pygsheets Documentation

Release 1.1

nithinmurali

Aug 28, 2017

Contents

A simple, intutive library for google spreadsheets based on api v4 which gets most of your work done.

CHAPTER 1

Features

- Google spreadsheet api v4 support
- Open, create, delete and share spreadsheets using _title_
- Control permissions of spreadsheets.
- Extract range, entire row or column values.
- Set cell format, text format, coloe, write notes
- NamedRanges Support
- Do all the updates and push the changes in a batch

CHAPTER 2

Small Example

Sample scenario : you want to share a numpy array with your remote friend

```
import pygsheets

gc = pygsheets.authorize()

# Open spreadsheet and then workseet
sh = gc.open('my new ssheet')
wks = sh.sheet1

# Update a cell with value (just to let him know values is updated ;) )
wks.update_cell('A1', "Hey yank this numpy array")

# update the sheet with array
wks.update_cells('A2', my_nparray.to_list())

# share the sheet with your friend
sh.share("myFriend@gmail.com")
```

Sample scenario: you want to store students name and their heights

```
## import and open the sheet as given above

header = wks.cell('A1')
header.value = 'Names'
header.text_format['bold'] = True # make the header bold
header.update()

# or achive the same in oneliner
wks.cell('B1').set_text_format('bold', True).value = 'heights'

# set the names
wks.update_cells('A2:A5',[['name1'],['name2'],['name3'],['name4']])

# set the heights
```

```
heights = wks.range('B2:B5')  # get the range
heights.name = "heights"  # name the range
heights.update_values([[50],[60],[67],[66]]) # update the vales
wks.update_cell('B6','=average(heights)') # set get the avg value
```

CHAPTER 3

Installation

```
pip install https://github.com/nithinmurali/pygsheets/archive/master.zip (recent)
pip install pygsheets (stable)
```

CHAPTER 4

Overview

There are mainly 4 models - `spreadsheet`, `worksheet`, `cell`, `DataRange` they are defined in their respectivr files. The communication with google api is implimented in `client.py`. The client.py also impliments the autorization functions.

CHAPTER 5

Authors and License

The `pygsheets` package is written by Nithin Murali and is inspried by gspread. It's MIT licensed and freely available.

Feel free to improve this package and send a pull request to GitHub.

Contents:

Authorizing pygsheets

There are multiple ways to authorize google sheets. First you should create a developer account (follow below steps) and create the type of credentials depending on your need. But remember not to give away any of there credentials, as your usage quota is limited.

1. Head to Google Developers Console and create a new project (or select the one you have.)
2. You will be redirected to the API Manager, there Under "Library", Google APIs click on "Sheets API".
3. In the API screen click on 'ENABLE' to enable this API
4. Similarly enable the "Drive API". We require drives api for getting list of spreadsheets, deleting them etc.

Now you have to choose the type of credential you want to use. For this you have following two options:

OAuth Credentials

This is the best option if you are trying to edit the spreadsheet on behalf of others. Here for the first time the user will be asked to authenticate your application. From therafter the application can acess all his spreadsheets. For using this you will need 'OAuth client ID' file. Follow this procedure below to generate it -

5. First you need to configure how the conscent sceen will look while asking for autorization. Go to "Credentials" Side Tab and choose "OAuth Conscent screen". There inset all the data you need to show while asking for authorization and save it
6. Go to "Credentials" Tab and choose "Create Credentials > OAuth Client ID".

7. Now choose the Application Type as 'Other'

Now click on the download button to download the 'client_secretxxx.json' file

8. Find the client_id from the file, your application will be able to acess any sheet which is shared with this email. To use this file initiliaze the pygsheets client as shown

```
gc = pygsheets.authorize(outh_file='client_secretxxx.json')
```

First time this will ask you to authorize pygsheets to acess your google sheets and drive. For this it will open a brower, where you have to provide your google credentials and authorize it. This will create a json file with the tokens based on the *outh_creds_store* param. So that you dont have to authorize it everytime you run the application. In case if you already have a file with tokens then you can just pass it as the *outh_file* instead of the client secret file.

Incase you are running the script in a headless server where it can't open a browser, you can enbale *non-local* authorization. Hence instead of opening a browser in the same meachine, it will provide a link which you can run on your local meachine and authorize the application.

```
gc = pygsheets.authorize(outh_file='client_secretxxx.json', outh_nonlocal=True)
```

Signed Credentials

In this option you will be given an unique email, and your application will be able to acesss all the sheets shared with that email. No Authentication will be required in this case.

5. Go to "Credentials" Tab and choose "Create Credentials > Service Account Key".
6. Now choose the service account as 'App Engine default' and Key type as JSON and click create

You will automatically download a JSON file with this data.

This is how this file may look like:

```
{
    "type": "service_account",
    "project_id": "p....sdf",
    "private_key_id": "48.....",
    "private_key": "-----BEGIN PRIVATE KEY-----\nNrDyLw ... jINQh/9\n-----END PRIVATE␣
↪KEY-----\n",
    "client_email": "p.....@appspot.gserviceaccount.com",
    "client_id": "10.....454",
}
```

7. Find the client_id from the file, your application will be able to acess any sheet which is shared with this email. To use this file initiliaze the pygsheets client as shown

```
gc = pygsheets.authorize(service_file='service_creds.json')
```

Custom Credentials Objects

If you have another method of authenicating you can easily create a custom credentials object.

```
class Credentials (object):
    def __init__ (self, access_token=None):
        self.access_token = access_token

    def refresh (self, http):
```

```
        # get new access_token
        # this only gets called if access_token is None
```

Then you could pass this for authorization as

```
gc = pygsheets.authorize(credentials=mycreds)
```

pygsheets Reference

pygsheets is a simple 'Google Spreadsheets v4 '_ API wrapper.

Top Level Interface

pygsheets.**authorize**(*outh_file='client_secret.json'*, *outh_creds_store=None*, *outh_nonlocal=False*, *service_file=None*, *credentials=None*, ***client_kwargs*)

Login to Google API using OAuth2 credentials.

This function instantiates *Client* and performs authentication.

Parameters

- **outh_file** – path to outh2 credentials file, or tokens file
- **outh_creds_store** – path to directory where tokens should be stored 'global' if you want to store in system-wide location None if you want to store in current script directory
- **outh_nonlocal** – if the authorization should be done in another computer, this will provide a url which when run will ask for credentials
- **service_file** – path to service credentials file
- **credentials** – outh2 credentials object
- **no_cache** – (http client arg) do not ask http client to use a cache in tmp dir, useful for environments where filesystem access prohibited default: False

Returns *Client* instance.

class pygsheets.**Client**(*oauth*, *http_client=None*, *retries=1*, *no_cache=False*)

An instance of this class communicates with Google API.

Parameters

- **oauth** – An OAuth2 credential object. Credential objects are those created by the oauth2client library. https://github.com/google/oauth2client
- **http_client** – (optional) A object capable of making HTTP requests
- **retries** – (optional) The number of times connection will be tried before raising a timeout error.

```
>>> c = pygsheets.Client(oauth=OAuthCredentialObject)
```

create(*title*, *parent_id=None*)

Creates a spreadsheet, returning a *Spreadsheet* instance.

Parameters

- **parent_id** – id of the parent folder, where the spreadsheet is to be created

- **title** – A title of a spreadsheet.

delete (*title=None*, *spreadsheet_id=None*)
Deletes, a spreadsheet by title or id.

Parameters

- **title** – title of a spreadsheet.
- **spreadsheet_id** – id of a spreadsheet this takes precedence if both given.

Raises ***pygsheets.SpreadsheetNotFound*** – if no spreadsheet is found.

open (*title*)
Opens a spreadsheet, returning a *Spreadsheet* instance.

Parameters **title** – A title of a spreadsheet.

If there's more than one spreadsheet with same title the first one will be opened.

Raises ***pygsheets.SpreadsheetNotFound*** – if no spreadsheet with specified *title* is found.

open_by_key (*key*, *returnas='spreadsheet'*)
Opens a spreadsheet specified by *key*, returning a *Spreadsheet* instance.

Parameters

- **key** – A key of a spreadsheet as it appears in a URL in a browser.
- **returnas** – return as spreadsheet or json object

Raises ***pygsheets.SpreadsheetNotFound*** – if no spreadsheet with specified *key* is found.

```
>>> c = pygsheets.authorize()
>>> c.open_by_key('0BmgG6nO_6dprdS1MN3d3MkdPa142WFRrdnRRUWl1UFE')
```

open_by_url (*url*)
Opens a spreadsheet specified by *url*,

Parameters **url** – URL of a spreadsheet as it appears in a browser.

Raises ***pygsheets.SpreadsheetNotFound*** – if no spreadsheet with specified *url* is found.

Returns a :class: *pygsheets.Spreadsheet* instance.

```
>>> c = pygsheets.authorize()
>>> c.open_by_url('https://docs.google.com/spreadsheet/ccc?key=0Bm...FE&hl')
```

open_all (*title=None*)
Opens all available spreadsheets,

Parameters **title** – (optional) If specified can be used to filter spreadsheets by title.

Returns list of *Spreadsheet* instances

list_ssheets ()
Lists all spreadsheets

Models

The models represent common spreadsheet objects: *spreadsheet*, *worksheet* and *cell*.

Spreadsheet

class pygsheets.**Spreadsheet**(*client*, *jsonsheet=None*, *id=None*)
A class for a spreadsheet object.

id
id of the spreadsheet

title
title of the spreadsheet

sheet1
Shortcut property for getting the first worksheet.

named_ranges
All named ranges in thi spreadsheet

defaultformat
deafault cell format

updated
Last time the spreadsheet was modified, in RFC 3339 format

update_properties(*jsonsheet=None*, *fetch_sheets=True*)
Update all sheet properies.

Parameters

- **jsonsheet** – json object to update values form if not specified, will fetch it and update (see google api, for json format)
- **fetch_sheets** – if the sheets should be fetched

worksheets(*sheet_property=None*, *value=None*, *force_fetch=False*)
Get all worksheets filtered by a property.

Parameters

- **sheet_property** – proptery to filter - 'title', 'index', 'id'
- **value** – value of property to match
- **force_fetch** – update the sheets, from cloud

Returns list of all *worksheets*

worksheet(*property='index'*, *value=0*)
Returns a worksheet with specified property.

Parameters

- **property** (*'title','index','id'*) – A property of a worksheet. If there're multiple worksheets with the same title, first one will be returned.
- **value** – value of given property

Returns instance of *Worksheet*

Example. Getting worksheet named 'Annual bonuses'

```
>>> sht = client.open('Sample one')
>>> worksheet = sht.worksheet('title','Annual bonuses')
```

worksheet_by_title(*title*)
returns worksheet by title

Parameters **title** – title of the sheet

Returns Spresheet instance

add_worksheet(*title*, *rows=100*, *cols=26*, *src_tuple=None*, *src_worksheet=None*)
Adds a new worksheet to a spreadsheet.

Parameters

- **title** – A title of a new worksheet.
- **rows** – Number of rows.
- **cols** – Number of columns.
- **src_tuple** – a tuple (spreadsheet id, worksheet id) specifying a worksheet to copy
- **src_worksheet** – source worksheet object to copy values from

Returns a newly created *worksheets*.

del_worksheet(*worksheet*)
Deletes a worksheet from a spreadsheet.

Parameters **worksheet** – The *worksheets* to be deleted.

find(*string*, *replace=None*, *regex=True*, *match_case=False*, *include_formulas=False*, *srange=None*, *sheet=True*)
Find and replace cells in spreadsheet

Parameters

- **string** – string to search for
- **replace** – string to replace with
- **regex** – is the search string regex
- **match_case** – match case in search
- **include_formulas** – include seach in formula
- **srange** – range to search in A1 format
- **sheet** – if True - search all sheets, else search specified sheet

share(*addr*, *role='reader'*, *expirationTime=None*, *is_group=False*)
create/update permission for user/group/domain

Parameters

- **addr** – this is the email for user/group and domain address for domains
- **role** – permission to be applied ('owner','writer','commenter','reader')
- **expirationTime** – (Not Implimented) time until this permission should last (datetime)
- **is_group** – boolean , Is this a use/group used only when email provided

list_permissions()
list all the permissions of the spreadsheet

Returns list of permissions as json object

remove_permissions(*addr*)
Removes all permissions of the user provided

Parameters **addr** – email/domain of the user

batch_start()
Start batch mode, where all updates to sheet values will be batched

batch_stop(*discard=False*)
Stop batch Mode and Update the changes

Parameters **discard** – discard all changes done in batch mode

link(*syncToCloud=False*)
Link the spreadsheet with cloud, so all local changes will be updated instantly, so does all data fetches

Parameters **syncToCloud** – true -> update the cloud with local changes false -> update the local copy with cloud

unlink()
Unlink the spread sheet with cloud, so all local changes will be made on local copy fetched

export(*fformat='text/csv:.csv'*, *filename=None*)
Export all the worksheet of the worksheet in specified format.

Parameters

- **fformat** – A format of the output as Enum ExportType
- **filename** – name of file exported with extension

custom_request(*request*, *fields*)
send a custom batch update request for this spreadsheet

Parameters

- **request** – the json batch update request or a list of requests
- **fields** – fields to include in the response

Returns json Response

Worksheet

class pygsheets.**Worksheet**(*spreadsheet*, *jsonSheet*)
A class for worksheet object.

Parameters

- **spreadsheet** – Spreadsheet object to which this worksheet belongs to
- **jsonSheet** – The JsonSheet containing all properties of this sheet Ref to api details for more info

id
Id of a worksheet.

index
Index of worksheet

title
Title of a worksheet.

rows
Number of rows

cols
Number of columns

refresh(*update_grid=False*)
refresh worksheet data

link(*syncToCloud=True*)
Link the spread sheet with cloud, so all local changes will be updated instantly, so does all data fetches

Parameters **syncToCloud** – update the cloud with local changes if set to true update the local copy with cloud if set to false

unlink()
Unlink the spread sheet with cloud, so all local changes will be made on local copy fetched

sync()
sync the worksheet to cloud

cell(*addr*)
Returns cell object at given address.

Parameters **addr** – cell address as either tuple (row, col) or cell label 'A1'

Returns an instance of a *Cell*

Example:

```
>>> wks.cell((1,1))
<Cell R1C1 "I'm cell A1">
>>> wks.cell('A1')
<Cell R1C1 "I'm cell A1">
```

range(*crange*, *returnas='cells'*)
Returns a list of *Cell* objects from specified range.

Parameters

- **crange** – A string with range value in common format, e.g. 'A1:A5'.
- **returnas** – cann be 'matrix', 'cell', 'range' the correponding type will be returned

get_value(*addr*)
value of a cell at given address

Parameters **addr** – cell address as either tuple or label

get_values(*start*, *end*, *returnas='matrix'*, *majdim='ROWS'*, *include_empty=True*, *include_all=False*, *value_render='FORMATTED_VALUE'*)
Returns value of cells given the topleft corner position and bottom right position

Parameters

- **start** – topleft position as tuple or label
- **end** – bottomright position as tuple or label
- **majdim** – output as rowwise or columwise, only for matrix takes - 'ROWS' or 'COLMUNS'
- **returnas** – return as list of strings of cell objects takes - 'matrix', 'cell', 'range'
- **include_empty** – include empty trailing cells/values until last non-zero value, ignored if inclue_all is True
- **include_all** – include all the cells in the range empty/non-empty, will return exact rectangle
- **value_render** – format of output values

Example:

```
>>> wks.get_values((1,1),(3,3))
[[u'another look.', u'', u'est'],
 [u'EE 4212', u"it's down there "],
 [u'ee 4210', u'somewhere, let me take ']]
```

get_all_values(*returnas='matrix'*, *majdim='ROWS'*, *include_empty=True*)
Returns a list of lists containing all cells' values as strings.

Parameters

- **majdim** – output as row wise or columwise
- **returnas** (*'matrix', 'cell'*) – return as list of strings of cell objects
- **include_empty** – whether to include empty values

Example:

```
>>> wks.get_all_values()
[[u'another look.', u'', u'est'],
 [u'EE 4212', u"it's down there "],
 [u'ee 4210', u'somewhere, let me take ']]
```

get_all_records(*empty_value=''*, *head=1*)

Returns a list of dictionaries, all of them having:

- the contents of the spreadsheet's with the head row as keys, And each of these dictionaries holding
- the contents of subsequent rows of cells as values.

Cell values are numericised (strings that can be read as ints or floats are converted).

Parameters

- **empty_value** – determines empty cell's value
- **head** – determines wich row to use as keys, starting from 1 following the numeration of the spreadsheet.

Returns a list of dict with header column values as head and rows as list

get_row(*row*, *returnas='matrix'*, *include_empty=True*)
Returns a list of all values in a *row*.

Empty cells in this list will be rendered as :const:' '.

Parameters

- **include_empty** – whether to include empty values
- **row** – index of row
- **returnas** – ('matrix' or 'cell') return as cell objects or just 2d array

get_col(*col*, *returnas='matrix'*, *include_empty=True*)
Returns a list of all values in column *col*.

Empty cells in this list will be rendered as :const:' '.

Parameters

- **include_empty** – whether to include empty values
- **col** – index of col

- **returnas** – ('matrix' or 'cell') return as cell objects or just values

update_cell(*addr*, *val*, *parse=True*)
Sets the new value to a cell.

Parameters

- **addr** – cell address as tuple (row,column) or label 'A1'.
- **val** – New value
- **parse** – if False, values will be stored as else as if the user typed them into the UI

Example:

```
>>> wks.update_cell('A1', '42') # this could be 'a1' as well
<Cell R1C1 "42">
>>> wks.update_cell('A3', '=A1+A2', True)
<Cell R1C3 "57">
```

update_cells(*crange=None*, *values=None*, *cell_list=None*, *extend=False*, *majordim='ROWS'*, *parse=True*)
Updates cells in batch, it can take either a cell list or a range and values. cell list is only efficient for large lists.

Parameters

- **cell_list** – List of a *Cell* objects to update with their values
- **crange** – range in format A1:A2 or just 'A1' or even (1,2) end cell will be infered from values
- **values** – matrix of values if range given, if a value is None its unchanged
- **extend** – add columns and rows to the workspace if needed (not for cell list)
- **majordim** – major dimension of given data
- **parse** – if the values should be as if the user typed them into the UI else its stored as is

update_col(*index*, *values*, *row_offset=0*)
update an existing colum with values

Parameters

- **index** – index of the starting column form where value should be inserted
- **values** – values to be inserted as matrix, column major
- **row_offset** – rows to skip before inserting values

update_row(*index*, *values*, *col_offset=0*)
update an existing row with values

Parameters

- **index** – index of the starting row form where value should be inserted
- **values** – values to be inserted as matrix
- **col_offset** – columns to skip before inserting values

resize(*rows=None*, *cols=None*)
Resizes the worksheet.

Parameters

- **rows** – New rows number.

- **cols** – New columns number.

add_rows(*rows*)
Adds rows to worksheet.

Parameters **rows** – Rows number to add.

add_cols(*cols*)
Adds colums to worksheet.

Parameters **cols** – Columns number to add.

delete_cols(*index*, *number=1*)
delete a number of colums stating from index

Parameters

- **index** – indenx of first col to delete
- **number** – number of cols to delete

delete_rows(*index*, *number=1*)
delete a number of rows stating from index

Parameters

- **index** – index of first row to delete
- **number** – number of rows to delete

insert_cols(*col*, *number=1*, *values=None*, *inherit=False*)
Insert a column after the column <col> and fill with values <values> Widens the worksheet if there are more values than columns.

Parameters

- **col** – column number after which new column should be inserted
- **number** – number of columns to be inserted
- **values** – values matrix to filled in new column
- **inherit** – If dimension properties should be extended from the dimensions before or after the newly inserted dimensions

insert_rows(*row*, *number=1*, *values=None*, *inherit=False*)
Insert a row after the row <row> and fill with values <values> Widens the worksheet if there are more values than columns.

Parameters

- **row** – row after which new colum should be inserted
- **number** – number of rows to be inserted
- **values** – values matrix to be filled in new row
- **inherit** – If dimension properties should be extended from the dimensions before or after the newly inserted dimensions

clear(*start='A1'*, *end=None*)
clears the worksheet by default, if range given then clears range

Parameters

- **start** – topright cell address
- **end** – bottom left cell of range

adjust_column_width(*start*, *end=None*, *pixel_size=100*)
Adjust the width of one or more columns

Parameters

- **start** – index of the column to be resized
- **end** – index of the end column that will be resized
- **pixel_size** – width in pixels

adjust_row_height(*start*, *end=None*, *pixel_size=100*)
Adjust the height of one or more rows

Parameters

- **start** – index of the row to be resized
- **end** – index of the end row that will be resized
- **pixel_size** – height in pixels

append_table(*start='A1'*, *end=None*, *values=None*, *dimension='ROWS'*, *overwrite=False*)

Search for a table in the given range and will append it with values

Parameters

- **start** – start cell of range
- **end** – end cell of range
- **values** – List of values for the new row.
- **dimension** – table dimension on which the values should be appended. can be 'ROWS' or 'COLUMNS'
- **overwrite** – The new data overwrites existing data in the areas it is written.

find(*query*, *replace=None*, *force_fetch=True*)
Finds first cell matching query.

Parameters

- **query** – A text string or compiled regular expression.
- **replace** – string to replace
- **force_fetch** – if local datagrid should be updated before searching, even if file is not modified

create_named_range(*name*, *start*, *end*)
Create a named range in this sheet

Parameters

- **name** – Name of the named range
- **start** – top right cell adress
- **end** – bottom right cell adress

get_named_ranges(*name=''*)
get a named range given name

Parameters **name** – Name of the named range to be retrived, if omitted all ranges are retrived

Returns *DataRange*

delete_named_range(*name*, *range_id=''*)
delete a named range

Parameters

- **name** – name of named range to be deleted
- **range_id** – id of the named range

set_dataframe(*df*, *start*, *copy_index=False*, *copy_head=True*, *fit=False*, *escape_formulae=False*, *nan='NaN'*)
set the values of a pandas dataframe at cell <start>

Parameters

- **df** – pandas dataframe
- **start** – top right cell address from where values are inserted
- **copy_index** – if index should be copied (multi index supported)
- **copy_head** – if headers should be copied
- **fit** – should the worksheet should be resized to fit the dataframe
- **escape_formulae** – If any value starts with an equals sign =, it will be prefixed with a apostrophe ', to avoid being interpreted as a formula.
- **nan** – value to replace NaN with

get_as_df(*has_header=True*, *index_colum=None*, *start=None*, *end=None*, *numerize=True*, *empty_value=''*)
get value of worksheet as a pandas dataframe

Parameters

- **has_header** – If is True intrept first row as DF header
- **index_colum** – worksheet column number to use as DF index
- **numerize** – If True, cell values will be numerized
- **empty_value** – value used to indicate empty cell value
- **start** – top left cell of dataframe, if not set whole sheet will be fetched
- **end** – bottom right cell of dataframe, if not set whole sheet will be fetched

Returns pandas.Dataframe

export(*fformat='text/csv:.csv'*, *filename=None*)
Export the worksheet in specified format.

Parameters

- **fformat** – A format of the output as Enum ExportType
- **filename** – name of file exported with extension

copy_to(*spreadsheet_id*)
copy the worksheet to specified spreadsheet

Parameters **spreadsheet_id** – id of the spreadsheet to copy

DataRange

class pygsheets.**DataRange**(*start=None*, *end=None*, *worksheet=None*, *name=''*, *data=None*, *name_id=None*, *namedjson=None*)

DataRange specifes a range of cells in the sheet

Parameters

- **start** – top left cell adress
- **end** – bottom right cell adress
- **worksheet** – worksheet where this range belongs
- **name** – name of the named range
- **name_id** – id of named range
- **namedjson** – json representing the NamedRange from api

name
name of the named range, setting a name will make this a range a named range setting this to '' will delete the named range

protect
(boolean) if this range is protected

start_addr
topleft adress of the range

end_addr
bottomright adress of the range

range
Range in format A1:C5

link(*update=True*)
link the dstarange so that all propertis are synced right after setting them

Parameters **update** – if the range should be synced to cloud on link

unlink()
unlink the sheet so that all properties are not synced as its changed

fetch(*only_data=True*)
update the range data/ properties from cloud

Parameters **only_data** – fetch only data

applay_format(*cell*)
Change format of all cells in the range

Parameters **cell** – a model :class: Cell whose format will be applied to all cells

update_values(*values=None*)
Update the values of the cells in this range

Parameters **values** – values as matrix

update_named_range()
update the named properties

Cell

class pygsheets.**Cell**(*pos*, *val=''*, *worksheet=None*, *cell_data=None*)

An instance of this class represents a single cell. A cell can be simple or complex. A complex cell will update all information on each value acess (more bandwidth). in a *worksheet*.

Parameters

- **pos** – position of the cell adress
- **val** – value of the cell
- **worksheet** – worksheet this cell belongs to
- **cell_data** – Data about the cell in json, corresponding to cellData of sheets api

format = None
tuple specifying data format (format type, pattern) or just format

borders = None
border properties as json, see gsheets api docs

parse_value = None
if set false, value will be shown as it is set

row
Row number of the cell.

col
Column number of the cell.

label
Cell Label - Eg A1

value
get/set formatted value of the cell

value_unformatted
get unformatted value of the cell

formula
get/set formula if any of the cell

note
get/set note on the cell

color
get/set background color of the cell as tuple (red, green, blue, alpha)

simple
If this cell is simple. Simple cells will only fetch value, else it would fetch all the cell attributes

set_text_format(*attribute*, *value*)
set the text format

Parameters

- **attribute** – one of the following "foregroundColor" "fontFamily", "fontSize", "bold", "italic", "strikethrough", "underline"
- **value** – corresponding value for the attribute

Returns

class Cell

set_text_rotation(*attribute*, *value*)
set the text rotation

Parameters

- **attribute** – "angle" or "vertical"
- **value** – corresponding value for the attribute. angle in (-90,90) for 'angle', boolean for 'vertical'

Returns *cell*

set_text_alignment(*alignment*, *direction=None*)
set text alignment in both the directions

param alignment either LEFT, CENTER, RIGHT, TOP, MIDDLE, BOTTOM, None

param direction Verical or horizondal; mandatory only if alignment is None

unlink()
unlink the cell from worksheet. Unliked cells wont updated if any properties are changed. you have to lihnk again or call update to sync all changes values

link(*worksheet=None*, *update=False*)
link cell with a worksheet. Linked sheets will be updated instantanoulsy if any properties are changed These are most helpful if you are using a python terminal.

Parameters

- **worksheet** – the worksheet to link to
- **update** – if the cell should be synces as after linking

Returns *cell*

neighbour(*position*)
get a neighbouring cell of this cell

Parameters **position** – a tuple of relative position of position as string as right, left, top, bottom or combinatoin

Returns *neighbouring cell*

fetch(*keep_simple=False*)
Update the value of the cell from sheet

update(*force=False*)
update the sheet cell value with the attributes set

Parameters **force** – update the cell even if its unlinked

get_json()
get the json representation of the cell as per google api

set_json(*cell_data*)
set the cell data from json obj of the cell as per google api

Parameters **cell_data** – json data about cell

Exceptions

exception pygsheets.**AuthenticationError**
An error during authentication process.

exception pygsheets.**SpreadsheetNotFound**
Trying to open non-existent or inaccessible spreadsheet.

exception pygsheets.**WorksheetNotFound**
Trying to open non-existent or inaccessible worksheet.

exception pygsheets.**NoValidUrlKeyFound**
No valid key found in URL.

exception pygsheets.**IncorrectCellLabel**
The cell label is incorrect.

exception pygsheets.**RequestError**
Error while sending API request.

exception pygsheets.**InvalidUser**
Invalid user/domain

exception pygsheets.**InvalidArgumentValue**
Invalid value for argument

Indices and tables

- genindex
- modindex
- search

CHAPTER 6

Indices and tables

- genindex
- modindex
- search

Python Module Index

p

Index

N

O

P

R

S

T

U

V

W