
csvio

Release 0.3.0

Salman Raza

Nov 01, 2021

CONTENTS

1	Installation	3
2	Documentation	5
3	Reading CSVs	7
4	Writing CSVs	9
5	Apply field processors to transform row values	11
6	Create nested dictionaries with specified path	15
7	Contents	21
7.1	API Specifications	21
7.2	CHANGE LOG	52
	Index	53



csvio is a Python library that provides a wrapper around Python's built in `csv.DictReader` and `csv.DictWriter`, for ease of reading and writing CSV files.

Rows in a CSV are represented and processed as a list of dictionaries. Each item in this list is a dictionary that represents a row. The key, value pairs in each dictionary is a mapping between the column and its associated row value from the CSV.

INSTALLATION

```
pip install csvio
```


DOCUMENTATION

[Readthedocs](#)

READING CSVS

```
>>> from csvio import CSVReader
>>> reader = CSVReader("fruit_stock.csv")
>>> reader.fieldnames
['Supplier', 'Fruit', 'Quantity']
>>> len(reader.rows)
4

>>> import json
>>> print(json.dumps(reader.rows, indent=4))
[
  {
    "Supplier": "Big Apple",
    "Fruit": "Apple",
    "Quantity": "1"
  },
  {
    "Supplier": "Big Melons",
    "Fruit": "Melons",
    "Quantity": "2"
  },
  {
    "Supplier": "Long Mangoes",
    "Fruit": "Mango",
    "Quantity": "3"
  },
  {
    "Supplier": "Small Strawberries",
    "Fruit": "Strawberry",
    "Quantity": "4"
  }
]
```

CSV file contents:

```
Supplier,Fruit,Quantity
Big Apple,Apple,1
Big Melons,Melons,2
Long Mangoes,Mango,3
Small Strawberries,Strawberry,4
```


WRITING CSVS

```
>>> from csvio import CSVWriter
>>> writer = CSVWriter("fruit_stock.csv", fieldnames=["Supplier", "Fruit", "Quantity"])
>>> row1 = {"Supplier": "Big Apple", "Fruit": "Apple", "Quantity": 1}
>>> writer.add_rows(row1)
>>> rows2_3_4 = [
...     {"Supplier": "Big Melons", "Fruit": "Melons", "Quantity": 2},
...     {"Supplier": "Long Mangoes", "Fruit": "Mango", "Quantity": 3},
...     {"Supplier": "Small Strawberries", "Fruit": "Strawberry", "Quantity": 4}
... ]
>>> writer.add_rows(rows2_3_4)
>>> len(writer.pending_rows)
4

>>> len(writer.rows)
0

>>> writer.flush()
>>> len(writer.pending_rows)
0

>>> len(writer.rows)
4
```

Once flush is called a CSV file with the name *fruit_stock.csv* will be written with the following contents.

```
Supplier,Fruit,Quantity
Big Apple,Apple,1
Big Melons,Melons,2
Long Mangoes,Mango,3
Small Strawberries,Strawberry,4
```


APPLY FIELD PROCESSORS TO TRANSFORM ROW VALUES

Example with CSVReader

Example with CSVWriter

Standalone use

Processor function definitions

```
def add1(x):  
    return x + 1  
  
def cast_to_int(x):  
    return int(x)  
  
def replace_big_huge(x):  
    return x.replace("Big", "Huge")
```

Field processors and sample rows

```
from csvio.processors import FieldProcessor  
from json import dumps  
  
row1 = {  
    "Supplier": "Big Apples",  
    "Fruit": "Apple",  
    "Origin": "Spain",  
    "Quantity": "1"  
}  
  
row2 = {  
    "Supplier": "Big Melons",  
    "Fruit": "Melons",  
    "Origin": "Italy",  
    "Quantity": "2"  
}  
  
row3 = {  
    "Supplier": "Long Mangoes",  
    "Fruit": "Mango",  
    "Origin": "India",  
    "Quantity": "3"  
}
```

(continues on next page)

(continued from previous page)

```
rows = [row1, row2, row3]

proc1 = FieldProcessor('increment_qty')
proc1.add_processor("Quantity", cast_to_int)
proc1.add_processor("Quantity", add1)

proc2 = FieldProcessor('replace')
proc2.add_processor("Supplier", replace_big_huge)
```

Using implicit processor object

If a processor object or handle is not passed to the `process_row` method, the processor functions associated with the processor object whose `process_row` method we are calling are used implicitly.

```
print("Using implicit processor object:")
pretty_print("Before:", row1)
pretty_print("After:", proc1.process_row(row1)) # Using implicit processor object
```

Output

```
Using implicit processor object:
Before:
{
    "Supplier": "Big Apples",
    "Fruit": "Apple",
    "Origin": "Spain",
    "Quantity": "1"
}

After:
{
    "Supplier": "Big Apples",
    "Fruit": "Apple",
    "Origin": "Spain",
    "Quantity": 2
}
```

Using processor handle

Any processor object can be used to apply the processors from another object by using the handle reference as shown below. We are using the handle 'replace' associated with the `proc2` object, however we are using the `proc1` object to apply the processor.

```
print("Using processor handle:")
pretty_print("Before:", rows)
pretty_print("After:", proc1.process_rows(rows, 'replace')) # Using processor handle
```

Output

```
Using processor handle:
Before:
[
    {
```

(continues on next page)

(continued from previous page)

```

        "Supplier": "Big Apples",
        "Fruit": "Apple",
        "Origin": "Spain",
        "Quantity": "1"
    },
    {
        "Supplier": "Big Melons",
        "Fruit": "Melons",
        "Origin": "Italy",
        "Quantity": "2"
    },
    {
        "Supplier": "Long Mangoes",
        "Fruit": "Mango",
        "Origin": "India",
        "Quantity": "3"
    }
]

After:
[
    {
        "Supplier": "Huge Apples",
        "Fruit": "Apple",
        "Origin": "Spain",
        "Quantity": "1"
    },
    {
        "Supplier": "Huge Melons",
        "Fruit": "Melons",
        "Origin": "Italy",
        "Quantity": "2"
    },
    {
        "Supplier": "Long Mangoes",
        "Fruit": "Mango",
        "Origin": "India",
        "Quantity": "3"
    }
]

```

Using explicit processor object

Similarly we can also pass any other processor object instead of a handle.

```

print("Using explicit processor object:")
pretty_print("Before:", rows)
pretty_print("After:", proc1.process_rows(rows, proc2)) # Using explicit processor object

```

Output

```

Using explicit processor object:
Before:

```

(continues on next page)

(continued from previous page)

```
[
  {
    "Supplier": "Big Apples",
    "Fruit": "Apple",
    "Origin": "Spain",
    "Quantity": "1"
  },
  {
    "Supplier": "Big Melons",
    "Fruit": "Melons",
    "Origin": "Italy",
    "Quantity": "2"
  },
  {
    "Supplier": "Long Mangoes",
    "Fruit": "Mango",
    "Origin": "India",
    "Quantity": "3"
  }
]
```

After:

```
[
  {
    "Supplier": "Huge Apples",
    "Fruit": "Apple",
    "Origin": "Spain",
    "Quantity": "1"
  },
  {
    "Supplier": "Huge Melons",
    "Fruit": "Melons",
    "Origin": "Italy",
    "Quantity": "2"
  },
  {
    "Supplier": "Long Mangoes",
    "Fruit": "Mango",
    "Origin": "India",
    "Quantity": "3"
  }
]
```

CREATE NESTED DICTIONARIES WITH SPECIFIED PATH

CSV Contents: *fruit_stock.csv*

```
Supplier,Fruit,Origin,Quantity
Big Apples,Apple,Spain,1
Big Melons,Melons,Italy,2
Long Mangoes,Mango,India,3
Small Strawberries,Strawberry,France,4
Short Mangoes,Mango,France,5
Sweet Strawberries,Strawberry,Spain,6
Square Apples,Apple,Italy,7
Small Melons,Melons,Italy,8
Dark Berries,Strawberry,Australia,9
Sweet Berries,Blackcurrant,Australia,10
```

Create dictionary with hierarchy {"Fruit": [rows]}

```
from csvio.csvreader import CSVReader
from json import dumps

reader = CSVReader("fruit_stock.csv")

col_order = ["Fruit"]

dict_tree= reader.rows_to_nested_dicts(col_order)

print(dumps(dict_tree, indent=4))
```

Output:

```
{
  "Apple": [
    {
      "Supplier": "Big Apples",
      "Fruit": "Apple",
      "Origin": "Spain",
      "Quantity": "1"
    },
    {
      "Supplier": "Square Apples",
      "Fruit": "Apple",
      "Origin": "Italy",
```

(continues on next page)

(continued from previous page)

```

        "Quantity": "7"
    },
    ],
    "Melons": [
        {
            "Supplier": "Big Melons",
            "Fruit": "Melons",
            "Origin": "Italy",
            "Quantity": "2"
        },
        {
            "Supplier": "Small Melons",
            "Fruit": "Melons",
            "Origin": "Italy",
            "Quantity": "8"
        }
    ],
    "Mango": [
        {
            "Supplier": "Long Mangoes",
            "Fruit": "Mango",
            "Origin": "India",
            "Quantity": "3"
        },
        {
            "Supplier": "Short Mangoes",
            "Fruit": "Mango",
            "Origin": "France",
            "Quantity": "5"
        }
    ],
    "Strawberry": [
        {
            "Supplier": "Small Strawberries",
            "Fruit": "Strawberry",
            "Origin": "France",
            "Quantity": "4"
        },
        {
            "Supplier": "Sweet Strawberries",
            "Fruit": "Strawberry",
            "Origin": "Spain",
            "Quantity": "6"
        },
        {
            "Supplier": "Dark Berries",
            "Fruit": "Strawberry",
            "Origin": "Australia",
            "Quantity": "9"
        }
    ],
    "Blackcurrant": [

```

(continues on next page)

(continued from previous page)

```

    {
        "Supplier": "Sweet Berries",
        "Fruit": "Blackcurrant",
        "Origin": "Australia",
        "Quantity": "10"
    }
]
}

```

Create dictionary with hierarchy {"Fruit": "Origin" : [rows]}

```

from csvio.csvreader import CSVReader
from json import dumps

reader = CSVReader("fruit_stock.csv")

col_order = ["Fruit", "Origin"]

dict_tree= reader.rows_to_nested_dicts(col_order)

print(dumps(dict_tree, indent=4))

```

Output:

```

{
  "Apple": {
    "Spain": [
      {
        "Supplier": "Big Apples",
        "Fruit": "Apple",
        "Origin": "Spain",
        "Quantity": "1"
      }
    ],
    "Italy": [
      {
        "Supplier": "Square Apples",
        "Fruit": "Apple",
        "Origin": "Italy",
        "Quantity": "7"
      }
    ]
  },
  "Melons": {
    "Italy": [
      {
        "Supplier": "Big Melons",
        "Fruit": "Melons",
        "Origin": "Italy",
        "Quantity": "2"
      }
    ],
    {

```

(continues on next page)

(continued from previous page)

```

        "Supplier": "Small Melons",
        "Fruit": "Melons",
        "Origin": "Italy",
        "Quantity": "8"
    }
]
},
"Mango": {
    "India": [
        {
            "Supplier": "Long Mangoes",
            "Fruit": "Mango",
            "Origin": "India",
            "Quantity": "3"
        }
    ],
    "France": [
        {
            "Supplier": "Short Mangoes",
            "Fruit": "Mango",
            "Origin": "France",
            "Quantity": "5"
        }
    ]
},
"Strawberry": {
    "France": [
        {
            "Supplier": "Small Strawberries",
            "Fruit": "Strawberry",
            "Origin": "France",
            "Quantity": "4"
        }
    ],
    "Spain": [
        {
            "Supplier": "Sweet Strawberries",
            "Fruit": "Strawberry",
            "Origin": "Spain",
            "Quantity": "6"
        }
    ],
    "Australia": [
        {
            "Supplier": "Dark Berries",
            "Fruit": "Strawberry",
            "Origin": "Australia",
            "Quantity": "9"
        }
    ]
},
"Blackcurrant": {

```

(continues on next page)

(continued from previous page)

```
    "Australia": [  
        {  
            "Supplier": "Sweet Berries",  
            "Fruit": "Blackcurrant",  
            "Origin": "Australia",  
            "Quantity": "10"  
        }  
    ]  
}
```

Construct a dictionary with number of rows for each unique Origin

```
from csvio.csvreader import CSVReader  
from json import dumps  
  
reader = CSVReader("fruit_stock.csv")  
  
col_order = ["Origin"]  
  
origin_fruit_count = {}  
dict_tree = reader.rows_to_nested_dicts(col_order)  
  
for origin in dict_tree:  
    origin_fruit_count.setdefault(origin, len(dict_tree[origin]))  
  
print(dumps(origin_fruit_count, indent=4))
```

Output:

```
{  
    "Spain": 2,  
    "Italy": 3,  
    "India": 1,  
    "France": 2,  
    "Australia": 2  
}
```


CONTENTS

7.1 API Specifications

7.1.1 FileBase

class `csvio.filebase.FileBase(filename: str)`

Bases: `object`

This is a base class representing a basic file.

Parameters `filename` (*required*) – Full path to a file.

delete(*missing_ok: bool = False*) → `bool`

Delete the file at the path provided in the *filename* parameter

Parameters `missing_ok` (*optional*) – Parameter to pass to the `pathlib.Path.unlink()` method.

Returns

True If file is deleted successfully.

False On failure.

property `file_ext: str`

Returns Extension suffix of the file without parent directory and file name.

property `filedir: str`

Returns Parent directory path of the file (excluding the name of the file)

property `filename: str`

Returns File name without the parent directory path.

property `filename_no_ext: str`

Returns File name without parent directory and file extension.

property `filepath: str`

Returns Complete file path including the parent directory, file name and extension

property `path_obj: pathlib.Path`

Returns `pathlib.Path` object representing *filename*.

touch(*exist_ok: bool = False*) → `bool`

Create a blank file at the path provided in the *filename* parameter.

Parameters **exist_ok** (*optional*) – Parameter to pass to the `pathlib.Path.touch()` method.

Returns

True If blank file is created successfully.

False On failure.

7.1.2 CSVBase

class `csvio.csvbase.CSVBase`(*filename: str, open_kwargs: Dict[str, Any] = {}, csv_kwargs: Dict[str, Any] = {}*)

Bases: `csvio.filebase.FileBase`

This is a base class representing a basic CSV file for reading/writing.

Parameters

- **filename** (*required*) – Full path to the CSV file for reading/writing.
- **open_kwargs** (*optional*) – A dictionary of key, value pairs that should be passed to the open method within this class.
- **csv_kwargs** (*optional*) – A dictionary of key, value pairs that should be passed to the DictReader constructor within this class.

property `csv_kwargs: Dict[str, Any]`

Returns A dictionary of key, value pairs that should be passed to the DictReader constructor within this class.

delete(*missing_ok: bool = False*) → bool

Delete the file at the path provided in the *filename* parameter

Parameters **missing_ok** (*optional*) – Parameter to pass to the `pathlib.Path.unlink()` method.

Returns

True If file is deleted successfully.

False On failure.

property `fieldnames: List[str]`

Returns List of column headings

property `file_ext: str`

Returns Extension suffix of the file without parent directory and file name.

property `filedir: str`

Returns Parent directory path of the file (excluding the name of the file)

property `filename: str`

Returns File name without the parent directory path.

property `filename_no_ext: str`

Returns File name without parent directory and file extension.

property `filepath: str`

Returns Complete file path including the parent directory, file name and extension

property num_rows: int

Returns The total number of rows in the CSV (excluding column headings)

property open_kwargs: Dict[str, Any]

Returns A dictionary of key, value pairs that should be passed to the open method within this class.

property path_obj: pathlib.Path

Returns pathlib.Path object representing *filename*.

property rows: List[Dict[str, Any]]

Returns A list of dictionaries where each item in it represents a row in the CSV file. Each dictionary in the list maps the column heading (fieldname) to the corresponding value for it from the CSV.

rows_from_column_key(*column_name: str, rows: Optional[List[Dict[str, Any]]] = None*) → Dict[str, List[Dict[str, Any]]]

Collect all the rows in the *rows* parameter that have the same values for the column defined in the *column_name* parameter, and construct a dictionary with the *column_name* value as the key and the corresponding rows as a list of dictionaries, as the value of this key.

Parameters

- **column_name** (*required*) – Name of the column that is to be used as the key under which all the rows having the same value of this column will be collected.
- **rows** (optional. If not provided *self.rows* will be used.) – List of dictionaries representing the rows that will be separated and collected under a common value of the column name provided in *column_name* parameter.

Returns A dictionary constructed using the logic as explained above.

rows_to_nested_dicts(*column_order: List[str], rows: Optional[List[Dict[str, Any]]] = None*) → Dict[str, Any]

Collect all values of columns that are the same and construct a nested dictionary that has the common values as the keys, in the same order of hierarchy as provided in the *column_order* parameter.

The value of the last column name in the *column_order* list

Parameters

- **column_order** (*required*) – An ordered list of column names, to be used for constructing the dictionary
- **rows** (optional. If not provided *self.rows* will be used.) – List of dictionaries representing the rows that will be transformed to the output Dictionary.

Returns A dictionary with same column values collected under a common key in a hierarchical order.

Example:

CSV Contents: *fruit_stock.csv*

```
Supplier,Fruit,Origin,Quantity
Big Apples,Apple,Spain,1
Big Melons,Melons,Italy,2
Long Mangoes,Mango,India,3
Small Strawberries,Strawberry,France,4
```

(continues on next page)

(continued from previous page)

```

Short Mangoes,Mango,France,5
Sweet Strawberries,Strawberry,Spain,6
Square Apples,Apple,Italy,7
Small Melons,Melons,Italy,8
Dark Berries,Strawberry,Australia,9
Sweet Berries,Blackcurrant,Australia,10

```

Create dictionary with hierarchy {"Fruit": [rows]}

```

from csvio.csvreader import CSVReader
from json import dumps

reader = CSVReader("fruit_stock.csv")

col_order = ["Fruit"]

dict_tree= reader.rows_to_nested_dicts(col_order)

print(dumps(dict_tree, indent=4))

```

Output:

```

{
  "Apple": [
    {
      "Supplier": "Big Apples",
      "Fruit": "Apple",
      "Origin": "Spain",
      "Quantity": "1"
    },
    {
      "Supplier": "Square Apples",
      "Fruit": "Apple",
      "Origin": "Italy",
      "Quantity": "7"
    }
  ],
  "Melons": [
    {
      "Supplier": "Big Melons",
      "Fruit": "Melons",
      "Origin": "Italy",
      "Quantity": "2"
    },
    {
      "Supplier": "Small Melons",
      "Fruit": "Melons",
      "Origin": "Italy",
      "Quantity": "8"
    }
  ],
  "Mango": [

```

(continues on next page)

(continued from previous page)

```

    {
        "Supplier": "Long Mangoes",
        "Fruit": "Mango",
        "Origin": "India",
        "Quantity": "3"
    },
    {
        "Supplier": "Short Mangoes",
        "Fruit": "Mango",
        "Origin": "France",
        "Quantity": "5"
    }
],
"Strawberry": [
    {
        "Supplier": "Small Strawberries",
        "Fruit": "Strawberry",
        "Origin": "France",
        "Quantity": "4"
    },
    {
        "Supplier": "Sweet Strawberries",
        "Fruit": "Strawberry",
        "Origin": "Spain",
        "Quantity": "6"
    },
    {
        "Supplier": "Dark Berries",
        "Fruit": "Strawberry",
        "Origin": "Australia",
        "Quantity": "9"
    }
],
"Blackcurrant": [
    {
        "Supplier": "Sweet Berries",
        "Fruit": "Blackcurrant",
        "Origin": "Australia",
        "Quantity": "10"
    }
]
]
}

```

Create dictionary with hierarchy {"Fruit": "Origin" : [rows]}

```

from csvio.csvreader import CSVReader
from json import dumps

reader = CSVReader("fruit_stock.csv")

col_order = ["Fruit", "Origin"]

```

(continues on next page)

(continued from previous page)

```
dict_tree= reader.rows_to_nested_dicts(col_order)

print(dumps(dict_tree, indent=4))
```

Output:

```
{
  "Apple": {
    "Spain": [
      {
        "Supplier": "Big Apples",
        "Fruit": "Apple",
        "Origin": "Spain",
        "Quantity": "1"
      }
    ],
    "Italy": [
      {
        "Supplier": "Square Apples",
        "Fruit": "Apple",
        "Origin": "Italy",
        "Quantity": "7"
      }
    ]
  },
  "Melons": {
    "Italy": [
      {
        "Supplier": "Big Melons",
        "Fruit": "Melons",
        "Origin": "Italy",
        "Quantity": "2"
      },
      {
        "Supplier": "Small Melons",
        "Fruit": "Melons",
        "Origin": "Italy",
        "Quantity": "8"
      }
    ]
  },
  "Mango": {
    "India": [
      {
        "Supplier": "Long Mangoes",
        "Fruit": "Mango",
        "Origin": "India",
        "Quantity": "3"
      }
    ],
    "France": [
      {
```

(continues on next page)

(continued from previous page)

```

        "Supplier": "Short Mangoes",
        "Fruit": "Mango",
        "Origin": "France",
        "Quantity": "5"
    }
]
},
"Strawberry": {
    "France": [
        {
            "Supplier": "Small Strawberries",
            "Fruit": "Strawberry",
            "Origin": "France",
            "Quantity": "4"
        }
    ],
    "Spain": [
        {
            "Supplier": "Sweet Strawberries",
            "Fruit": "Strawberry",
            "Origin": "Spain",
            "Quantity": "6"
        }
    ],
    "Australia": [
        {
            "Supplier": "Dark Berries",
            "Fruit": "Strawberry",
            "Origin": "Australia",
            "Quantity": "9"
        }
    ]
},
"Blackcurrant": {
    "Australia": [
        {
            "Supplier": "Sweet Berries",
            "Fruit": "Blackcurrant",
            "Origin": "Australia",
            "Quantity": "10"
        }
    ]
}
}

```

Construct a dictionary with number of rows for each unique Origin

```

from csvio.csvreader import CSVReader
from json import dumps

reader = CSVReader("fruit_stock.csv")

```

(continues on next page)

(continued from previous page)

```
col_order = ["Origin"]

origin_fruit_count = {}
dict_tree = reader.rows_to_nested_dicts(col_order)

for origin in dict_tree:
    origin_fruit_count.setdefault(origin, len(dict_tree[origin]))

print(dumps(origin_fruit_count, indent=4))
```

Output:

```
{
  "Spain": 2,
  "Italy": 3,
  "India": 1,
  "France": 2,
  "Australia": 2
}
```

touch(*exist_ok: bool = False*) → boolCreate a blank file at the path provided in the *filename* parameter.**Parameters** **exist_ok** (*optional*) – Parameter to pass to the `pathlib.Path.touch()` method.**Returns**

True If blank file is created successfully.

False On failure.

7.1.3 CSVReader

```
class csvio.CSVReader(filename: str, fieldprocessor: Optional[csvio.processors.field_processor.FieldProcessor]
                      = None, fieldnames: List[str] = [], open_kwargs: Dict[str, Any] = {}, csv_kwargs:
                      Dict[str, Any] = {})
```

Bases: `csvio.csvbase.CSVBase`

This object represents a CSV file for reading.

Parameters

- **filename** (*required*) – Full path to the CSV file for reading.
- **fieldnames** (*optional*) – A list of strings representing the column headings for the CSV file. If this list is specified while initiating an Object of this class then it is used as the column headings. This is handy when the CSV to read does not have column headings. Otherwise this list is populated from the CSV that is set in the *filename* argument of this Class's constructor.
- **fieldprocessor** (*optional*) – An instance of the `FieldProcessor` object. The processor functions defined in the `FieldProcessor` object are applied to the rows in the CSV after they read.
- **open_kwargs** (*optional*) – A dictionary of key, value pairs that should be passed to the open method within this class.

- **csv_kwargs** (*optional*) – A dictionary of key, value pairs that should be passed to the DictReader constructor within this class.

CSVReader usage without fieldprocessor:

```
>>> from csvio import CSVReader
>>> reader = CSVReader("fruit_stock.csv")
>>> reader.fieldnames
['Supplier', 'Fruit', 'Quantity']

>>> len(reader.rows)
4

>>> import json
>>> print(json.dumps(reader.rows, indent=4))
[
  {
    "Supplier": "Big Apple",
    "Fruit": "Apple",
    "Quantity": "1"
  },
  {
    "Supplier": "Big Melons",
    "Fruit": "Melons",
    "Quantity": "2"
  },
  {
    "Supplier": "Big Mangoes",
    "Fruit": "Mango",
    "Quantity": "3"
  },
  {
    "Supplier": "Small Strawberries",
    "Fruit": "Strawberry",
    "Quantity": "4"
  }
]
```

CSV file contents:

```
Supplier,Fruit,Quantity
Big Apple,Apple,1
Big Melons,Melons,2
Long Mangoes,Mango,3
Small Strawberries,Strawberry,4
```

CSVReader usage with fieldprocessor

Original CSV Contents: fruit_stock.csv

```
Supplier,Fruit,Origin,Quantity
Big Apples,Apple,Spain,1
Big Melons,Melons,Italy,2
Long Mangoes,Mango,India,3
Small Strawberries,Strawberry,France,4
```

(continues on next page)

(continued from previous page)

```

Short Mangoes,Mango,France,5
Sweet Strawberries,Strawberry,Spain,6
Square Apples,Apple,Italy,7
Small Melons,Melons,Italy,8
Dark Berries,Strawberry,Australia,9
Sweet Berries,Blackcurrant,Australia,10

```

Processor function definitions

```

def add1(x):
    return x + 1

def cast_to_int(x):
    return int(x)

def replace_big_huge(x):
    return x.replace("Big", "Huge")

```

Define and apply processors

```

from csvio import CSVReader, CSVWriter
from csvio.processors import FieldProcessor

processor = FieldProcessor("proc1")

processor.add_processor("Quantity", [cast_to_int, add1])
processor.add_processor("Supplier", replace_big_huge)
processor.add_processor("Origin", lambda x: x.upper())

processor.add_processor(
    "Supplier", lambda x: x.replace("Strawberries", "Strawberry")
)

processor.add_processor("Supplier", lambda x: x.replace("Huge", "Enormous"))

reader = CSVReader("fruit_stock.csv", processor)

writer = CSVWriter(
    "fruit_stock_processed.csv",
    fieldnames=reader.fieldnames
)

writer.add_rows(reader.rows)
writer.flush()

```

CSV Contents after Processing: fruit_stock_processed.csv

```

Supplier,Fruit,Origin,Quantity
Enormous Apples,Apple,SPAIN,2
Enormous Melons,Melons,ITALY,3
Long Mangoes,Mango,INDIA,4
Small Strawberry,Strawberry,FRANCE,5
Short Mangoes,Mango,FRANCE,6

```

(continues on next page)

(continued from previous page)

```
Sweet Strawberry, Strawberry, SPAIN, 7
Square Apples, Apple, ITALY, 8
Small Melons, Melons, ITALY, 9
Dark Berries, Strawberry, AUSTRALIA, 10
Sweet Berries, Blackcurrant, AUSTRALIA, 11
```

property csv_kwargs: Dict[str, Any]

Returns A dictionary of key, value pairs that should be passed to the DictReader constructor within this class.

delete(missing_ok: bool = False) → bool

Delete the file at the path provided in the *filename* parameter

Parameters **missing_ok** (optional) – Parameter to pass to the pathlib.Path.unlink() method.

Returns

True If file is deleted successfully.

False On failure.

property fieldnames: List[str]

Returns List of column headings

property file_ext: str

Returns Extension suffix of the file without parent directory and file name.

property filedir: str

Returns Parent directory path of the file (excluding the name of the file)

property filename: str

Returns File name without the parent directory path.

property filename_no_ext: str

Returns File name without parent directory and file extension.

property filepath: str

Returns Complete file path including the parent directory, file name and extension

property num_rows: int

Returns The total number of rows in the CSV (excluding column headings)

property open_kwargs: Dict[str, Any]

Returns A dictionary of key, value pairs that should be passed to the open method within this class.

property path_obj: pathlib.Path

Returns pathlib.Path object representing *filename*.

property rows: List[Dict[str, Any]]

Returns A list of dictionaries where each item in it represents a row in the CSV file. Each dictionary in the list maps the column heading (fieldname) to the corresponding value for it from the CSV.

rows_from_column_key(*column_name*: str, *rows*: Optional[List[Dict[str, Any]]] = None) → Dict[str, List[Dict[str, Any]]]

Collect all the rows in the `rows` parameter that have the same values for the column defined in the `column_name` parameter, and construct a dictionary with the `column_name` value as the key and the corresponding rows as a list of dictionaries, as the value of this key.

Parameters

- **column_name** (*required*) – Name of the column that is to be used as the key under which all the rows having the same value of this column will be collected.
- **rows** (optional. If not provided `self.rows` will be used.) – List of dictionaries representing the rows that will be separated and collected under a the common value of the column name provided in `column_name` parameter.

Returns A dictionary constructed using the logic as explained above.

rows_to_nested_dicts(*column_order*: List[str], *rows*: Optional[List[Dict[str, Any]]] = None) → Dict[str, Any]

Collect all values of columns that are the same and construct a nested dictionary that has the common values as the keys, in the same order of hierarchy as provided in the `column_order` parameter.

The value of the last column name in the `column_order` list

Parameters

- **column_order** (*required*) – An ordered list of column names, to be used for constructing the dictionary
- **rows** (optional. If not provided `self.rows` will be used.) – List of dictionaries representing the rows that will be transformed to the output Dictionary.

Returns A dictionary with same column values collected under a common key in a hierarchical order.

Example:

CSV Contents: *fruit_stock.csv*

```
Supplier,Fruit,Origin,Quantity
Big Apples,Apple,Spain,1
Big Melons,Melons,Italy,2
Long Mangoes,Mango,India,3
Small Strawberries,Strawberry,France,4
Short Mangoes,Mango,France,5
Sweet Strawberries,Strawberry,Spain,6
Square Apples,Apple,Italy,7
Small Melons,Melons,Italy,8
Dark Berries,Strawberry,Australia,9
Sweet Berries,Blackcurrant,Australia,10
```

Create dictionary with hierarchy {"Fruit": [rows]}

```
from csvio.csvreader import CSVReader
from json import dumps

reader = CSVReader("fruit_stock.csv")

col_order = ["Fruit"]
```

(continues on next page)

(continued from previous page)

```
dict_tree= reader.rows_to_nested_dicts(col_order)

print(dumps(dict_tree, indent=4))
```

Output:

```
{
  "Apple": [
    {
      "Supplier": "Big Apples",
      "Fruit": "Apple",
      "Origin": "Spain",
      "Quantity": "1"
    },
    {
      "Supplier": "Square Apples",
      "Fruit": "Apple",
      "Origin": "Italy",
      "Quantity": "7"
    }
  ],
  "Melons": [
    {
      "Supplier": "Big Melons",
      "Fruit": "Melons",
      "Origin": "Italy",
      "Quantity": "2"
    },
    {
      "Supplier": "Small Melons",
      "Fruit": "Melons",
      "Origin": "Italy",
      "Quantity": "8"
    }
  ],
  "Mango": [
    {
      "Supplier": "Long Mangoes",
      "Fruit": "Mango",
      "Origin": "India",
      "Quantity": "3"
    },
    {
      "Supplier": "Short Mangoes",
      "Fruit": "Mango",
      "Origin": "France",
      "Quantity": "5"
    }
  ],
  "Strawberry": [
    {
```

(continues on next page)

(continued from previous page)

```

        "Supplier": "Small Strawberries",
        "Fruit": "Strawberry",
        "Origin": "France",
        "Quantity": "4"
    },
    {
        "Supplier": "Sweet Strawberries",
        "Fruit": "Strawberry",
        "Origin": "Spain",
        "Quantity": "6"
    },
    {
        "Supplier": "Dark Berries",
        "Fruit": "Strawberry",
        "Origin": "Australia",
        "Quantity": "9"
    }
],
"Blackcurrant": [
    {
        "Supplier": "Sweet Berries",
        "Fruit": "Blackcurrant",
        "Origin": "Australia",
        "Quantity": "10"
    }
]
}

```

Create dictionary with hierarchy {"Fruit": "Origin" : [rows]}

```

from csvio.csvreader import CSVReader
from json import dumps

reader = CSVReader("fruit_stock.csv")

col_order = ["Fruit", "Origin"]

dict_tree= reader.rows_to_nested_dicts(col_order)

print(dumps(dict_tree, indent=4))

```

Output:

```

{
    "Apple": {
        "Spain": [
            {
                "Supplier": "Big Apples",
                "Fruit": "Apple",
                "Origin": "Spain",
                "Quantity": "1"
            }
        ]
    }
}

```

(continues on next page)

(continued from previous page)

```
    ],
    "Italy": [
      {
        "Supplier": "Square Apples",
        "Fruit": "Apple",
        "Origin": "Italy",
        "Quantity": "7"
      }
    ]
  },
  "Melons": {
    "Italy": [
      {
        "Supplier": "Big Melons",
        "Fruit": "Melons",
        "Origin": "Italy",
        "Quantity": "2"
      },
      {
        "Supplier": "Small Melons",
        "Fruit": "Melons",
        "Origin": "Italy",
        "Quantity": "8"
      }
    ]
  },
  "Mango": {
    "India": [
      {
        "Supplier": "Long Mangoes",
        "Fruit": "Mango",
        "Origin": "India",
        "Quantity": "3"
      }
    ],
    "France": [
      {
        "Supplier": "Short Mangoes",
        "Fruit": "Mango",
        "Origin": "France",
        "Quantity": "5"
      }
    ]
  },
  "Strawberry": {
    "France": [
      {
        "Supplier": "Small Strawberries",
        "Fruit": "Strawberry",
        "Origin": "France",
        "Quantity": "4"
      }
    ]
  }
}
```

(continues on next page)

(continued from previous page)

```

    ],
    "Spain": [
        {
            "Supplier": "Sweet Strawberries",
            "Fruit": "Strawberry",
            "Origin": "Spain",
            "Quantity": "6"
        }
    ],
    "Australia": [
        {
            "Supplier": "Dark Berries",
            "Fruit": "Strawberry",
            "Origin": "Australia",
            "Quantity": "9"
        }
    ]
},
"Blackcurrant": {
    "Australia": [
        {
            "Supplier": "Sweet Berries",
            "Fruit": "Blackcurrant",
            "Origin": "Australia",
            "Quantity": "10"
        }
    ]
}
}

```

Construct a dictionary with number of rows for each unique Origin

```

from csvio.csvreader import CSVReader
from json import dumps

reader = CSVReader("fruit_stock.csv")

col_order = ["Origin"]

origin_fruit_count = {}
dict_tree = reader.rows_to_nested_dicts(col_order)

for origin in dict_tree:
    origin_fruit_count.setdefault(origin, len(dict_tree[origin]))

print(dumps(origin_fruit_count, indent=4))

```

Output:

```

{
    "Spain": 2,
    "Italy": 3,

```

(continues on next page)

(continued from previous page)

```

    "India": 1,
    "France": 2,
    "Australia": 2
}

```

touch(*exist_ok: bool = False*) → bool

Create a blank file at the path provided in the *filename* parameter.

Parameters *exist_ok* (*optional*) – Parameter to pass to the `pathlib.Path.touch()` method.

Returns

True If blank file is created successfully.

False On failure.

7.1.4 CSVWriter

class `csvio.CSVWriter`(*filename: str, fieldnames: List[str], fieldprocessor: Optional[csvio.processors.field_processor.FieldProcessor] = None, open_kwargs: Dict[str, str] = {}, csv_kwargs: Dict[str, Any] = {}*)

Bases: `csvio.csvbase.CSVBase`

This object represents a CSV file for writing.

Parameters

- **filename** (*required*) – Full path to the CSV file for writing.
- **fieldnames** (*required*) – A list of strings representing the column headings for the CSV file.
- **fieldprocessor** (*optional*) – An instance of the `FieldProcessor` object. The processor functions defined in the `FieldProcessor` object are applied to the rows as soon as they are added for writing to the output CSV using `add_rows()` method
- **open_kwargs** (*optional*) – A dictionary of key, value pairs that should be passed to the open method within this class.
- **csv_kwargs** (*optional*) – A dictionary of key, value pairs that should be passed to the DictReader constructor within this class.

add_rows(*rows: Union[Dict[str, Any], List[Dict[str, Any]]]*) → None

Add rows for writing to the output CSV.

All the rows to be written to the output CSV are collected using this method.

This only collects the rows to be written without writing anything to the output CSV. The rows are written to output only when the method `csvio.CSVWriter.flush()` is called.

Parameters *rows* (*required*) – A single dictionary or a list of dictionaries that represent the row(s) to be written to the output CSV.

CSVWriter usage without fieldprocessor:

```

>>> from csvio import CSVWriter
>>> writer = CSVWriter("fruit_stock.csv", fieldnames=["Supplier", "Fruit",
↳ "Quantity"])

```

(continues on next page)

(continued from previous page)

```

>>> row1 = {"Supplier": "Big Apple", "Fruit": "Apple", "Quantity": 1}
>>> writer.add_rows(row1)
>>> rows2_3_4 = [
...     {"Supplier": "Big Melons", "Fruit": "Melons", "Quantity": 2},
...     {"Supplier": "Long Mangoes", "Fruit": "Mango", "Quantity": 3},
...     {"Supplier": "Small Strawberries", "Fruit": "Strawberry", "Quantity": 4}
... ]
>>> writer.add_rows(rows2_3_4)
>>> len(writer.pending_rows)
4
>>> len(writer.rows)
0

```

Notice that the `rows` property is still empty. This property is incremented by the number of currently pending rows once they are flushed using `flush()`. **CSVWriter usage with fieldprocessor:**

```

from csvio import CSVWriter
from csvio.processors import FieldProcessor
from json import dumps

def add1(x):
    return x + 1

def cast_to_int(x):
    return int(x)

def replace_big_huge(x):
    return x.replace("Big", "Huge")

processor = FieldProcessor("proc1")

processor.add_processor("Quantity", [cast_to_int, add1])
processor.add_processor("Supplier", replace_big_huge)
processor.add_processor("Origin", lambda x: x.upper())

processor.add_processor(
    "Supplier", lambda x: x.replace("Strawberries", "Strawberry")
)

processor.add_processor("Supplier", lambda x: x.replace("Huge", "Enormous"))

row1 = {
    "Supplier": "Big Apples",
    "Fruit": "Apple",
    "Origin": "Spain",
    "Quantity": "1"
}

row2 = {
    "Supplier": "Big Melons",

```

(continues on next page)

(continued from previous page)

```

    "Fruit": "Melons",
    "Origin": "Italy",
    "Quantity": "2"
}

row3 = {
    "Supplier": "Long Mangoes",
    "Fruit": "Mango",
    "Origin": "India",
    "Quantity": "3"
}

rows = [row1, row2, row3]

writer = CSVWriter(
    "fruit_stock_processed.csv",
    ["Supplier", "Fruit", "Origin", "Quantity"],
    processor
)

writer.add_rows(rows)
writer.flush()

print("Before:")
print(dumps(rows, indent=4))
print()

print("After:")
print(dumps(writer.rows, indent=4))

```

Output

```

Before:
[
  {
    "Supplier": "Big Apples",
    "Fruit": "Apple",
    "Origin": "Spain",
    "Quantity": "1"
  },
  {
    "Supplier": "Big Melons",
    "Fruit": "Melons",
    "Origin": "Italy",
    "Quantity": "2"
  },
  {
    "Supplier": "Long Mangoes",
    "Fruit": "Mango",
    "Origin": "India",
    "Quantity": "3"
  }
]

```

(continues on next page)

(continued from previous page)

```

]

After:
[
  {
    "Supplier": "Enormous Apples",
    "Fruit": "Apple",
    "Origin": "SPAIN",
    "Quantity": 2
  },
  {
    "Supplier": "Enormous Melons",
    "Fruit": "Melons",
    "Origin": "ITALY",
    "Quantity": 3
  },
  {
    "Supplier": "Long Mangoes",
    "Fruit": "Mango",
    "Origin": "INDIA",
    "Quantity": 4
  }
]

```

Contents of fruit_stock_processed.csv

```

Supplier,Fruit,Origin,Quantity
Enormous Apples,Apple,SPAIN,2
Enormous Melons,Melons,ITALY,3
Long Mangoes,Mango,INDIA,4

```

property `csv_kwargs: Dict[str, Any]`

Returns A dictionary of key, value pairs that should be passed to the DictReader constructor within this class.

delete(*missing_ok: bool = False*) → bool

Delete the file at the path provided in the *filename* parameter

Parameters `missing_ok` (*optional*) – Parameter to pass to the `pathlib.Path.unlink()` method.

Returns

True If file is deleted successfully.

False On failure.

property `fieldnames: List[str]`

Returns List of column headings

property `file_ext: str`

Returns Extension suffix of the file without parent directory and file name.

property `filedir: str`

Returns Parent directory path of the file (excluding the name of the file)

property filename: str

Returns File name without the parent directory path.

property filename_no_ext: str

Returns File name without parent directory and file extension.

property filepath: str

Returns Complete file path including the parent directory, file name and extension

flush() → None

Write pending rows to the output CSV and reset the `CSVWriter.pending_rows` property to an empty list

Usage:

```
>>> from csvio import CSVWriter
>>> writer = CSVWriter("fruit_stock.csv", fieldnames=["Supplier", "Fruit",
↳ "Quantity"])
>>> row1 = {"Supplier": "Big Apple", "Fruit": "Apple", "Quantity": 1}
>>> writer.add_rows(row1)
>>> rows2_3_4 = [
...     {"Supplier": "Big Melons", "Fruit": "Melons", "Quantity": 2},
...     {"Supplier": "Long Mangoes", "Fruit": "Mango", "Quantity": 3},
...     {"Supplier": "Small Strawberries", "Fruit": "Strawberry", "Quantity": 4}
... ]
>>> writer.add_rows(rows2_3_4)
>>> len(writer.pending_rows)
4

>>> len(writer.rows)
0

>>> writer.flush()
>>> len(writer.pending_rows)
0

>>> len(writer.rows)
4
```

Once flush is called a CSV file with the name *fruit_stock.csv* will be written with the following contents.

```
Supplier,Fruit,Quantity
Big Apple,Apple,1
Big Melons,Melons,2
Long Mangoes,Mango,3
Small Strawberries,Strawberry,4
```

property num_rows: int

Returns The total number of rows in the CSV (excluding column headings)

property open_kwargs: Dict[str, Any]

Returns A dictionary of key, value pairs that should be passed to the open method within this class.

property path_obj: pathlib.Path

Returns `pathlib.Path` object representing *filename*.

property pending_rows: `List[Dict[str, Any]]`

Returns List of rows not flushed yet and are pending to be written

property rows: `List[Dict[str, Any]]`

Returns A list of dictionaries where each item in it represents a row in the CSV file. Each dictionary in the list maps the column heading (fieldname) to the corresponding value for it from the CSV.

rows_from_column_key(*column_name: str, rows: Optional[List[Dict[str, Any]]] = None*) → `Dict[str, List[Dict[str, Any]]]`

Collect all the rows in the `rows` parameter that have the same values for the column defined in the `column_name` parameter, and construct a dictionary with the `column_name` value as the key and the corresponding rows as a list of dictionaries, as the value of this key.

Parameters

- **column_name** (*required*) – Name of the column that is to be used as the key under which all the rows having the same value of this column will be collected.
- **rows** (optional. If not provided `self.rows` will be used.) – List of dictionaries representing the rows that will be separated and collected under a the common value of the column name provided in `column_name` parameter.

Returns A dictionary constructed using the logic as explained above.

rows_to_nested_dicts(*column_order: List[str], rows: Optional[List[Dict[str, Any]]] = None*) → `Dict[str, Any]`

Collect all values of columns that are the same and construct a nested dictionary that has the common values as the keys, in the same order of hierarchy as provided in the `column_order` parameter.

The value of the last column name in the `column_order` list

Parameters

- **column_order** (*required*) – An ordered list of column names, to be used for constructing the dictionary
- **rows** (optional. If not provided `self.rows` will be used.) – List of dictionaries representing the rows that will be transformed to the output Dictionary.

Returns A dictionary with same column values collected under a common key in a hierarchical order.

Example:

CSV Contents: *fruit_stock.csv*

```
Supplier,Fruit,Origin,Quantity
Big Apples,Apple,Spain,1
Big Melons,Melons,Italy,2
Long Mangoes,Mango,India,3
Small Strawberries,Strawberry,France,4
Short Mangoes,Mango,France,5
Sweet Strawberries,Strawberry,Spain,6
Square Apples,Apple,Italy,7
Small Melons,Melons,Italy,8
Dark Berries,Strawberry,Australia,9
Sweet Berries,Blackcurrant,Australia,10
```

Create dictionary with hierarchy {"Fruit": [rows]}

```
from csvio.csvreader import CSVReader
from json import dumps

reader = CSVReader("fruit_stock.csv")

col_order = ["Fruit"]

dict_tree= reader.rows_to_nested_dicts(col_order)

print(dumps(dict_tree, indent=4))
```

Output:

```
{
  "Apple": [
    {
      "Supplier": "Big Apples",
      "Fruit": "Apple",
      "Origin": "Spain",
      "Quantity": "1"
    },
    {
      "Supplier": "Square Apples",
      "Fruit": "Apple",
      "Origin": "Italy",
      "Quantity": "7"
    }
  ],
  "Melons": [
    {
      "Supplier": "Big Melons",
      "Fruit": "Melons",
      "Origin": "Italy",
      "Quantity": "2"
    },
    {
      "Supplier": "Small Melons",
      "Fruit": "Melons",
      "Origin": "Italy",
      "Quantity": "8"
    }
  ],
  "Mango": [
    {
      "Supplier": "Long Mangoes",
      "Fruit": "Mango",
      "Origin": "India",
      "Quantity": "3"
    },
    {
      "Supplier": "Short Mangoes",
      "Fruit": "Mango",
```

(continues on next page)

(continued from previous page)

```

        "Origin": "France",
        "Quantity": "5"
    },
],
"Strawberry": [
    {
        "Supplier": "Small Strawberries",
        "Fruit": "Strawberry",
        "Origin": "France",
        "Quantity": "4"
    },
    {
        "Supplier": "Sweet Strawberries",
        "Fruit": "Strawberry",
        "Origin": "Spain",
        "Quantity": "6"
    },
    {
        "Supplier": "Dark Berries",
        "Fruit": "Strawberry",
        "Origin": "Australia",
        "Quantity": "9"
    }
],
"Blackcurrant": [
    {
        "Supplier": "Sweet Berries",
        "Fruit": "Blackcurrant",
        "Origin": "Australia",
        "Quantity": "10"
    }
]
}

```

Create dictionary with hierarchy {"Fruit": "Origin" : [rows]}

```

from csvio.csvreader import CSVReader
from json import dumps

reader = CSVReader("fruit_stock.csv")

col_order = ["Fruit", "Origin"]

dict_tree= reader.rows_to_nested_dicts(col_order)

print(dumps(dict_tree, indent=4))

```

Output:

```

{
    "Apple": {
        "Spain": [

```

(continues on next page)

(continued from previous page)

```

        {
            "Supplier": "Big Apples",
            "Fruit": "Apple",
            "Origin": "Spain",
            "Quantity": "1"
        }
    ],
    "Italy": [
        {
            "Supplier": "Square Apples",
            "Fruit": "Apple",
            "Origin": "Italy",
            "Quantity": "7"
        }
    ]
},
"Melons": {
    "Italy": [
        {
            "Supplier": "Big Melons",
            "Fruit": "Melons",
            "Origin": "Italy",
            "Quantity": "2"
        },
        {
            "Supplier": "Small Melons",
            "Fruit": "Melons",
            "Origin": "Italy",
            "Quantity": "8"
        }
    ]
},
"Mango": {
    "India": [
        {
            "Supplier": "Long Mangoes",
            "Fruit": "Mango",
            "Origin": "India",
            "Quantity": "3"
        }
    ],
    "France": [
        {
            "Supplier": "Short Mangoes",
            "Fruit": "Mango",
            "Origin": "France",
            "Quantity": "5"
        }
    ]
},
"Strawberry": {
    "France": [

```

(continues on next page)

(continued from previous page)

```

        {
            "Supplier": "Small Strawberries",
            "Fruit": "Strawberry",
            "Origin": "France",
            "Quantity": "4"
        }
    ],
    "Spain": [
        {
            "Supplier": "Sweet Strawberries",
            "Fruit": "Strawberry",
            "Origin": "Spain",
            "Quantity": "6"
        }
    ],
    "Australia": [
        {
            "Supplier": "Dark Berries",
            "Fruit": "Strawberry",
            "Origin": "Australia",
            "Quantity": "9"
        }
    ]
},
"Blackcurrant": {
    "Australia": [
        {
            "Supplier": "Sweet Berries",
            "Fruit": "Blackcurrant",
            "Origin": "Australia",
            "Quantity": "10"
        }
    ]
}
}

```

Construct a dictionary with number of rows for each unique Origin

```

from csvio.csvreader import CSVReader
from json import dumps

reader = CSVReader("fruit_stock.csv")

col_order = ["Origin"]

origin_fruit_count = {}
dict_tree = reader.rows_to_nested_dicts(col_order)

for origin in dict_tree:
    origin_fruit_count.setdefault(origin, len(dict_tree[origin]))

print(dumps(origin_fruit_count, indent=4))

```

Output:

```
{
  "Spain": 2,
  "Italy": 3,
  "India": 1,
  "France": 2,
  "Australia": 2
}
```

touch(*exist_ok: bool = False*) → bool

Create a blank file at the path provided in the *filename* parameter.

Parameters **exist_ok** (*optional*) – Parameter to pass to the `pathlib.Path.touch()` method.

Returns

True If blank file is created successfully.

False On failure.

write_blank_csv() → None

Write a blank CSV with only the column headings.

If the CSV already exists with any rows in it, it will be overwritten and its contents will be replaced with only the column headings.

7.1.5 Field Processor

A Field Processor may be used to transform the values in a row represented by a dictionary that maps column->value pairs.

In *csvio* a CSV file is represented by a list of dictionaries that is populated in the *rows* attribute of the *CSVReader* or *CSVWriter* Classes.

Once instantiated, a Field Processor Object can be used by itself to process an arbitrary dictionary that represents a row or can be passed to the constructors of *CSVReader* or *CSVWriter*.

In the case where a Field Processor Object is passed to the constructor of *CSVReader*, it is applied to the rows of the *CSVReader* as soon as they are read from the CSV file. See *example code* for further details.

Similarly, in the case where a Field Processor Object is passed to the constructor of *CSVWriter*, it is applied to the rows of the *CSVWriter* as soon as they are added for writing to the output CSV using its *add_rows()* method. See *example code* for further details.

Standalone Example Usage

Processor function definitions

```
def add1(x):
    return x + 1

def cast_to_int(x):
    return int(x)

def replace_big_huge(x):
    return x.replace("Big", "Huge")
```

Field processors and sample rows

```
from csvio.processors import FieldProcessor
from json import dumps

row1 = {
    "Supplier": "Big Apples",
    "Fruit": "Apple",
    "Origin": "Spain",
    "Quantity": "1"
}

row2 = {
    "Supplier": "Big Melons",
    "Fruit": "Melons",
    "Origin": "Italy",
    "Quantity": "2"
}

row3 = {
    "Supplier": "Long Mangoes",
    "Fruit": "Mango",
    "Origin": "India",
    "Quantity": "3"
}

rows = [row1, row2, row3]

proc1 = FieldProcessor('increment_qty')
proc1.add_processor("Quantity", cast_to_int)
proc1.add_processor("Quantity", add1)

proc2 = FieldProcessor('replace')
proc2.add_processor("Supplier", replace_big_huge)
```

Using implicit processor object

If a processor object or handle is not passed to the `process_row` method, the processor functions associated with the processor object whose `process_row` method we are calling are used implicitly.

```
print("Using implicit processor object:")
pretty_print("Before:", row1)
pretty_print("After:", proc1.process_row(row1)) # Using implicit processor object
```

Output

```
Using implicit processor object:
Before:
{
    "Supplier": "Big Apples",
    "Fruit": "Apple",
    "Origin": "Spain",
    "Quantity": "1"
}
```

(continues on next page)

(continued from previous page)

After:

```
{
  "Supplier": "Big Apples",
  "Fruit": "Apple",
  "Origin": "Spain",
  "Quantity": 2
}
```

Using processor handle

Any processor object can be used to apply the processors from another object by using the handle reference as shown below. We are using the handle 'replace' associated with the proc2 object, however we are using the proc1 object to apply the processor.

```
print("Using processor handle:")
pretty_print("Before:", rows)
pretty_print("After:", proc1.process_rows(rows, 'replace')) # Using processor handle
```

Output

Using processor handle:

Before:

```
[
  {
    "Supplier": "Big Apples",
    "Fruit": "Apple",
    "Origin": "Spain",
    "Quantity": "1"
  },
  {
    "Supplier": "Big Melons",
    "Fruit": "Melons",
    "Origin": "Italy",
    "Quantity": "2"
  },
  {
    "Supplier": "Long Mangoes",
    "Fruit": "Mango",
    "Origin": "India",
    "Quantity": "3"
  }
]
```

After:

```
[
  {
    "Supplier": "Huge Apples",
    "Fruit": "Apple",
    "Origin": "Spain",
    "Quantity": "1"
  },
  {
```

(continues on next page)

(continued from previous page)

```

    "Supplier": "Huge Melons",
    "Fruit": "Melons",
    "Origin": "Italy",
    "Quantity": "2"
  },
  {
    "Supplier": "Long Mangoes",
    "Fruit": "Mango",
    "Origin": "India",
    "Quantity": "3"
  }
]

```

Using explicit processor object

Similarly we can also pass any other processor object instead of a handle.

```

print("Using explicit processor object:")
pretty_print("Before:", rows)
pretty_print("After:", proc1.process_rows(rows, proc2)) # Using explicit processor object

```

Output

```

Using explicit processor object:
Before:
[
  {
    "Supplier": "Big Apples",
    "Fruit": "Apple",
    "Origin": "Spain",
    "Quantity": "1"
  },
  {
    "Supplier": "Big Melons",
    "Fruit": "Melons",
    "Origin": "Italy",
    "Quantity": "2"
  },
  {
    "Supplier": "Long Mangoes",
    "Fruit": "Mango",
    "Origin": "India",
    "Quantity": "3"
  }
]

After:
[
  {
    "Supplier": "Huge Apples",
    "Fruit": "Apple",
    "Origin": "Spain",
    "Quantity": "1"
  }
]

```

(continues on next page)

(continued from previous page)

```

    },
    {
        "Supplier": "Huge Melons",
        "Fruit": "Melons",
        "Origin": "Italy",
        "Quantity": "2"
    },
    {
        "Supplier": "Long Mangoes",
        "Fruit": "Mango",
        "Origin": "India",
        "Quantity": "3"
    }
]

```

class csvio.processors.field_processor.**FieldProcessor**(handle: str)

Bases: csvio.processors.processor_base.ProcessorBase

Parameters **handle** (*required*) – Reference handle for the field processor

add_processor(fieldname: str, func_: Union[List[Callable[[str], Any]], Callable[[str], Any]], handle: Optional[str] = None) → None

Add a processor to process fields in a row.

Parameters

- **fieldname** (*required*) – Name of the field upon which the processor should be executed.
- **func** (*required*) – Field processor function reference or a list of such function references. All function references added with the same handle will be executed for the field, to transform its value in the same order as they are added.
- **handle** (*optional*) – Processor reference handle to which the processor will be added. If not provided, the handle of the current object will be used.

Returns None

See [example code](#) for using with [CSVReader](#)

See [example code](#) for using with [CSVWriter](#)

process_row(row: Dict[str, Any], processor_handle: Optional[Union[Type[csvio.processors.processor_base.ProcessorBase], str]] = None) → Dict[str, Any]

Process a single row

Parameters

- **row** (*required*) – A single dictionary of fieldname->value pairs representing a single row
- **processor_handle** (*optional*) – A processor handle or an object that references the processor functions to apply and transform the row values. The processor functions of the current object are used if this argument is not provided.

Returns A dictionary representing a processed CSV row

process_rows(rows: List[Dict[str, Any]], processor_handle: Optional[str] = None) → List[Dict[str, Any]]

Process a list of rows

Parameters

- **row** (*required*) – A list of dictionaries of `fieldname->value` pairs representing a list of rows
- **processor_handle** (*optional*) – A processor handle or an object that references the processor functions to apply and transform the row values. The processor functions of the current object are used if this argument is not provided.

Returns A list of dictionaries representing processed CSV rows

- `genindex`
- `search`

7.2 CHANGE LOG

2021-11-01

Version 0.3.0

- Process and transform row values using field processors [read more](#)
- Update/Fix documentation
- Refactor/Add tests

2021-10-03

Version 0.2.0

- Construct a nested dictionary from rows based on a list of ordered column names [read more](#)
- Update/Fix documentation
- Refactor/Add tests

2021-09-24

Version 0.1.0

- First release

A

`add_processor()` (*csvio.processors.field_processor.FieldProcessor* method), 51

`add_rows()` (*csvio.CSVWriter* method), 37

C

`csv_kwargs` (*csvio.csvbase.CSVBase* property), 22

`csv_kwargs` (*csvio.CSVReader* property), 31

`csv_kwargs` (*csvio.CSVWriter* property), 40

CSVBase (class in *csvio.csvbase*), 22

CSVReader (class in *csvio*), 28

CSVWriter (class in *csvio*), 37

D

`delete()` (*csvio.csvbase.CSVBase* method), 22

`delete()` (*csvio.CSVReader* method), 31

`delete()` (*csvio.CSVWriter* method), 40

`delete()` (*csvio.filebase.FileBase* method), 21

F

`fieldnames` (*csvio.csvbase.CSVBase* property), 22

`fieldnames` (*csvio.CSVReader* property), 31

`fieldnames` (*csvio.CSVWriter* property), 40

FieldProcessor (class
csvio.processors.field_processor), 51

`file_ext` (*csvio.csvbase.CSVBase* property), 22

`file_ext` (*csvio.CSVReader* property), 31

`file_ext` (*csvio.CSVWriter* property), 40

`file_ext` (*csvio.filebase.FileBase* property), 21

FileBase (class in *csvio.filebase*), 21

`filedir` (*csvio.csvbase.CSVBase* property), 22

`filedir` (*csvio.CSVReader* property), 31

`filedir` (*csvio.CSVWriter* property), 40

`filedir` (*csvio.filebase.FileBase* property), 21

`filename` (*csvio.csvbase.CSVBase* property), 22

`filename` (*csvio.CSVReader* property), 31

`filename` (*csvio.CSVWriter* property), 40

`filename` (*csvio.filebase.FileBase* property), 21

`filename_no_ext` (*csvio.csvbase.CSVBase* property),
22

`filename_no_ext` (*csvio.CSVReader* property), 31

`filename_no_ext` (*csvio.CSVWriter* property), 41

`filename_no_ext` (*csvio.filebase.FileBase* property), 21

`filepath` (*csvio.csvbase.CSVBase* property), 22

`filepath` (*csvio.CSVReader* property), 31

`filepath` (*csvio.CSVWriter* property), 41

`filepath` (*csvio.filebase.FileBase* property), 21

`flush()` (*csvio.CSVWriter* method), 41

N

`num_rows` (*csvio.csvbase.CSVBase* property), 23

`num_rows` (*csvio.CSVReader* property), 31

`num_rows` (*csvio.CSVWriter* property), 41

O

`open_kwargs` (*csvio.csvbase.CSVBase* property), 23

`open_kwargs` (*csvio.CSVReader* property), 31

`open_kwargs` (*csvio.CSVWriter* property), 41

P

`path_obj` (*csvio.csvbase.CSVBase* property), 23

`path_obj` (*csvio.CSVReader* property), 31

`path_obj` (*csvio.CSVWriter* property), 41

`path_obj` (*csvio.filebase.FileBase* property), 21

`pending_rows` (*csvio.CSVWriter* property), 42

in `process_row()` (*csvio.processors.field_processor.FieldProcessor*
method), 51

`process_rows()` (*csvio.processors.field_processor.FieldProcessor*
method), 51

R

`rows` (*csvio.csvbase.CSVBase* property), 23

`rows` (*csvio.CSVReader* property), 31

`rows` (*csvio.CSVWriter* property), 42

`rows_from_column_key()` (*csvio.csvbase.CSVBase*
method), 23

`rows_from_column_key()` (*csvio.CSVReader* method),
31

`rows_from_column_key()` (*csvio.CSVWriter* method),
42

`rows_to_nested_dicts()` (*csvio.csvbase.CSVBase*
method), 23

`rows_to_nested_dicts()` (*csvio.CSVReader* method),
32

`rows_to_nested_dicts()` (*csvio.CSVWriter method*),
[42](#)

T

`touch()` (*csvio.csvbase.CSVBase method*), [28](#)

`touch()` (*csvio.CSVReader method*), [37](#)

`touch()` (*csvio.CSVWriter method*), [47](#)

`touch()` (*csvio.filebase.FileBase method*), [21](#)

W

`write_blank_csv()` (*csvio.CSVWriter method*), [47](#)