gspread-formatting Documentation

Robin Thomas

Dec 03, 2022

Contents:

CHAPTER 1

gspread-formatting

This package provides complete cell formatting for Google spreadsheets using the popular `gspread` package, along with a few related features such as setting "frozen" rows and columns in a worksheet. Both basic and conditional formatting operations are supported.

The package also offers graceful formatting of Google spreadsheets using a Pandas DataFrame. See the section below for usage and details.

1.1 Usage

Basic formatting of a range of cells in a worksheet is offered by the `format_cell_range` function. All basic formatting components of the v4 Sheets API's `CellFormat` are present as classes in the `gspread_formatting` module, available both by `InitialCaps` names and `camelCase` names: for example, the background color class is `BackgroundColor` but is also available as `backgroundColor`, while the color class is `Color` but available also as `color`. Attributes of formatting components are best specified as keyword arguments using `camelCase` naming, e.g. `backgroundColor=...`. Complex formats may be composed easily, by nesting the calls to the classes.

See the CellFormat page of the Sheets API documentation to learn more about each formatting component.:

```
from gspread_formatting import *

fmt = cellFormat(
    backgroundColor=color(1, 0.9, 0.9),
    textFormat=textFormat(bold=True, foregroundColor=color(1, 0, 1)),
    horizontalAlignment='CENTER'
    )
```

(continues on next page)

(continued from previous page)

```
format_cell_range(worksheet, 'A1:J1', fmt)
```

The `format_cell_ranges` function allows for formatting multiple ranges with corresponding formats, all in one function call and Sheets API operation:

```
fmt = cellFormat(
    backgroundColor=color(1, 0.9, 0.9),
    textFormat=textFormat(bold=True, foregroundColor=color(1, 0, 1)),
    horizontalAlignment='CENTER'
    )

fmt2 = cellFormat(
    backgroundColor=color(0.9, 0.9, 0.9),
    horizontalAlignment='RIGHT'
    )

format_cell_ranges(worksheet, [('A1:J1', fmt), ('K1:K200', fmt2)])
```

1.2 Specifying Cell Ranges

The *format_cell_range* function and friends allow a string to specify a cell range using the "A1" convention to name a column-and-row cell address with column letter and row number; in addition, one may specify an entire column or column range with unbounded rows, or an entire row or row range with unbounded columns, or a combination thereof. Here are some examples:

```
A1      # column A row 1
A1:A2   # column A, rows 1-2
A       # entire column A, rows unbounded
A:A     # entire column A, rows unbounded
A:C     # entire columns A through C
A:B100  # columns A and B, unbounded start through row 100
A100:B  # columns A and B, from row 100 with unbounded end
1:3     # entire rows 1 through 3, all columns
1       # entire row 1
```

1.3 Retrieving, Comparing, and Composing CellFormats

A Google spreadsheet's own default format, as a CellFormat object, is available via `get_default_format(spreadsheet)`. `get_effective_format(worksheet, label)` and `get_user_entered_format(worksheet, label)` also will return for any provided cell label either a CellFormat object (if any formatting is present) or None.

`CellFormat` objects are comparable with == and `!=`, and are mutable at all times; they can be safely copied with Python's `copy.deepcopy` function. `CellFormat` objects can be combined into a new `CellFormat` object using the `add` method (or + operator). `CellFormat` objects also offer `difference` and `intersection` methods, as well as the corresponding operators − (for difference) and `&` (for intersection).:

```
>>> default_format = CellFormat(backgroundColor=color(1,1,1),␣
↪textFormat=textFormat(bold=True))
>>> user_format = CellFormat(textFormat=textFormat(italic=True))
```

(continues on next page)

(continued from previous page)

```
>>> effective_format = default_format + user_format
>>> effective_format
CellFormat(backgroundColor=color(1,1,1), textFormat=textFormat(bold=True,␣
↪italic=True))
>>> effective_format - user_format
CellFormat(backgroundColor=color(1,1,1), textFormat=textFormat(bold=True))
>>> effective_format - user_format == default_format
True
```

1.4 Frozen Rows and Columns

The following functions get or set "frozen" row or column counts for a worksheet:

```
get_frozen_row_count(worksheet)
get_frozen_column_count(worksheet)
set_frozen(worksheet, rows=1)
set_frozen(worksheet, cols=1)
set_frozen(worksheet, rows=1, cols=0)
```

1.5 Setting Row Heights and Column Widths

The following functions set the height (in pixels) of rows or width (in pixels) of columns:

```
set_row_height(worksheet, 1, 42)
set_row_height(worksheet, '1:100', 42)
set_row_heights(worksheet, [ ('1:100', 42), ('101:', 22) ])
set_column_width(worksheet, 'A', 190)
set_column_width(worksheet, 'A:D', 100)
set_column_widths(worksheet, [ ('A', 200), ('B:', 100) ])
```

1.6 Working with Right-to-Left Language Alphabets

The following example shows the functions to get or set the *rightToLeft* property of a worksheet:

get_right_to_left(worksheet) set_right_to_left(worksheet, True)

Also note the presence of the argument *textDirection=* to *CellFormat*: set it to *'RIGHT_TO_LEFT'* in order to use right-to-left text in an individual cell in an otherwise left-to-right worksheet.

1.7 Getting and Setting Data Validation Rules for Cells and Cell Ranges

The following functions get or set the "data validation rule" for a cell or cell range:

```
get_data_validation_rule(worksheet, label)
set_data_validation_for_cell_range(worksheet, range, rule)
set_data_validation_for_cell_ranges(worksheet, ranges)
```

The full functionality of data validation rules is supported: all of `BooleanCondition`. See the API documentation for more information. Here's a short example:

```
validation_rule = DataValidationRule(
    BooleanCondition('ONE_OF_LIST', ['1', '2', '3', '4']),
    showCustomUi=True
)
set_data_validation_for_cell_range(worksheet, 'A2:D2', validation_rule)
# data validation for A2
eff_rule = get_data_validation_rule(worksheet, 'A2')
eff_rule.condition.type
>>> 'ONE_OF_LIST'
eff_rule.showCustomUi
>>> True
# No data validation for A1
eff_rule = get_data_validation_rule(worksheet, 'A1')
eff_rule
>>> None
# Clear data validation rule by using None
set_data_validation_for_cell_range(worksheet, 'A2', None)
eff_rule = get_data_validation_rule(worksheet, 'A2')
eff_rule
>>> None
```

1.8 Formatting a Worksheet Using a Pandas DataFrame

If you are using Pandas DataFrames to provide data to a Google spreadsheet – using perhaps the `gspread-dataframe` package available on PyPI – the `format_with_dataframe` function in `gspread_formatting.dataframe` allows you to use that same DataFrame object and specify formatting for a worksheet. There is a `DEFAULT_FORMATTER` in the module, which will be used if no formatter object is provided to `format_with_dataframe`:

```
from gspread_formatting.dataframe import format_with_dataframe, BasicFormatter
from gspread_formatting import Color

# uses DEFAULT_FORMATTER
format_with_dataframe(worksheet, dataframe, include_index=True, include_column_
↪header=True)

formatter = BasicFormatter(
    header_background_color=Color(0,0,0),
    header_text_color=Color(1,1,1),
    decimal_format='#,##0.00'
)

format_with_dataframe(worksheet, dataframe, formatter, include_index=False, include_
↪column_header=True)
```

1.9 Batch Mode for API Call Efficiency

This package offers a "batch updater" object, with methods having the same names and parameters as the formatting functions in the package. The batch updater will gather all formatting requests generated by calling these methods, and send them all to the Google Sheets API in a single `batchUpdate` request when `.execute()` is invoked on

the batch updater. Alternately, you can use the batch updater as a context manager in a `with:` block, which will automate the call to `.execute()`:

```
from gspread_formatting import batch_updater

sheet = some_gspread_worksheet

# Option 1: call execute() directly
batch = batch_updater(sheet.spreadsheet)
batch.format_cell_range(sheet, '1', cellFormat(textFormat=textFormat(bold=True)))
batch.set_row_height(sheet, '1', 32)
batch.execute()

# Option 2: use with: block
with batch_updater(sheet.spreadsheet) as batch:
    batch.format_cell_range(sheet, '1', cellFormat(textFormat=textFormat(bold=True)))
    batch.set_row_height(sheet, '1', 32)
```

1.10 Conditional Format Rules

A conditional format rule allows you to specify a cell format that (additively) applies to cells in certain ranges only when the value of the cell meets a certain condition. The ConditionalFormatRule documentation for the Sheets API describes the two kinds of rules allowed: a `BooleanRule` in which the *CellFormat* is applied to the cell if the value meets the specified boolean condition; or a `GradientRule` in which the `Color` or `ColorStyle` of the cell varies depending on the numeric value of the cell or cells.

You can specify multiple rules for each worksheet present in a Google spreadsheet. To add or remove rules, use the `get_conditional_format_rules(worksheet)` function, which returns a list-like object which you can modify as you would modify a list, and then call `.save()` to store the rule changes you've made.

Here is an example that applies bold text and a bright red color to cells in column A if the cell value is numeric and greater than 100:

```
from gspread_formatting import *

worksheet = some_spreadsheet.worksheet('My Worksheet')

rule = ConditionalFormatRule(
    ranges=[GridRange.from_a1_range('A1:A2000', worksheet)],
    booleanRule=BooleanRule(
        condition=BooleanCondition('NUMBER_GREATER', ['100']),
        format=CellFormat(textFormat=textFormat(bold=True), backgroundColor=Color(1,0,
↪0))
    )
)

rules = get_conditional_format_rules(worksheet)
rules.append(rule)
rules.save()

# or, to replace any existing rules with just your single rule:
rules.clear()
rules.append(rule)
rules.save()
```

An important note: A `ConditionalFormatRule` is, like all other objects provided by this package, mutable in

all of its fields. Mutating a `ConditionalFormatRule` object in place will not automatically store the changes via the Sheets API; but calling *.save()* on the list-like rules object will store the mutated rule as expected.

CHAPTER 2

Installation

2.1 Requirements

- Python 2.7, 3.x; PyPy and PyPy3
- gspread >= 3.0.0

2.2 From PyPI

```
pip install gspread-formatting
```

2.3 From GitHub

```
git clone https://github.com/robin900/gspread-formatting.git
cd gspread-formatting
python setup.py install
```

CHAPTER 3

Development and Testing

Install packages listed in `requirements-dev.txt`. To run the test suite in `test.py` you will need to:

- Authorize as the Google account you wish to use as a test, and download a JSON file containing the credentials. Name the file `creds.json` and locate it in the top-level folder of the repository.
- Set up a `tests.config` file using the `tests.config.example` file as a template. Specify the ID of a spreadsheet that the Google account you are using can access with write privileges.

CHAPTER 4

Module Documentation - Version 1.1.2

CHAPTER 5

Indices and tables

- genindex
- modindex
- search