# styleframe

unknown

Nov 14, 2023

# CONTENTS

A library that wraps pandas and openpyxl and allows easy styling of dataframes.

Contents:

CHAPTER

# ONE

# INSTALLATION AND TESTING

```
$ pip install styleframe
```

To make sure everything works as expected, run styleframe's unittests:

```
from styleframe import tests

tests.run()
```

CHAPTER

TWO

# BASIC USAGE EXAMPLES

StyleFrame's `init` supports all the ways you are used to initiate pandas dataframe. An existing dataframe, a dictionary or a list of dictionaries:

```
from styleframe import StyleFrame, Styler, utils

sf = StyleFrame({'col_a': range(100)})
```

Applying a style to rows that meet a condition using pandas selecting syntax. In this example all the cells in the *col_a* column with the value > 50 will have blue background and a bold, sized 10 font:

```
sf.apply_style_by_indexes(
    indexes_to_style=sf[sf['col_a'] > 50],
    cols_to_style=['col_a'],
    styler_obj=Styler(
        bg_color=utils.colors.blue,
        bold=True,
        font_size=10
    )
)
```

Creating ExcelWriter object:

```
ew = StyleFrame.ExcelWriter(r'C:\my_excel.xlsx')
sf.to_excel(ew)
ew.close()
```

It is also possible to style a whole column or columns, and decide whether to style the headers or not:

```
sf.apply_column_style(
    cols_to_style=['a'],
    styler_obj=Styler(bg_color=utils.colors.green),
    style_header=True
)
```

## 2.1 Accessors

### 2.1.1 .style

Combined with *.loc*, allows easy selection/indexing based on style. For example:

```
only_rows_with_yellow_bg_color = sf.loc[sf['col_name'].style.bg_color == utils.colors.
↪yellow]
only_rows_with_non_bold_text = sf.loc[~sf['col_name'].style.bold]
```

CHAPTER

# THREE

# API DOCUMENTATION

## 3.1 styleframe

The *styleframe* module contains a single class *StyleFrame* which servers as the main interaction point.

**class StyleFrame**(*obj*, *styler_obj:* Styler | *None* = *None*, *columns: List[str]* | *None* = *None*)

A wrapper class that wraps a `pandas.DataFrame` object and represent a stylized dataframe. Stores container objects that have values and styles that will be applied to excel

**Parameters**

- **obj** – Any object that pandas' dataframe can be initialized with: an existing dataframe, a dictionary, a list of dictionaries or another StyleFrame.
- **styler_obj** (*Styler*) – Will be used as the default style of all cells.
- **columns** (*None or list[str]*) – Names of columns to use. Only applicable if `obj` is `numpy.ndarray`

**classmethod ExcelWriter**(*path*, ***kwargs*)

A shortcut for `pandas.ExcelWriter`, and accepts any argument it accepts except for `engine`

**add_color_scale_conditional_formatting**(*start_type: str*, *start_value: int* | *float*, *start_color: str*, *end_type: str*, *end_value: int* | *float*, *end_color: str*, *mid_type: str* | *None* = *None*, *mid_value: int* | *float* | *None* = *None*, *mid_color: str* | *None* = *None*, *columns_range=None*)

**Parameters**

- **start_type** (str: one of *utils.conditional_formatting_types* or any other type Excel supports) – The type for the minimum bound
- **start_value** – The threshold for the minimum bound
- **start_color** (str: one of *utils.colors*, hex string or color name ie *'yellow'* Excel supports) – The color for the minimum bound
- **end_type** (str: one of *utils.conditional_formatting_types* or any other type Excel supports) – The type for the maximum bound
- **end_value** – The threshold for the maximum bound
- **end_color** (str: one of *utils.colors*, hex string or color name ie *'yellow'* Excel supports) – The color for the maximum bound
- **mid_type** (None or str: one of *utils.conditional_formatting_types* or any other type Excel supports) – The type for the middle bound

- **mid_value** – The threshold for the middle bound
- **mid_color** (None or str: one of *utils.colors*, hex string or color name ie *'yellow'* Excel supports) – The color for the middle bound
- **columns_range** (*None or list[str or int] or tuple[str or int]*) – A two-elements list or tuple of columns to which the conditional formatting will be added to. If not provided at all the conditional formatting will be added to all columns. If a single element is provided then the conditional formatting will be added to the provided column. If two elements are provided then the conditional formatting will start in the first column and end in the second. The provided columns can be a column name, letter or index.

**Returns**
self

**Return type**
*StyleFrame*

**apply_column_style**(*cols_to_style: str | List[str] | Tuple[str] | Set[str]*, *styler_obj: Styler*, *style_header: bool = False*, *use_default_formats: bool = True*, *width: int | float | None = None*, *overwrite_default_style: bool = True*)

Apply style to a whole column

**Parameters**

- **cols_to_style** (*str or list or tuple or set*) – The column names to style.
- **styler_obj** (*Styler*) – A *Styler* object.
- **style_header** (*bool*) – If `True`, the column(s) header will also be styled.
- **use_default_formats** (*bool*) – If `True`, the default formats for date and times will be used.
- **width** (*None or int or float*) – If provided, the new width for the specified columns.
- **overwrite_default_style** (*bool*) – (bool) If `True`, the default style (the style used when initializing StyleFrame) will be overwritten. If `False` then the default style and the provided style wil be combined using *Styler.combine()* method.

**Returns**
self

**Return type**
*StyleFrame*

**apply_headers_style**(*styler_obj: Styler*, *style_index_header: bool = True*, *cols_to_style: str | List[str] | Tuple[str] | Set[str] | None = None*)

Apply style to the headers only

**Parameters**
**styler_obj** (*Styler*) – The style to apply

New in version 1.6.1.

**Parameters**
**style_index_header** (*bool*) – If True then the style will also be applied to the header of the index column

New in version 2.0.5.

**Parameters**
**cols_to_style** (*None or str or list[str] or tuple[str] or set[str]*) – the columns to apply the style to, if not provided all the columns will be styled

**Returns**
self

**Return type**
*StyleFrame*

**apply_style_by_indexes**(*indexes_to_style: list | tuple | int | Container, styler_obj: Styler, cols_to_style: str | List[str] | Tuple[str] | Set[str] | None = None, height: int | float | None = None, complement_style: Styler | None = None, complement_height: int | float | None = None, overwrite_default_style: bool = True*)

Applies a certain style to the provided indexes in the dataframe in the provided columns

**Parameters**

- **indexes_to_style** (*list or tuple or int or Container*) – Indexes to which the provided style will be applied. Usually passed as pandas selecting syntax. For example,

```
sf[sf['some_col'] == 20]
```

- **styler_obj** (*Styler*) – *Styler* object that contains the style that will be applied to indexes in *indexes_to_style*
- **cols_to_style** (*None or str or list[str] or tuple[str] or set[str]*) – The column names to apply the provided style to. If `None` all columns will be styled.
- **height** (*None or int or float*) – If provided, set height for rows whose indexes are in *indexes_to_style*.

New in version 1.5.

**Parameters**

- **complement_style** (None or *Styler*) – *Styler* object that contains the style which will be applied to indexes not in *indexes_to_style*
- **complement_height** (*None or int or float*) – Height for rows whose indexes are not in *indexes_to_style*. If not provided then *height* will be used (if provided).

New in version 1.6.

**Parameters**
**overwrite_default_style** (*bool*) – If `True`, the default style (the style used when initializing StyleFrame) will be overwritten. If `False` then the default style and the provided style wil be combined using *Styler.combine()* method.

**Returns**
self

**Return type**
*StyleFrame*

**classmethod read_excel**(*path: str, sheet_name: str | int = 0, read_style: bool = False, use_openpyxl_styles: bool = False, read_comments: bool = False, **kwargs*) → *StyleFrame*

Creates a StyleFrame object from an existing Excel.

**Note:** *read_excel()* also accepts all arguments that pandas.read_excel() accepts as kwargs.

**Parameters**

- **path** (*str*) – The path to the Excel file to read.
- **sheet_name** (*str or int*) – The sheet name to read. If an integer is provided then it be used as a zero-based sheet index. Default is 0.
- **read_style** (*bool*) – If True the sheet's style will be loaded to the returned StyleFrame object.
- **use_openpyxl_styles** (*bool*) – If True (and *read_style* is also True) then the styles in the returned StyleFrame object will be Openpyxl's style objects. If False, the styles will be *Styler* objects.

  **Note:** Using use_openpyxl_styles=False is useful if you are going to filter columns or rows by style, for example:

  ```
  sf = sf[[col for col in sf.columns if col.style.font == utils.fonts.
  ↪arial]]
  ```

- **read_comments** (*bool*) – If True (and *read_style* is also True) cells' comments will be loaded to the returned StyleFrame object. Note that reading comments without reading styles is currently not supported.

**Returns**
StyleFrame object

**Return type**
*StyleFrame*

**classmethod read_excel_as_template**(*path: str*, *df: DataFrame*, *use_df_boundaries: bool = False*, ***kwargs*) → *StyleFrame*

New in version 3.0.1.

Create a StyleFrame object from an excel template with data of the given DataFrame.

**Note:** *read_excel_as_template()* also accepts all arguments that *read_excel()* accepts as kwargs except for read_style which must be True.

**Parameters**

- **path** (*str*) – The path to the Excel file to read.
- **df** (pandas.DataFrame) – The data to apply to the given template.
- **use_df_boundaries** (*bool*) – If True the template will be cut according to the boundaries of the given DataFrame.

**Returns**
StyleFrame object

**Return type**
*StyleFrame*

**rename**(*columns=None*, *inplace=False*)

Renames the underlying dataframe's columns

**Parameters**

- **columns** (*dict*) – A dictionary from old columns names to new columns names.
- **inplace** (*bool*) – If `False`, a new StyleFrame object will be returned. If `True`, renames the columns inplace.

**Returns**
self if inplace is `True`, new StyleFrame object is `False`

**Return type**
*StyleFrame*

**set_column_width**(*columns: str | List[str] | Tuple[str] | List[int] | Tuple[int]*, *width: int | float*) → *StyleFrame*

Set the width of the given columns

**Parameters**

- **columns** (*str or list[str] or tuple[str] or int or list[int] or tuple[int]*) – Column name(s) or index(es).
- **width** (*int or float*) – The new width for the specified columns.

**Returns**
self

**Return type**
*StyleFrame*

**set_column_width_dict**(*col_width_dict: Dict[str, int | float]*) → *StyleFrame*

**Parameters**
**col_width_dict** (*dict[str, int or float]*) – A dictionary from column names to width.

**Returns**
self

**Return type**
*StyleFrame*

**set_row_height**(*rows: int | List[int] | Tuple[int] | Set[int]*, *height: int | float*) → *StyleFrame*

Set the height of the given rows

**Parameters**

- **rows** (*int or list[int] or tuple[int] or set[int]*) – Row(s) index.
- **height** (*int or float*) – The new height for the specified indexes.

**Returns**
self

**Return type**
*StyleFrame*

**set_row_height_dict**(*row_height_dict: Dict[int, int | float]*) → *StyleFrame*

**Parameters**
**row_height_dict** (*dict[int, int or float]*) – A dictionary from row indexes to height.

**Returns**
self

**Return type**
*StyleFrame*

**style_alternate_rows**(*styles: List[*Styler*] | Tuple[*Styler*], **kwargs*) → *StyleFrame*

New in version 1.2.

Applies the provided styles to rows in an alternating manner.

---

**Note:** *style_alternate_rows()* also accepts all arguments that *apply_style_by_indexes()* accepts as kwargs.

---

**Parameters**
**styles** (list[*Styler*] or tuple[*Styler*]) – List or tuple of *Styler* objects to be applied to rows in an alternating manner

**Returns**
self

**Return type**
*StyleFrame*

**to_excel**(*excel_writer: str | ExcelWriter | Path = 'output.xlsx', sheet_name: str = 'Sheet1', allow_protection: bool = False, right_to_left: bool = False, columns_to_hide: None | str | list | tuple | set = None, row_to_add_filters: int | None = None, columns_and_rows_to_freeze: str | None = None, best_fit: None | str | list | tuple | set = None, index: bool = False, **kwargs*) → ExcelWriter

Saves the dataframe to excel and applies the styles.

---

**Note:** *to_excel()* also accepts all arguments that `pandas.DataFrame.to_excel()` accepts as kwargs.

---

**Parameters**

- **excel_writer** (str or `pandas.ExcelWriter` or `pathlib.Path`) – File path or existing ExcelWriter
- **sheet_name** (*str*) – Name of sheet the StyleFrame will be exported to
- **allow_protection** (*bool*) – Allow to protect the cells that specified as protected. If used `protection=True` in a Styler object this must be set to `True`.
- **right_to_left** (*bool*) – Makes the sheet right-to-left.
- **columns_to_hide** (*None or str or list or tuple or set*) – Columns names to hide.
- **row_to_add_filters** (*None or int*) – Add filters to the given row index, starts from 0 (which will add filters to header row).
- **columns_and_rows_to_freeze** (*None or str*) – Column and row string to freeze. For example "C3" will freeze columns: A, B and rows: 1, 2.

New in version 1.4.

**Parameters**

**best_fit** (*None or str or list or tuple or set*) – single column, list, set or tuple of columns names to attempt to best fit the width for.

---

**Note:** `best_fit` will attempt to calculate the correct column-width based on the longest value in each provided column. However this isn't guaranteed to work for all fonts (works best with monospaced fonts). The formula used to calculate a column's width is equivalent to

```
(len(longest_value_in_column) + A_FACTOR) * P_FACTOR
```

The default values for `A_FACTOR` and `P_FACTOR` are 13 and 1.3 respectively, and can be modified before calling `StyleFrame.to_excel` by directly modifying `StyleFrame.A_FACTOR` and `StyleFrame.P_FACTOR`

---

New in version 4.2.

**Parameters**

**index** (*bool*) – Write row names.

**Return type**

pandas.ExcelWriter

## 3.2 styler

**class Styler**(*bg_color: str | None = None, bold: bool = False, font: str = 'Arial', font_size: int | float = 12.0, font_color: str | None = None, number_format: str = 'General', protection: bool = False, underline: str | None = None, border_type: str | Set[str] | Dict[str, str] = 'thin', horizontal_alignment: str = 'center', vertical_alignment: str = 'center', wrap_text: bool = True, shrink_to_fit: bool = True, fill_pattern_type: str = 'solid', indent: int | float = 0.0, comment_author: str | None = None, comment_text: str | None = None, text_rotation: int = 0, date_format: str = 'DD/MM/YY', time_format: str = 'HH:MM', date_time_format: str = 'DD/MM/YY HH:MM', strikethrough: bool = False, italic: bool = False*)

Used to represent a style

**Parameters**

- **bg_color** (str: one of *utils.colors*, hex string or color name ie *'yellow'* Excel supports) – The background color
- **bold** (*bool*) – If `True`, a bold typeface is used
- **font** (str: one of *utils.fonts* or other font name Excel supports) – The font to use
- **font_size** (*int*) – The font size
- **font_color** (str: one of *utils.colors*, hex string or color name ie *'yellow'* Excel supports) – The font color
- **number_format** (str: one of *utils.number_formats* or any other format Excel supports) – The format of the cell's value
- **protection** (*bool*) – If `True`, the cell/column will be write-protected

- **underline** (str: one of *utils.underline* or any other underline Excel supports) – The underline type

Changed in version 4.2.

**Parameters**

**border_type** (*str or set[str] or dict[str, str]*) –

- If provided a string (one of *utils.borders* or any other border type Excel supports): all borders will be set to that type.
- If provided a set of strings (*utils.border_locations* or any other border location Excel supports): each provided border will be set to the default border type.
- If provided a dict (from location, one of *utils.border_locations* or any other border location Excel supports) to border type (one of *utils.borders* or any other border type Excel supports): each provided border will be set to the provided border type.

New in version 1.2.

**Parameters**

- **horizontal_alignment** (str: one of *utils.horizontal_alignments* or any other horizontal alignment Excel supports) – Text's horizontal alignment
- **vertical_alignment** (str: one of *utils.vertical_alignments* or any other vertical alignment Excel supports) – Text's vertical alignment

New in version 1.3.

**Parameters**

- **wrap_text** (*bool*) –
- **shrink_to_fit** (*bool*) –
- **fill_pattern_type** (str: one of *utils.fill_pattern_types* or any other fill pattern type Excel supports) – Cells's fill pattern type
- **indent** (*int*) –
- **comment_author** (*str*) –
- **comment_text** (*str*) –
- **text_rotation** (*int*) – Integer in the range 0 - 180

New in version 4.0.

**Parameters**

- **date_format** (str: one of *utils.number_formats* or any other format Excel supports) –
- **time_format** (str: one of *utils.number_formats* or any other format Excel supports) –
- **date_time_format** (str: one of *utils.number_formats* or any other format Excel supports) –

---

**Note:** For any of `date_format`, `time_format` and `date_time_format` to take effect, the value being styled must be an actual `date`/`time`/`datetime` object.

---

New in version 4.1.

**Parameters**

- **strikethrough** (*bool*) –
- **italic** (*bool*) –

**classmethod combine**(**styles:* Styler)

New in version 1.6.

Used to combine *Styler* objects. The right-most object has precedence. For example:

```
Styler.combine(Styler(bg_color='yellow', font_size=24), Styler(bg_color='blue'))
```

will return

```
Styler(bg_color='blue', font_size=24)
```

**Parameters**
**styles** (*list or tuple or set*) – Iterable of Styler objects

**Returns**
self

**Return type**
*Styler*

## 3.3 utils

The *utils* module contains the most widely used values for styling elements such as colors and border types for convenience. It is possible to directly use a value that is not present in the utils module as long as the spreadsheet software recognises it.

**class number_formats**

**Variables**

- **general** (*str*) – 'General'
- **general_integer** (*str*) – '0'
- **general_float** (*str*) – '0.00'
- **percent** (*str*) – '0.0%'
- **thousands_comma_sep** (*str*) – '#,##0'
- **date** (*str*) – 'DD/MM/YY'
- **time_24_hours** (*str*) – 'HH:MM'
- **time_24_hours_with_seconds** (*str*) – 'HH:MM:SS'
- **time_12_hours** (*str*) – 'h:MM AM/PM'
- **time_12_hours_with_seconds** (*str*) – 'h:MM:SS AM/PM'
- **date_time** (*str*) – 'DD/MM/YY HH:MM'
- **date_time_with_seconds** (*str*) – 'DD/MM/YY HH:MM:SS'

**static decimal_with_num_of_digits**(*num_of_digits: int*) → str

New in version 1.6.

**Parameters**
**num_of_digits** (*int*) – Number of digits after the decimal point

**Returns**

A format string that represents a floating point number with the provided number of digits after the decimal point.

For example, `utils.number_formats.decimal_with_num_of_digits(2)` will return `'0.00'`

**Return type**
str

**class colors**

**Variables**

- **white** (*str*) – '00FFFFFF'
- **blue** (*str*) – '000000FF'
- **dark_blue** (*str*) – '00000080'
- **yellow** (*str*) – '00FFFF00'
- **dark_yellow** (*str*) – '00808000'
- **green** (*str*) – '0000FF00'
- **dark_green** (*str*) – '00008000'
- **black** (*str*) – '00000000'
- **red** (*str*) – '00FF0000'
- **dark_red** (*str*) – '00800000'
- **purple** (*str*) – '800080'
- **grey** (*str*) – 'D3D3D3'

**class fonts**

New in version 1.1.

**Variables**

- **aegean** (*str*) – 'Aegean'
- **aegyptus** (*str*) – 'Aegyptus'
- **aharoni** (*str*) – 'Aharoni CLM'
- **anaktoria** (*str*) – 'Anaktoria'
- **analecta** (*str*) – 'Analecta'
- **anatolian** (*str*) – 'Anatolian'
- **arial** (*str*) – 'Arial'
- **calibri** (*str*) – 'Calibri'
- **david** (*str*) – 'David CLM'
- **dejavu_sans** (*str*) – 'DejaVu Sans'

- **ellinia** (*str*) – 'Ellinia CLM'

class borders

**Variables**

- **dash_dot** (*str*) – 'dashDot'
- **dash_dot_dot** (*str*) – 'dashDotDot'
- **dashed** (*str*) – 'dashed'
- **default_grid** (*str*) – 'default_grid'
- **dotted** (*str*) – 'dotted'
- **double** (*str*) – 'double'
- **hair** (*str*) – 'hair'
- **medium** (*str*) – 'medium'
- **medium_dash_dot** (*str*) – 'mediumDashDot'
- **medium_dash_dot_dot** (*str*) – 'mediumDashDotDot'
- **medium_dashed** (*str*) – 'mediumDashed'
- **slant_dash_dot** (*str*) – 'slantDashDot'
- **thick** (*str*) – 'thick'
- **thin** (*str*) – 'thin'

class border_locations

New in version 4.2.

**Variables**

- **top** (*str*) – 'top'
- **bottom** (*str*) – 'bottom'
- **left** (*str*) – 'left'
- **right** (*str*) – 'right'

class horizontal_alignments

**Variables**

- **general** (*str*) – 'general'
- **left** (*str*) – 'left'
- **center** (*str*) – 'center'
- **right** (*str*) – 'right'
- **fill** (*str*) – 'fill'
- **justify** (*str*) – 'justify'
- **center_continuous** (*str*) – 'centerContinuous'
- **distributed** (*str*) – 'distributed'

class vertical_alignments

**Variables**

- **top** (*str*) – 'top'
- **center** (*str*) – 'center'
- **bottom** (*str*) – 'bottom'
- **justify** (*str*) – 'justify'
- **distributed** (*str*) – 'distributed'

class underline

**Variables**

- **single** (*str*) – 'single'
- **double** (*str*) – 'double'

class fill_pattern_types

New in version 1.2.

**Variables**

- **solid** (*str*) – 'solid'
- **dark_down** (*str*) – 'darkDown'
- **dark_gray** (*str*) – 'darkGray'
- **dark_grid** (*str*) – 'darkGrid'
- **dark_horizontal** (*str*) – 'darkHorizontal'
- **dark_trellis** (*str*) – 'darkTrellis'
- **dark_up** (*str*) – 'darkUp'
- **dark_vertical** (*str*) – 'darkVertical'
- **gray0625** (*str*) – 'gray0625'
- **gray125** (*str*) – 'gray125'
- **light_down** (*str*) – 'lightDown'
- **light_gray** (*str*) – 'lightGray'
- **light_grid** (*str*) – 'lightGrid'
- **light_horizontal** (*str*) – 'lightHorizontal'
- **light_trellis** (*str*) – 'lightTrellis'
- **light_up** (*str*) – 'lightUp'
- **light_vertical** (*str*) – 'lightVertical'
- **medium_gray** (*str*) – 'mediumGray'

class conditional_formatting_types

**Variables**

- **num** (*str*) – 'num'
- **percent** (*str*) – 'percent'

- **max** (*str*) – 'max'
- **min** (*str*) – 'min'
- **formula** (*str*) – 'formula'
- **percentile** (*str*) – 'percentile'

CHAPTER

FOUR

# COMMANDLINE INTERFACE

## 4.1 General Information

Starting with version 1.1 styleframe offers a commandline interface that lets you create an xlsx file from a json file.

## 4.2 Usage

| Flag | Explanation |
|---|---|
| `-v` | Displays the installed versions of styleframe and its dependencies |
| `--json_path/--json-path` | Path to the json file |
| `--json` | The json string which defines the Excel file, see example below |
| `--output_path/--output-path` | Path to the output xlsx file. If not provided defaults to `output.xlsx` |
| `--test` | Execute the tests |

### 4.2.1 Usage Examples

```
$ styleframe --json_path data.json --output_path data.xlsx
```

```
$ styleframe --json "[{\"sheet_name\": \"sheet_1\", \"columns\": [{\"col_name\": \"col_a\
↪", \"cells\": [{\"value\": 1}]}]}]"
```

**Note:** You may need to use different syntax to pass a JSON string depending on your OS and terminal application.

## 4.3 JSON Format

The input JSON should be thought of as an hierarchy of predefined entities, some of which correspond to a Python class used by StyleFrame. The top-most level should be a list of `sheet` entities (see below).

The provided JSON is validated against the following schema:

```
{
    "$schema": "http://json-schema.org/draft-04/schema#",
    "title": "sheets",
    "definitions": {
        "Sheet": {
            "$id": "#sheet",
            "title": "sheet",
            "type": "object",
            "properties": {
                "sheet_name": {
                    "type": "string"
                },
                "columns": {
                    "type": "array",
                    "items": {
                        "$ref": "#/definitions/Column"
                    },
                    "minItems": 1
                },
                "row_heights": {
                    "type": "object"
                },
                "extra_features": {
                    "type": "object"
                },
                "default_styles": {
                    "type": "object",
                    "properties": {
                        "headers": {
                            "$ref": "#/definitions/Style"
                        },
                        "cells": {
                            "$ref": "#/definitions/Style"
                        }
                    },
                    "additionalProperties": false
                }
            },
            "required": [
                "sheet_name",
                "columns"
            ]
        },
        "Column": {
            "$id": "#column",
            "title": "column",
            "type": "object",
            "properties": {
                "col_name": {
                    "type": "string"
                },
                "style": {
                    "$ref": "#/definitions/Style"
```

(continues on next page)

(continued from previous page)

```
                },
                "width": {
                    "type": "number"
                },
                "cells": {
                    "type": "array",
                    "items": {
                        "$ref": "#/definitions/Cell"
                    }
                }
            },
            "required": [
                "col_name",
                "cells"
            ]
        },
        "Cell": {
            "$id": "#cell",
            "title": "cell",
            "type": "object",
            "properties": {
                "value": {},
                "style": {
                    "$ref": "#/definitions/Style"
                }
            },
            "required": [
                "value"
            ],
            "additionalProperties": false
        },
        "Style": {
            "$id": "#style",
            "title": "style",
            "type": "object",
            "properties": {
                "bg_color": {
                    "type": "string"
                },
                "bold": {
                    "type": "boolean"
                },
                "font": {
                    "type": "string"
                },
                "font_size": {
                    "type": "number"
                },
                "font_color": {
                    "type": "string"
                },
                "number_format": {
```

(continues on next page)

(continued from previous page)

```
                    "type": "string"
                },
                "protection": {
                    "type": "boolean"
                },
                "underline": {
                    "type": "string"
                },
                "border_type": {
                    "type": "string"
                },
                "horizontal_alignment": {
                    "type": "string"
                },
                "vertical_alignment": {
                    "type": "string"
                },
                "wrap_text": {
                    "type": "boolean"
                },
                "shrink_to_fit": {
                    "type": "boolean"
                },
                "fill_pattern_type": {
                    "type": "string"
                },
                "indent": {
                    "type": "number"
                }
            },
            "additionalProperties": false
        }
    },
    "type": "array",
    "items": {
        "$ref": "#/definitions/Sheet"
    },
    "minItems": 1
}
```

An example JSON:

```
[
  {
    "sheet_name": "Sheet1",
    "default_styles": {
      "headers": {
        "font_size": 17,
        "bg_color": "yellow"
      },
      "cells": {
        "bg_color": "red"
```

(continues on next page)

(continued from previous page)

```
      }
    },
    "columns": [
      {
        "col_name": "col_a",
        "style": {"bg_color": "blue", "font_color": "yellow"},
        "width": 30,
        "cells": [
          {
            "value": 1
          },
          {
            "value": 2,
            "style": {
              "bold": true,
              "font": "Arial",
              "font_size": 30,
              "font_color": "green",
              "border_type": "double"
            }
          }
        ]
      },
      {
        "col_name": "col_b",
        "cells": [
          {
            "value": 3
          },
          {
            "value": 4,
            "style": {
              "bold": true,
              "font": "Arial",
              "font_size": 16
            }
          }
        ]
      }
    ],
    "row_heights": {
      "3": 40
    },
    "extra_features": {
      "row_to_add_filters": 0,
      "columns_and_rows_to_freeze": "A7",
      "startrow": 5
    }
  }
]
```

### 4.3.1 style

Corresponds to *Styler* class.

This entity uses the arguments of `Styler.__init__()` as keys. Any missing keys in the JSON will be given the same default values.

```
"style": {"bg_color": "yellow", "bold": true}
```

### 4.3.2 cell

This entity represents a single cell in the sheet.

Required keys:

`"value"` - The cell's value.

Optional keys:

`"style"` - The `style` entity for this cell. If not provided, the `style` provided to the `coloumn` entity will be used. If that was not provided as well, the default `Styler.__init__()` values will be used.

```
{"value": 42, "style": {"border": "double"}}
```

### 4.3.3 column

This entity represents a column in the sheet.

Required keys:

`"col_name"` - The column name.

`"cells"` - A list of `cell` entities.

Optional keys:

`"style"` - A style used for the entire column. If not provided the default `Styler.__init__()` values will be used.

`"width"` - The column's width. If not provided Excel's default column width will be used.

### 4.3.4 sheet

This entity represents the entire sheet.

Required keys:

`"sheet_name"` - The sheet's name.

`"columns"` - A list of `column` entities.

Optional keys:

`"default_styles"` - A JSON object with items as keys and `style` entities as values. Currently supported items: `headers` and `cells`.

```
"default_styles": {"headers": {"bg_color": "blue"}}
```

`"row_heights"` - A JSON object with rows indexes as keys and heights as value.

`"extra_features"` - A JSON that contains the same arguments as the `to_excel` method, such as `"row_to_add_filters"`, `"columns_and_rows_to_freeze"`, `"columns_to_hide"`, `"right_to_left"` and `"allow_protection"`. You can also use other arguments that Pandas' `"to_excel"` accepts.

# INDEX