csvio

Release 1.0.0

Salman Raza

Nov 07, 2021

CONTENTS

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.

CHAPTER

ONE

INSTALLATION

```
pip install csvio
```

CHAPTER

TWO

DOCUMENTATION

Readthedocs

CHAPTER

THREE

READING CSVS

CSV file contents:

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

```
>>> 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"
   }
]
```

CHAPTER

FOUR

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
```

CHAPTER

FIVE

APPLY PROCESSORS TO TRANSFORM ROW VALUES

- Field Processor Standalone use
- Row Processor Standalone use

CHAPTER

SIX

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
}
```

CHAPTER

SEVEN

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 samee 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
```

(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*) → 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.

7.1.3 CSVReader

CSVReader use without processors:

```
>>> 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"
    },
    {
```

(continues on next page)

(continued from previous page)

```
        "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
```

CSV Reader with Processors

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
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 update_row(row):

    row["Supplier"] = f"{row['Supplier']} ({row['Origin']})"

    row["Quantity"] = int(row["Quantity"])

    if row["Quantity"] > 2:
        row["Quantity"] += 1

    return row
```

(continues on next page)

(continued from previous page)

```
def capitalize(x):
    return x.upper()

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

Define and apply processors

```
from csvio import CSVReader, CSVWriter
from csvio.processors import FieldProcessor, RowProcessor
from json import dumps

fp = FieldProcessor("fp1")

fp.add_processor("Supplier", replace_big_huge)
fp.add_processor("Fruit", capitalize)
fp.add_processor("Quantity", lambda x: int(x))
fp.add_processor("Origin", capitalize)

rp = RowProcessor("rp1")
rp.add_processor(update_row)

processors_list = [fp,rp]

reader = CSVReader("fruit_stock.csv", processors=processors_list)

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

print(dumps(reader.rows, indent=4))
```

Output

```
[
    {
        "Supplier": "Huge Apples (SPAIN)",
        "Fruit": "APPLE",
        "Origin": "SPAIN",
        "Quantity": 1
    },
    {
        "Supplier": "Huge Melons (ITALY)",
        "Fruit": "MELONS",
        "Origin": "ITALY",
        "Quantity": 2
    },
    {
        "Supplier": "Long Mangoes (INDIA)",
        "Fruit": "MANGO",
        "Origin": "INDIA",
        "Quantity": 4
```

(continues on next page)

(continued from previous page)

```
    },
    {
        "Supplier": "Small Strawberries (FRANCE)",
        "Fruit": "STRAWBERRY",
        "Origin": "FRANCE",
        "Quantity": 5
    },
    {
        "Supplier": "Short Mangoes (FRANCE)",
        "Fruit": "MANGO",
        "Origin": "FRANCE",
        "Quantity": 6
    },
    {
        "Supplier": "Sweet Strawberries (SPAIN)",
        "Fruit": "STRAWBERRY",
        "Origin": "SPAIN",
        "Quantity": 7
    },
    {
        "Supplier": "Square Apples (ITALY)",
        "Fruit": "APPLE",
        "Origin": "ITALY",
        "Quantity": 8
    },
    {
        "Supplier": "Small Melons (ITALY)",
        "Fruit": "MELONS",
        "Origin": "ITALY",
        "Quantity": 9
    },
    {
        "Supplier": "Dark Berries (AUSTRALIA)",
        "Fruit": "STRAWBERRY",
        "Origin": "AUSTRALIA",
        "Quantity": 10
    },
    {
        "Supplier": "Sweet Berries (AUSTRALIA)",
        "Fruit": "BLACKCURRANT",
        "Origin": "AUSTRALIA",
        "Quantity": 11
    }
]
```

CSV Contents after Processing: fruit_stock_processed.csv

```
Supplier,Fruit,Origin,Quantity
Huge Apples (SPAIN),APPLE,SPAIN,1
Huge Melons (ITALY),MELONS,ITALY,2
Long Mangoes (INDIA),MANGO,INDIA,4
Small Strawberries (FRANCE),STRAWBERRY,FRANCE,5
```

(continues on next page)

(continued from previous page)

```
Short Mangoes (FRANCE),MANGO,FRANCE,6
Sweet Strawberries (SPAIN),STRAWBERRY,SPAIN,7
Square Apples (ITALY),APPLE,ITALY,8
Small Melons (ITALY),MELONS,ITALY,9
Dark Berries (AUSTRALIA),STRAWBERRY,AUSTRALIA,10
Sweet Berries (AUSTRALIA),BLACKCURRANT,AUSTRALIA,11
```

class csvio.**CSVReader**(*filename: str*, *processors: Optional[List[csvio.processors.processor_base.ProcessorBase]] = 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.

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 samee 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"
        },
        {
```

(continues on next page)

(continued from previous page)


```
            "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": [
        {
            "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"
            },
            {
                "Supplier": "Small Melons",
                "Fruit": "Melons",
                "Origin": "Italy",
                "Quantity": "8"
            }
        ]
    },
    "Mango": {
        "India": [
            {
                "Supplier": "Long Mangoes",
                "Fruit": "Mango",
```

(continues on next page)

(continued from previous page)


```
                "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": {
        "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.

7.1.4 CSVWriter

CSVWriter use without processors:

```
>>> 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
```

(continues on next page)

(continued from previous page)

```
>>> 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()*. **CSV Writer with Processors**

Processor function definitions

```
def update_row(row):

    row["Supplier"] = f"{row['Supplier']} ({row['Origin']})"

    row["Quantity"] = int(row["Quantity"])

    if row["Quantity"] > 2:
        row["Quantity"] += 1

    return row

def capitalize(x):
    return x.upper()

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

Define and apply processors

```
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]

fp = FieldProcessor("fp1")

fp.add_processor("Supplier", replace_big_huge)
```

(continues on next page)

(continued from previous page)

```
fp.add_processor("Fruit", capitalize)
fp.add_processor("Quantity", lambda x: int(x))
fp.add_processor("Origin", capitalize)

rp = RowProcessor("rp1")
rp.add_processor(update_row)

processors_list = [fp,rp]

fieldnames = ["Supplier", "Fruit", "Origin", "Quantity"]

writer = CSVWriter(
    "fruit_stock_processed.csv", fieldnames, processors=processors_list
)

writer.add_rows(rows)
writer.flush()

print(dumps(writer.rows, indent=4))
```

Output

```
[
    {
        "Supplier": "Huge Apples (SPAIN)",
        "Fruit": "APPLE",
        "Origin": "SPAIN",
        "Quantity": 1
    },
    {
        "Supplier": "Huge Melons (ITALY)",
        "Fruit": "MELONS",
        "Origin": "ITALY",
        "Quantity": 2
    },
    {
        "Supplier": "Long Mangoes (INDIA)",
        "Fruit": "MANGO",
        "Origin": "INDIA",
        "Quantity": 4
    }
]
```

Contents of `fruit_stock_processed.csv`

```
Supplier,Fruit,Origin,Quantity
Huge Apples (SPAIN),APPLE,SPAIN,1
Huge Melons (ITALY),MELONS,ITALY,2
Long Mangoes (INDIA),MANGO,INDIA,4
```

class csvio.**CSVWriter**(*filename: str*, *fieldnames: List[str]*, *processors: Optional[List[csvio.processors.processor_base.ProcessorBase]] = 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 repsresent the row(s) to be written to the output CSV.

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 samee 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",
            "Origin": "France",
```

(continues on next page)

(continued from previous page)

```
            "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 Processors

Various types of processors are available in *csvio* that can be used to transform the values of rows in a CSV file.

Currently available processors:

Field Processor

A Field Processor is used to transform the values in a row represented by a dictionary that maps `column->value` pairs. This processor can be used to transform the values of particular fields in a row, where the values of other fields in the row are not required for making the transformation.

Row Processors can be used to make transformations where values of other fields within the same row are required.

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.

Field Processor 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"
}

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 be 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:
[
```

(continues on next page)

(continued from previous page)

```
    {
        "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:
[
    {
        "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": "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"
    }
]
```

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 function to process fields in a row.

The processor function reference is essentially a callback function that accepts a single argument that represents the value of the `fieldname` argument from the row upon which the processors will be executed.

The value of the `fieldname` argument from the row of a CSV is used within this callback function. This callback function should return a single value that should be set to the value of the `fieldname` once all the required transformations are applied.

Parameters

- **fieldname** (*required*) – Name of the field upon which the processor should be executed.
- **func** (*required*) – Field processor callback 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

This applies the processors defined using the *add_processor()* function in the same order that they were added using *add_processor()*

The output row after application of the previous processor function is passed on to the next processor function that was added using *add_processor()*, and the output of the last processor function added is returned as the final output of this function.

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

Row Processor

A Row Processor is used to transform the values of a row represented by a dictionary that maps `column->value` pairs. This processor is used in situations where you need to transform values of particular fields in a row depending upon the values of some other fields within the same row.

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 Row 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 Row 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 Row 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.

Row Processor Standalone use

Processor function definitions

```
def update_row(row):

    row["Supplier"] = f"{row['Supplier']} ({row['Origin']})"

    row["Quantity"] = int(row["Quantity"])

```

(continues on next page)

(continued from previous page)

```
    if row["Quantity"] > 2:
        row["Quantity"] += 1

    return row
```

Row processor and sample rows

```
from csvio.processors import RowProcessor
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]

rowproc = RowProcessor("rp1")

rowproc.add_processor(update_row)

processed_rows = rowproc.process_rows(rows)

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

print("After:")
print(dumps(processed_rows, indent=4))
```

Output

```
Before:
[
    {
        "Supplier": "Big Apples",
```

(continues on next page)

(continued from previous page)

```
        "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": "Big Apples (Spain)",
        "Fruit": "Apple",
        "Origin": "Spain",
        "Quantity": 1
    },
    {
        "Supplier": "Big Melons (Italy)",
        "Fruit": "Melons",
        "Origin": "Italy",
        "Quantity": 2
    },
    {
        "Supplier": "Long Mangoes (India)",
        "Fruit": "Mango",
        "Origin": "India",
        "Quantity": 4
    }
]
```

class csvio.processors.row_processor.**RowProcessor**(*handle: str*)

Bases: csvio.processors.processor_base.ProcessorBase

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

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

Add a processor function to process rows.

The processor function reference is essentially a callback function that accepts a single argument that represents a row.

The `fieldnames` from a CSV can be used within this callback function as the keys to this single argument that represents a row to access its values and perform the required transformations. This callback function should return a dictionary representing a row, once all the required transformations are applied.

Parameters

- **func** (*required*) – Row processor callback function reference or a list of such function references. All function references added with the same handle will be executed for the row, to transform its value in the same order as they are added. A single processor function will be sufficient to perform all the transformations for the rows in a CSV, if it has all the transformation operations required in its definition.
- **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.

This applies the processors defined using the `add_processor()` function in the same order that they were added using `add_processor()`

The output row after application of the previous processor function is passed on to the next processor function that was added using `add_processor()`, and the output of the last processor function added is returned as the final output of this function.

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

2021-11-07

Version 1.0.0

- Process and transform row values using row processors read more.
- Define and pass multiple types of processors to constructors of CSVReader and CSVWriter.
- **Breaking update**: `fieldprocessor` argument to constructors of CSVWriter and CSVReader renamed to `processors`, that takes accepts a list of processors.
- Update/Fix documentation.
- Refactor/Add tests.

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. names read more.
- Update/Fix documentation.
- Refactor/Add tests.

2021-09-24

Version 0.1.0

- First release.

INDEX

T

W