boolean value.

ACTION_DEFAULT = 2
When invalid data is encountered, return a set defaut value.

ACTION_DROPROW = 3
When invalid data is encountered, drop the row.

ACTION_ERROR = 1
When invalid data is encountered, raise an exception.

__init__(*allow_null=False*, *action=1*, *default=None*)

Parameters **action** – take the behavior indicated by ACTION_ERROR, ACTION_DEFAULT, or ACTION_DROPROW.

class rigidity.rules.**Bytes**(*encoding='utf8'*)
Bases: *rigidity.rules.Rule*

When reading data, encode it as a bytes object using the given encoding. When writing data, decode it using the given encoding.

class rigidity.rules.**CapitalizeWords**(*seperators=' tnr'*, *cap_first=True*)
Bases: *rigidity.rules.Rule*

Capitalize words in a string. By default, words are detected by searching for space, tab, new line, and carriage return characters. You may override this setting.

Also, by default, the first character is capitalized automatically.

__init__(*seperators=' \t\n\r'*, *cap_first=True*)

Parameters

- **seperators** (*str*) – capitalize any character following a character in this string.
- **cap_first** (*bool*) – automatically capitalize the first character in the string.

class rigidity.rules.**Contains**(*string*)
Bases: *rigidity.rules.Rule*

Check that a string field value contains the string (or all strings in a list of strings) passed as a parameter to this rule.

class rigidity.rules.**Drop**
Bases: *rigidity.rules.Rule*

Drop the data in this column, replacing all data with an empty string value.

class rigidity.rules.**Float**(*action=1*)
Bases: *rigidity.rules.Rule*

Cast all data to floats or die trying.

ACTION_DROPROW = 3
When invalid data is encountered, drop the row.

ACTION_ERROR = 1
When invalid data is encountered, raise an exception.

ACTION_ZERO = 2
When invalid data is encountered, return zero.

__init__(*action=1*)

Parameters **action** – take the behavior indicated by ACTION_ERROR, ACTION_ZERO, or ACTION_DROPROW.

class rigidity.rules.**Integer**(*action=1*)
Bases: *rigidity.rules.Rule*

Cast all data to ints or die trying.

ACTION_DROPROW = 3
When invalid data is encountered, drop the row.

ACTION_ERROR = 1
When invalid data is encountered, raise an exception.

ACTION_ZERO = 2
When invalid data is encountered, return zero.

__init__(*action=1*)

Parameters **action** – take the behavior indicated by ACTION_ERROR, ACTION_ZERO, or ACTION_DROPROW.

class rigidity.rules.**Lower**
Bases: *rigidity.rules.Rule*

Convert a string value to lower-case.

class rigidity.rules.**NoneToEmptyString**
Bases: *rigidity.rules.Rule*

Replace None values with an empty string. This is useful in cases where legacy software uses None to create an empty cell, but your other checks require a string.

class rigidity.rules.**RemoveLinebreaks**
Bases: *rigidity.rules.Rule*

Remove linebreaks from the start and end of field values. These can sometimes be introduced into files and create problems for humans because they are invisible.to human users.

class rigidity.rules.**ReplaceValue**(*replacements={}*, *missing_action=4*, *default_value=''*)
Bases: *rigidity.rules.Rule*

Check if the value has a specified replacement. If it does, replace it with that value. If it does not, take one of the following configurable actions: pass it through unmodified, drop the row, or use a default value.

ACTION_BLANK = 5
When no replacement is found, return an empty string.

ACTION_DEFAULT_VALUE = 2
When no replacement is found, return a set default value.

ACTION_DROP = 5

> **Warning:** ACTION_DROP is deprecated due to the name being similar to ACTION_DROPROW. Use ACTION_BLANK instead.

ACTION_DROPROW = 1
When no replacement is found, drop the row.

ACTION_ERROR = 4
When no replacement is found, raise an exception.

ACTION_PASSTHROUGH = 3
When no replacement is found, allow the original to pass through.

__init__(*replacements={}*, *missing_action=4*, *default_value=''*)

Parameters

- **replacements** (*dict*) – a mapping between original values and replacement values.
- **missing_action** – when a replacement is not found for a value, take the behavior specified by the specified value, such as ACTION_DROP, ACTION_DEFAULT_VALUE, ACTION_PASSTHROUGH, or ACTION_ERROR.
- **default_value** – if ACTION_DEFAULT_VALUE is the missing replacement behavior, use this variable as the default replacement value.

class rigidity.rules.**Rule**
Base rule class implementing a simple apply() method that returns the given data unchanged.

apply(*value*)
This is the default method for applying a rule to data. By default, the *read()* and *write()* methods will use this method to validate and modify data.

Parameters **value** – the data to be validated.

Returns the validated and possibly modified value as documented by the rule.

Raises *rigidity.errors.DropRow* when the rule wants to cancel processing of an entire row, it may do so with the DropRow error. This signifies to the *rigidity.Rigidity* class that it should discontinue processing the row.

read(*value*)
When reading data, it is validated with this method. By default, this method calls the *apply()* method of this class. However, you may override this method to achieve different behavior when reading and writing.

Parameters **value** – the data to be validated.

Returns the validated and possibly modified value as documented by the rule.

Raises *rigidity.errors.DropRow* when the rule wants to cancel processing of an entire row, it may do so with the DropRow error. This signifies to the *rigidity.Rigidity* class that it should discontinue processing the row.

write(*value*)
When writing data, it is validated with this method. By default, this method calls the *apply()* method of this class. However, you may override this method to achieve different behavior when reading and writing.

Parameters **value** – the data to be validated.

Returns the validated and possibly modified value as documented by the rule.

Raises *rigidity.errors.DropRow* when the rule wants to cancel processing of an entire row, it may do so with the DropRow error. This signifies to the *rigidity.Rigidity* class that it should discontinue processing the row.

class rigidity.rules.**Static**(*value*)
Bases: *rigidity.rules.Rule*

Replace a field's value with a static value declared during initialization.

class rigidity.rules.**Strip**(*chars=None*)
Bases: *rigidity.rules.Rule*

Strip excess white space from the beginning and end of a value.

class rigidity.rules.**Unique**(*action=1*)
Bases: *rigidity.rules.Rule*

Only allow unique values to pass. When a repeated value is found, the row may be dropped or an error may be raised.

ACTION_DROPROW = 2
When repeat data is encountered, drop the row.

ACTION_ERROR = 1
When repeat data is encountered, raise an exception.

__init__(*action=1*)

Parameters **action** – Accepts either ACTION_ERROR or ACTION_DROPROW as the behavior to be performed when a value is not unique.

class rigidity.rules.**UpcA**(*strict=False*)
Bases: *rigidity.rules.Rule*

Validate UPC-A barscode numbers to ensure that they are 12 digits. Strict validation of the check digit may also be enabled.

__init__(*strict=False*)

Parameters **strict** (*bool*) – If *true*, raise a ValueError if the given UPC code fails the check digit validation.

apply(*value*)
Cast the value to a string, then check that it is numeric. Afterwards, zero-pad the left side to reach the standard length of 12 digits.

Raises ValueError when strict mode is enabled and the given UPC code fails the check digit validation.

class rigidity.rules.**Upper**
Bases: *rigidity.rules.Rule*

Convert a string value to upper-case.

CHAPTER 5

Indices and tables

- genindex
- modindex
- search

r

Symbols

A

B

C

D

F

I

L

N

R

S

U

V

W