pygsheets Documentation

Release 1.1

nithinmurali

Nov 30, 2022

Contents

A simple, intuitive python library to access google spreadsheets through the Google Sheets API v4. So for example if you have few csv files which you want to export to google sheets and then plot some graphs based on the data. You can use this library to automate that.

CHAPTER 1

Features

- Google Sheets API v4 support.
- Limited Google Drive API v3 support.
- Open and create spreadsheets by **title**.
- Add or remove permissions from you spreadsheets.
- Simple calls to get a row, column or defined range of values.
- Change the formatting properties of a cell.
- Supports named ranges & protected ranges.
- Queue up requests in batch mode and then process them in one go.

CHAPTER 2

Updates

New version 2.0.0 released. Please see changelog to migrate from 1.x.

CHAPTER 3

Small Example

First example - Share a numpy array with a friend:

```
import pygsheets

client = pygsheets.authorize()

# Open the spreadsheet and the first sheet.
sh = client.open('spreadsheet-title')
wks = sh.sheet1

# Update a single cell.
wks.update_value('A1', "Numbers on Stuff")

# Update the worksheet with the numpy array values. Beginning at cell 'A2'.
wks.update_values('A2', my_numpy_array.to_list())

# Share the sheet with your friend. (read access only)
sh.share('friend@gmail.com')
# sharing with write access
sh.share('friend@gmail.com', role='writer')
```

Second example - Store some data and change cell formatting:

```
# open a worksheet as in the first example.

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

# The same can be achieved in one line
wks.cell('B1').set_text_format('bold', True).value = 'heights'

# set the names
```

(continues on next page)

(continued from previous page)

```
wks.update_values('A2:A5',[['name1'],['name2'],['name3'],['name4']])

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

CHAPTER 4

Installation

Install recent:

```
pip install https://github.com/nithinmurali/pygsheets/archive/master.zip
```

Install stable:

```
pip install pygsheets
```

CHAPTER 5

Overview

The *Client* is used to create and access spreadsheets. The property *drive* exposes some Google Drive API functionality and the *sheet* property exposes used Google Sheets API functionality.

A Google Spreadsheet is represented by the *Spreadsheet* class. Each spreadsheet contains one or more *Worksheet*. The data inside of a worksheet can be accessed as plain values or inside of a *Cell* object. The cell has properties and attributes to change formatting, formulas and more. To work with several cells at once a *DataRange* can be used.

CHAPTER 6

Authors and License

The `pygsheets` package is written by Nithin M and is inspired by gspread.

Licensed under the MIT-License.

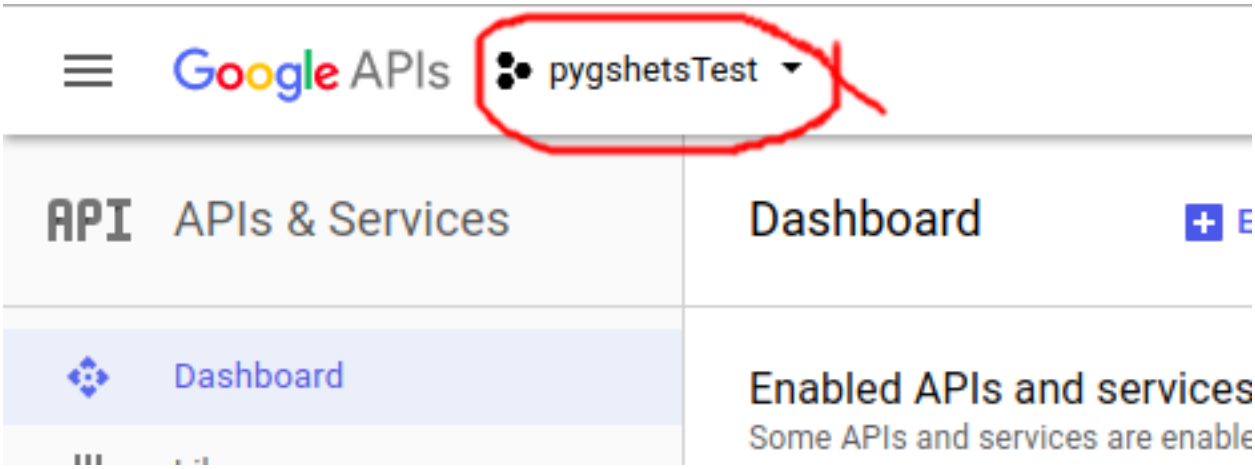
Feel free to improve this package and send a pull request to GitHub.

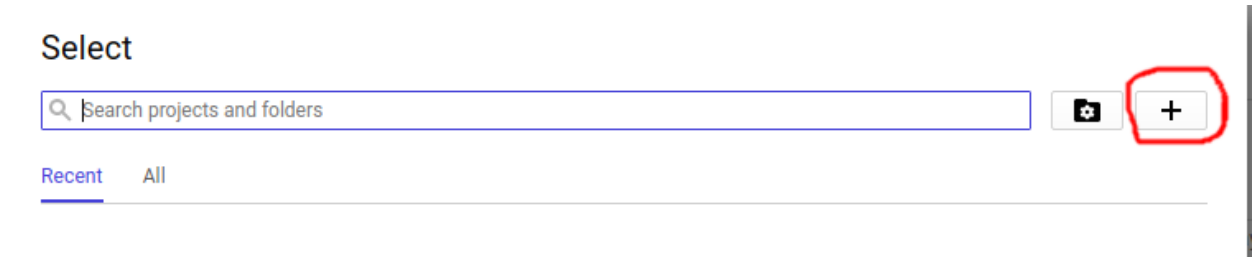
Contents:

6.1 Authorizing pygsheets

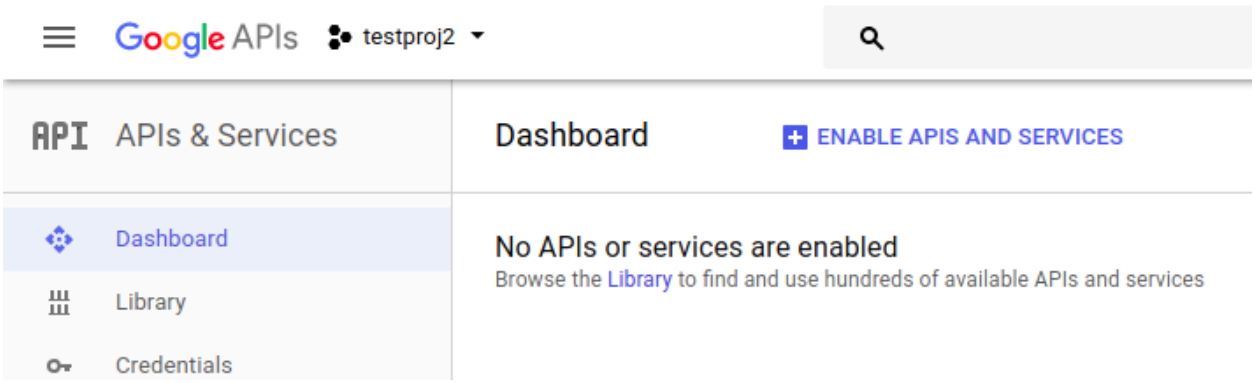
There are multiple ways to authorize google sheets. First you should create a developer account (follow below steps) and create the type of credentials depending on your need. These credentials give the python script access to a google account. But remember not to give away any of these credentials,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

1. Head to Google Developers Console and create a new project (or select the one you have.)

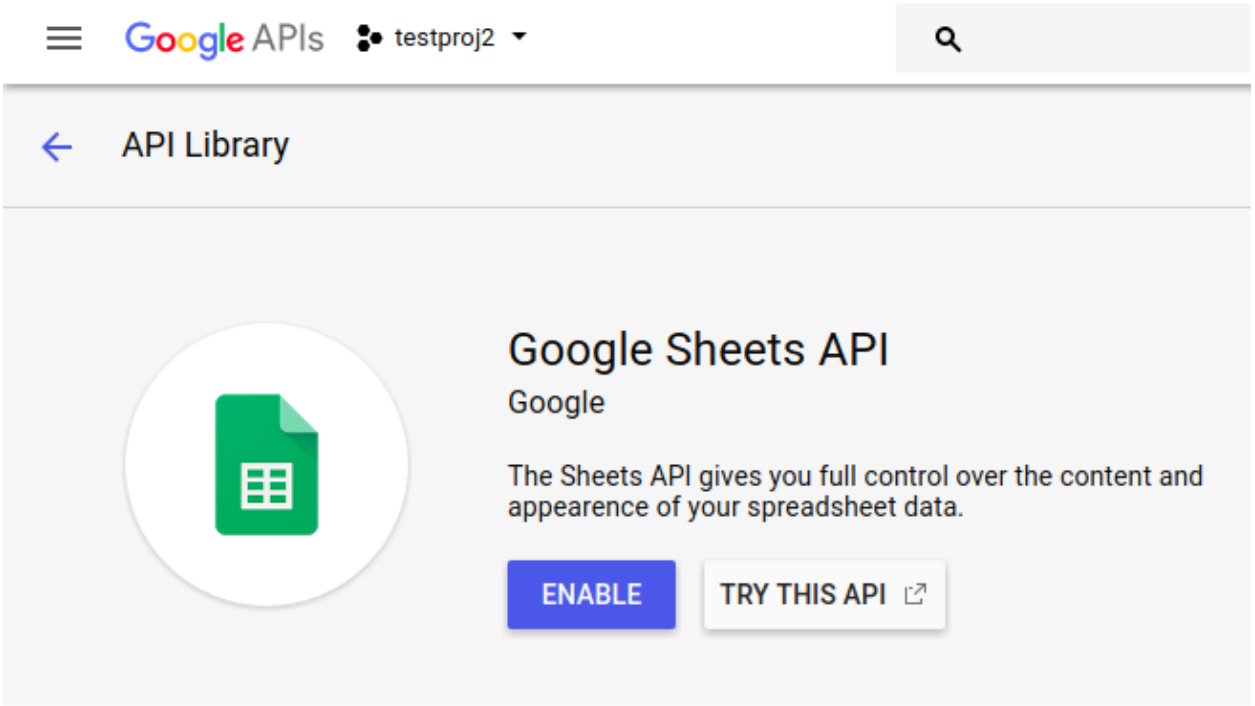




2. You will be redirected to the Project Dashboard, there click on “Enable Apis and services”, search for “Sheets API”.



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:

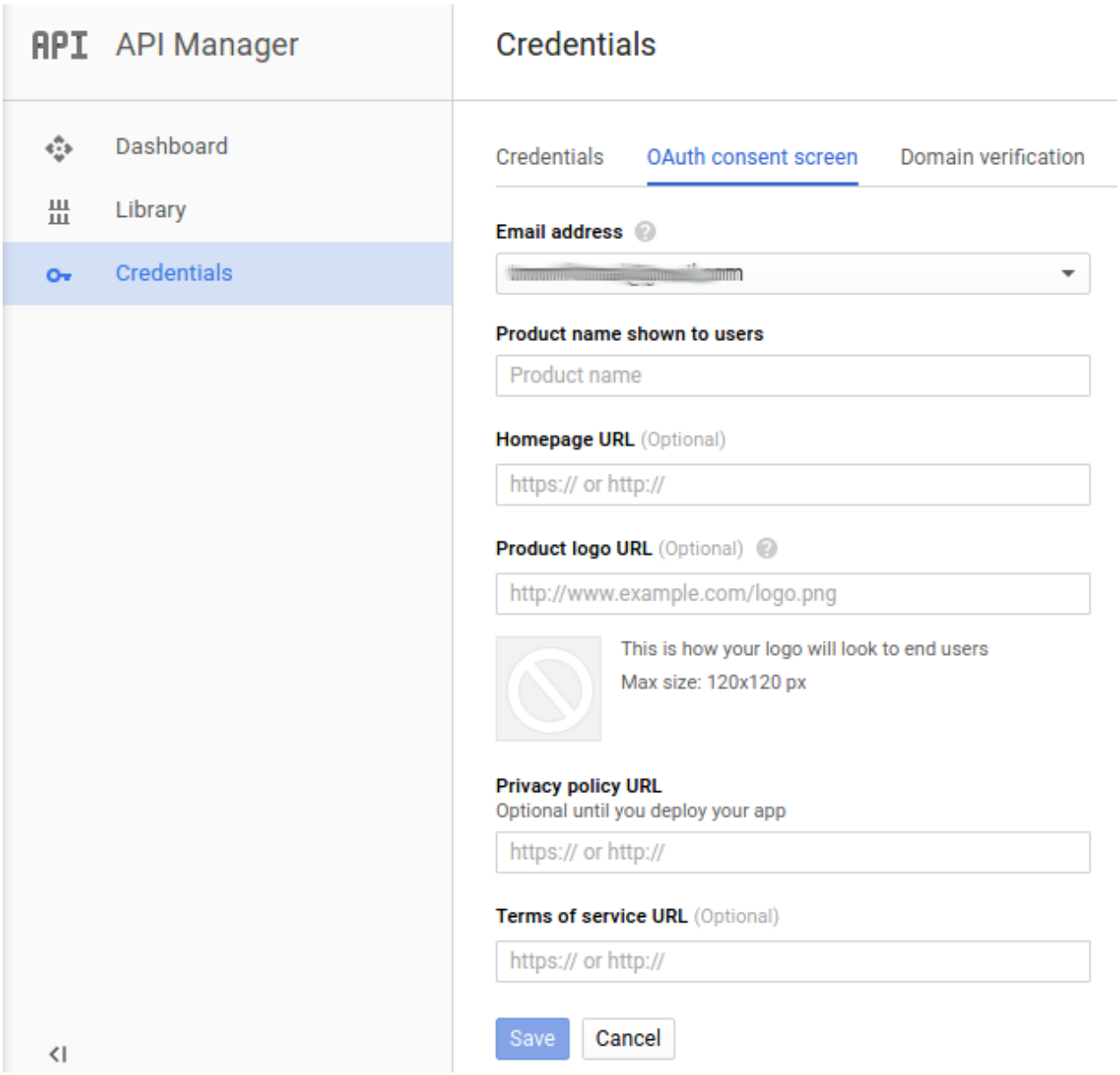
6.1.1 OAuth Credentials

This is the best option if you are trying to edit the spreadsheet on behalf of others. Using this method, your script can get access to all the spreadsheets of the account. The authorization process (giving password and email) has to be completed only once. Which will grant the application full access to all of the users sheets. Follow this procedure below to get the client secret:

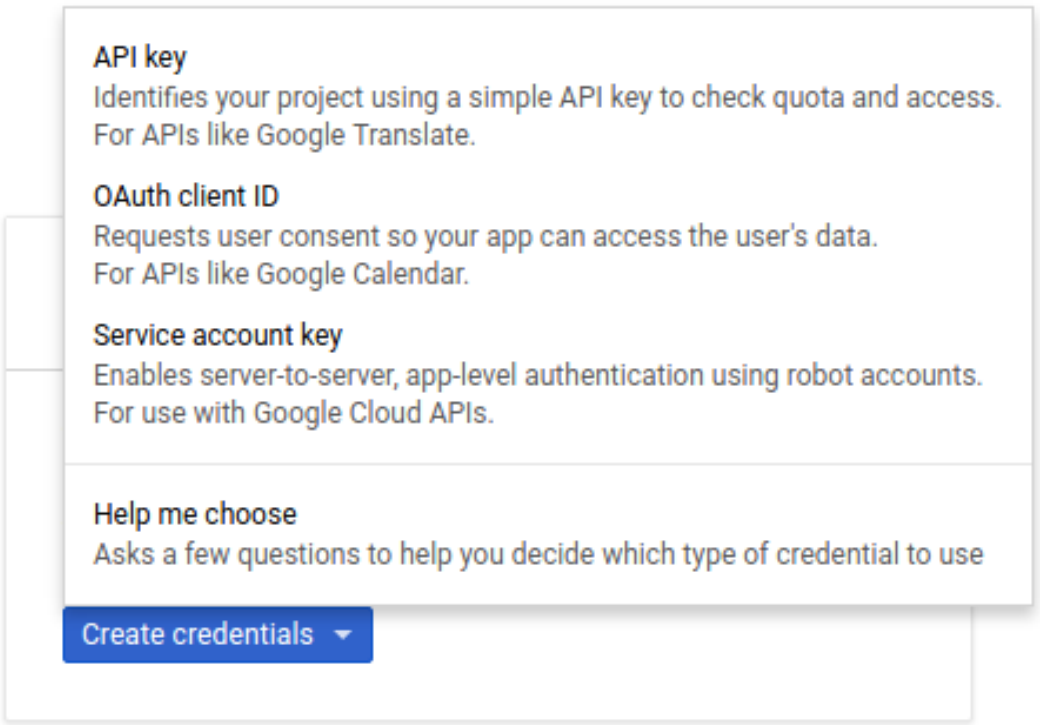
Note: Make sure not to share the created authentication file with anyone, as it will give direct access to your enabled APIs.

---

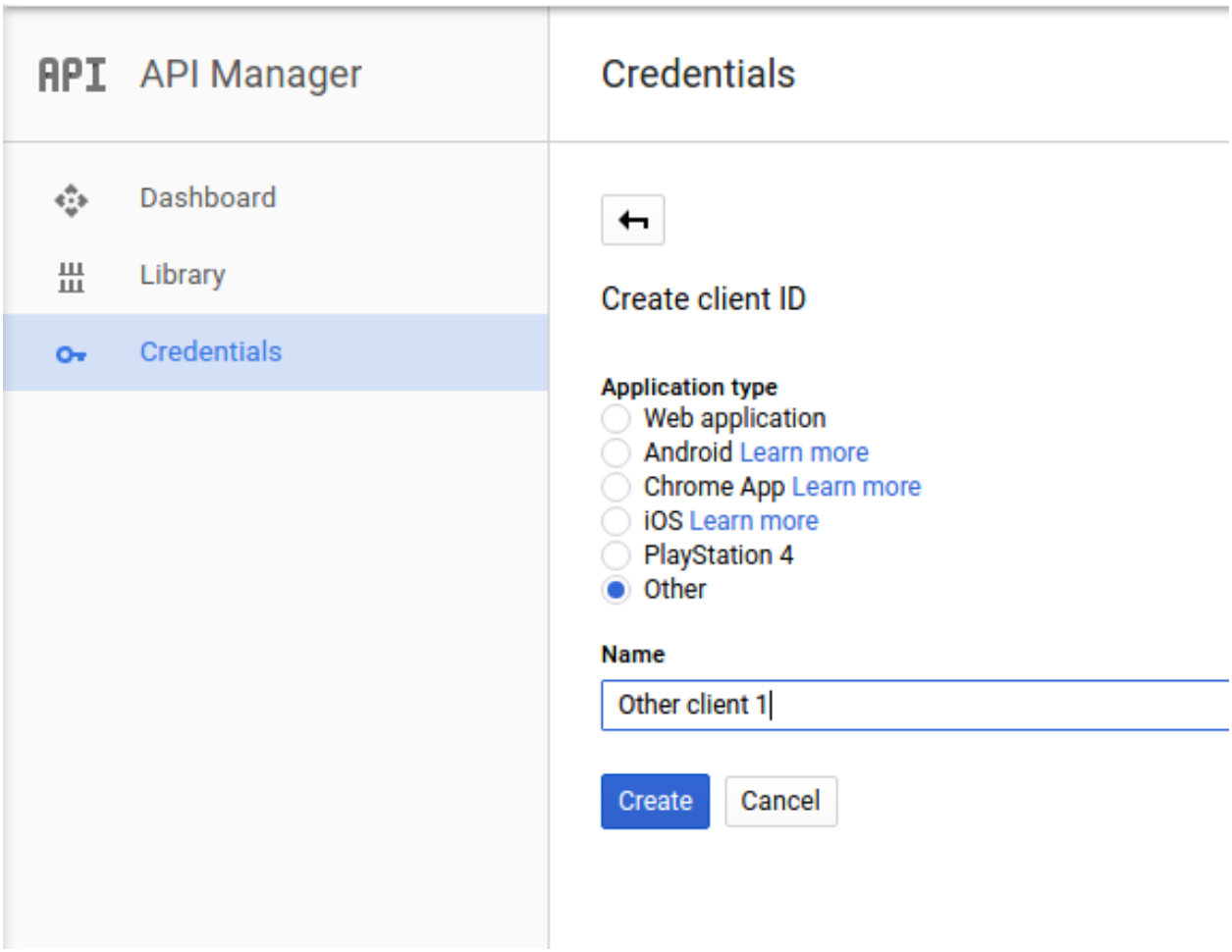
5. First you need to configure how the consent screen will look while asking for authorization. Go to "Credentials" side tab and choose "OAuth Consent screen". Input all the required data in the form.



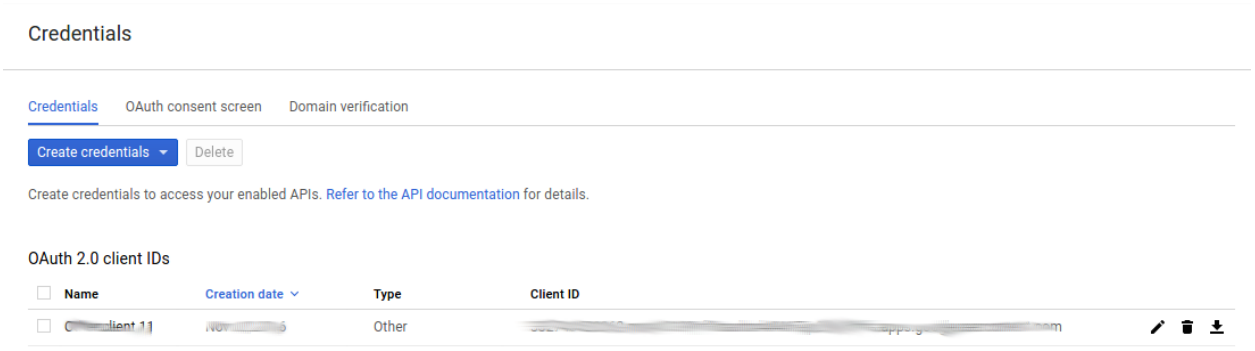
6. Go to “Credentials” tab and choose “Create Credentials > OAuth Client ID”.



7. Next choose the application type as 'Other'



8. Next click on the download button to download the 'client_secret[...].json' file, make sure to remember where you saved it:



9. By default authorize() expects a filed named 'client_secret.json' in the current working directory. If you did not save the file there and renamed it, make sure to set the path:

```
gc = pygsheets.authorize(client_secret='path/to/client_secret[...].json')
```

The first time this will ask you to complete the authentication flow. Follow the instructions in the console to complete. Once completed a file with the authentication token will be stored in your current working directory (to change this set credentials_directory). This file is used so that you don't have to authorize it every time you run the application. So if you need to authorize script again you don't need the client_secret but just this generated json file will do (pass

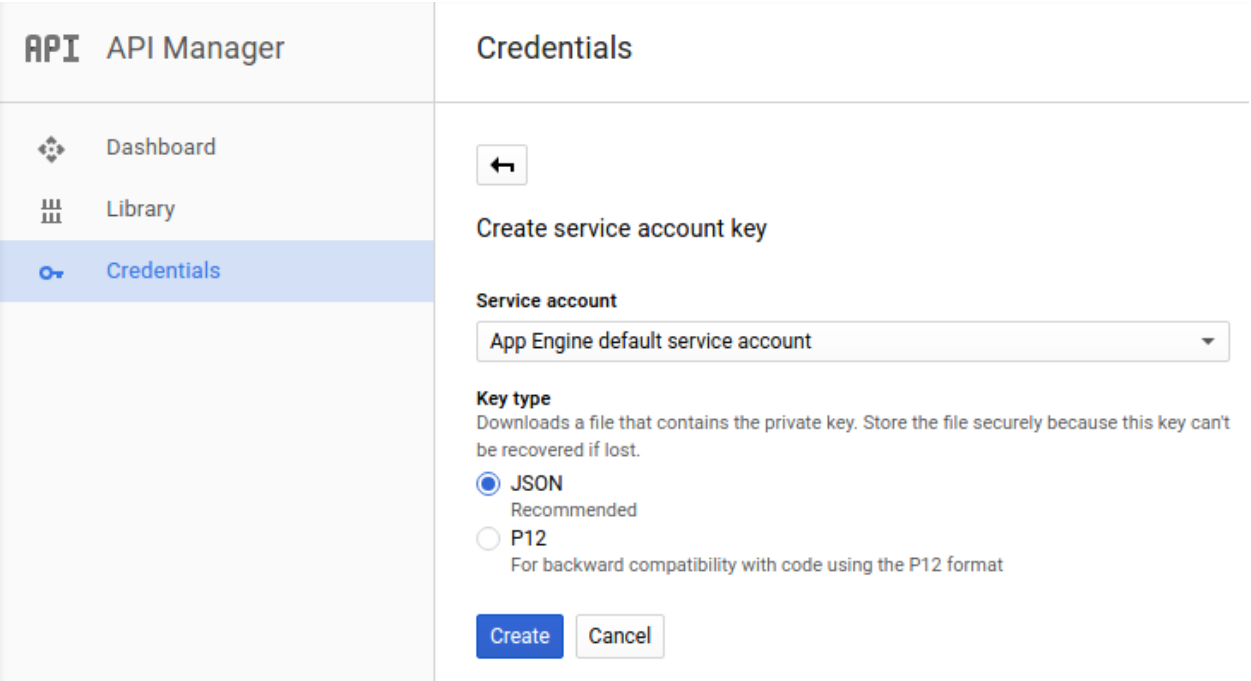
its path as credentials_directory).

Please note that credentials_directory will override your client_secret. So if you keep getting logged in with some other account even when you are passing your accounts client_secret, credentials_directory might be the culprit.

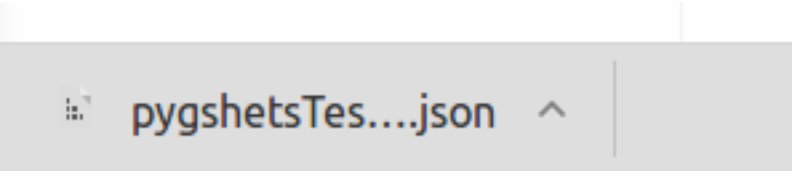
6.1.2 Service Account

A service account is an account associated with an email. The account is authenticated with a pair of public/private key making it more secure than other options. For as long as the private key stays private. To create a service account follow these steps:

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



7. You will now be prompted to download a .json file. This file contains the necessary private key for account authorization. Remember where you stored the file and how you named it.



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",
```

(continues on next page)

(continued from previous page)

```
    "client_id": "10.....454",
}
```

8. The authorization process can be completed without any further interactions:

```
gc = pygsheets.authorize(service_file='path/to/service_account_credentials.json')
```

6.1.3 Environment Variables

For services like Heroku that recommend using Twelve-Factor App principles, e.g. for storing config data in the environment instead of files, you can pass in Service Account credentials via an environment variable:

```
gc = pygsheets.authorize(service_account_env_var = 'GDRIVE_API_CREDENTIALS')
```

This assumes you have set an environment variable key *GDRIVE_API_CREDENTIALS* set with the value of the Service Account .json file described in the above Service Account section.

6.1.4 Custom Credentials Objects

You can create your own authentication method and pass them like this:

```
gc = pygsheets.authorize(custom_credentials=my_credentials)
```

This option will ignore any other parameters.

An example of creating credentials from secrect dict.:

```
SCOPES = ('https://www.googleapis.com/auth/spreadsheets', 'https://www.googleapis.com/
↪auth/drive')
service_account_info = json.loads(secret_dict)
my_credentials = service_account.Credentials.from_service_account_info(service_
↪account_info, scopes=SCOPES)
gc = pygsheets.authorize(custom_credentials=my_credentials)
```

6.2 pygsheets Reference

pygsheets is a simple Google Sheets API v4 Wrapper. Some functionality uses the Google Drive API v3 as well.

6.2.1 Authorization

pygsheets.**authorize**(*client_secret='client_secret.json'*, *service_account_file=None*, *service_account_env_var=None*, *service_account_json=None*, *credentials_directory=''*, *scopes=('https://www.googleapis.com/auth/spreadsheets'*, *'https://www.googleapis.com/auth/drive')*, *custom_credentials=None*, *local=False*, ***kwargs*)

Authenticate this application with a google account.

See general authorization documentation for details on how to attain the necessary files.

Parameters

- **client_secret** – Location of the oauth2 credentials file.
- **service_account_file** – Location of a service account file.
- **service_account_env_var** – Use an environment variable to provide service account credentials.
- **service_account_json** – pass in json string directly; could use aws secret manager or azure key vault to store value
- **credentials_directory** – Location of the token file created by the OAuth2 process. Use 'global' to store in global location, which is OS dependent. Default None will store token file in current working directory. Please note that this is override your client secret.
- **custom_credentials** – A custom or pre-made credentials object. Will ignore all other params.
- **scopes** – The scopes for which the authentication applies.
- **local** – If local then a browser will be opened to autheticate
- **kwargs** – Parameters to be handed into the client constructor.

Returns `Client`

Warning: The *credentials_directory* overrides *client_secret*. So you might be accidently using a different credential than intended, if you are using global *credentials_directory* in more than one script.

6.2.2 Client

class pygsheets.client.**Client**(*credentials*, *retries=3*, *http=None*, *check=True*, *seconds_per_quota=100*)

Create or access Google spreadsheets.

Exposes members to create new spreadsheets or open existing ones. Use *authorize* to instantiate an instance of this class.

```
>>> import pygsheets
>>> c = pygsheets.authorize()
```

The sheet API service object is stored in the sheet property and the drive API service object in the drive property.

```
>>> c.sheet.get('<SPREADSHEET ID>')
>>> c.drive.delete('<FILE ID>')
```

Parameters

- **credentials** – The credentials object returned by google-auth or google-auth-oauthlib.
- **retries** – (Optional) Number of times to retry a connection before raising a TimeOut error. Default: 3
- **http** – The underlying HTTP object to use to make requests. If not specified, a `httplib2.Http` instance will be constructed.
- **check** – Check for quota error and apply rate limiting.
- **seconds_per_quota** – Default value is 100 seconds

spreadsheet_ids (*query=None*)
Get a list of all spreadsheet ids present in the Google Drive or TeamDrive accessed.

spreadsheet_titles (*query=None*)
Get a list of all spreadsheet titles present in the Google Drive or TeamDrive accessed.

create (*title*, *template=None*, *folder=None*, *folder_name=None*, ***kwargs*)
Create a new spreadsheet.

The title will always be set to the given value (even overwriting the templates title). The template can either be a spreadsheet resource or an instance of *Spreadsheet*. In both cases undefined values will be ignored.

Parameters

- **title** – Title of the new spreadsheet.
- **template** – A template to create the new spreadsheet from.
- **folder** – The Id of the folder this sheet will be stored in.
- **folder_name** – The Name of the folder this sheet will be stored in.
- **kwargs** – Standard parameters (see reference for details).

Returns *Spreadsheet*

open (*title*)
Open a spreadsheet by title.

In a case where there are several sheets with the same title, the first one found is returned.

```
>>> import pygsheets
>>> c = pygsheets.authorize()
>>> c.open('TestSheet')
```

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

Returns *Spreadsheet*

Raises ***pygsheets.SpreadsheetNotFound*** – No spreadsheet with the given title was found.

open_by_key (*key*)
Open a spreadsheet by key.

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

Parameters **key** – The key of a spreadsheet. (can be found in the sheet URL)

Returns *Spreadsheet*

Raises ***pygsheets.SpreadsheetNotFound*** – The given spreadsheet ID was not found.

open_by_url (*url*)
Open a spreadsheet by URL.

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

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

Returns *Spreadsheet*

Raises ***pygsheets.SpreadsheetNotFound*** – No spreadsheet was found with the given URL.

open_all (*query=''*)
Opens all available spreadsheets.

Result can be filtered when specifying the query parameter. On the details on how to form the query:

Reference

Parameters **query** – (Optional) Can be used to filter the returned metadata.

Returns A list of *Spreadsheet*.

open_as_json (*key*)
Return a json representation of the spreadsheet.

See Reference for details.

get_range (*spreadsheet_id*, *value_range=None*, *major_dimension='ROWS'*, *value_render_option=<ValueRenderOption.FORMATTED_VALUE: 'FORMATTED_VALUE'>*, *date_time_render_option=<DateTimeRenderOption.SERIAL_NUMBER: 'SERIAL_NUMBER'>*, *value_ranges=None*)
Returns a range of values from a spreadsheet. The caller must specify the spreadsheet ID and a range.

Reference: request

Parameters

- **spreadsheet_id** – The ID of the spreadsheet to retrieve data from.
- **value_range** – The A1 notation of the values to retrieve.
- **value_ranges** – The list of A1 notation of the values to retrieve.
- **major_dimension** – The major dimension that results should use. For example, if the spreadsheet data is: A1=1,B1=2,A2=3,B2=4, then requesting range=A1:B2,majorDimension=ROWS will return [[1,2],[3,4]], whereas requesting range=A1:B2,majorDimension=COLUMNS will return [[1,3],[2,4]].
- **value_render_option** – How values should be represented in the output. The default render option is *ValueRenderOption.FORMATTED_VALUE*.
- **date_time_render_option** – How dates, times, and durations should be represented in the output. This is ignored if *valueRenderOption* is *FORMATTED_VALUE*. The default dateTime render option is [*DateTimeRenderOption.SERIAL_NUMBER*].

Returns An array of arrays with the values fetched. Returns an empty array if no values were fetched. Values are dynamically typed as int, float or string.

6.2.3 Models

Python objects for the main Google Sheets API Resources: *spreadsheet*, *worksheet*, *cell* and datarange.

Spreadsheet

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

worksheet_cls
alias of `pygsheets.worksheet.Worksheet`

id
Id of the spreadsheet.

title
Title of the spreadsheet.

locale
Locale of the spreadsheet.

sheet1
Direct access to the first worksheet.

url
Url of the spreadsheet.

named_ranges
All named ranges in this spreadsheet.

protected_ranges
All protected ranges in this spreadsheet.

defaultformat
Default cell format used.

updated
Last time the spreadsheet was modified using RFC 3339 format.

update_properties()
Update the sheet properties in cloud

fetch_properties(*jsonsheet=None*, *fetch_sheets=True*)
Update all properties of this spreadsheet with the remote.

The provided json representation must be the same as the Google Sheets v4 Response. If no sheet is given this will simply fetch all data from remote and update the local representation.

Reference: https://developers.google.com/sheets/api/reference/rest/v4/spreadsheets

Parameters

- **jsonsheet** – Used to update the spreadsheet.
- **fetch_sheets** – Fetch sheets from remote.

worksheets(*sheet_property=None*, *value=None*, *force_fetch=False*)
Get worksheets matching the specified property.

Parameters

- **sheet_property** – Property used to filter ('title', 'index', 'id').
- **value** – Value of the property.
- **force_fetch** – Fetch data from remote.

Returns List of *Worksheets*.

worksheet(*property='index'*, *value=0*)
Returns the worksheet with the specified index, title or id.

If several worksheets with the same property are found the first is returned. This may not be the same worksheet every time.

Example: Getting a worksheet named 'Annual bonuses'

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

Parameters

- **property** – The searched property.
- **value** – Value of the property.

Returns *Worksheets*.

worksheet_by_title(*title*)
Returns worksheet by title.

Parameters **title** – Title of the sheet

Returns *Worksheets*.

add_worksheet(*title*, *rows=100*, *cols=26*, *src_tuple=None*, *src_worksheet=None*, *index=None*)
Creates or copies a worksheet and adds it to this spreadsheet.

When creating only a title is needed. Rows & columns can be adjusted to match your needs. Index can be specified to set position of the sheet.

When copying another worksheet supply the spreadsheet id & worksheet id and the worksheet wrapped in a Worksheet class.

Parameters

- **title** – Title of the worksheet.
- **rows** – Number of rows which should be initialized (default 100)
- **cols** – Number of columns which should be initialized (default 26)
- **src_tuple** – Tuple of (spreadsheet id, worksheet id) specifying the worksheet to copy.
- **src_worksheet** – The source worksheet.
- **index** – Tab index of the worksheet.

Returns *Worksheets*.

del_worksheet(*worksheet*)
Deletes the worksheet from this spreadsheet.

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

replace(*pattern*, *replacement=None*, ***kwargs*)
Replace values in any cells matched by pattern in all worksheets.

Keyword arguments not specified will use the default value. If the spreadsheet is -

Unlinked: Uses *self.find(pattern, **kwargs)* to find the cells and then replace the values in each cell.

Linked: The replacement will be done by a findReplaceRequest as defined by the Google Sheets API. After the request the local copy is updated.

Parameters

- **pattern** – Match cell values.
- **replacement** – Value used as replacement.

- **searchByRegex** – Consider pattern a regex pattern. (default False)
- **matchCase** – Match case sensitive. (default False)
- **matchEntireCell** – Only match on full match. (default False)
- **includeFormulas** – Match fields with formulas too. (default False)

find(*pattern*, ***kwargs*)
Searches through all worksheets.

Search all worksheets with the options given. If an option is not given, the default will be used. Will return a list of cells for each worksheet packed into a list. If a worksheet has no cell which matches pattern an empty list is added.

Parameters

- **pattern** – The value to search.
- **searchByRegex** – Consider pattern a regex pattern. (default False)
- **matchCase** – Match case sensitive. (default False)
- **matchEntireCell** – Only match on full match. (default False)
- **includeFormulas** – Match fields with formulas too. (default False)

Returns A list of lists of *Cells*

share(*email_or_domain*, *role='reader'*, *type='user'*, ***kwargs*)
Share this file with a user, group or domain.

User and groups need an e-mail address and domain needs a domain for a permission. Share sheet with a person and send an email message.

```
>>> spreadsheet.share('example@gmail.com', role='commenter', type='user',␣
↪emailMessage='Here is the spreadsheet we talked about!')
```

Make sheet public with read only access:

```
>>> spreadsheet.share('', role='reader', type='anyone')
```

Parameters

- **email_or_domain** – The email address or domain this file should be shared to.
- **role** – The role of the new permission.
- **type** – The type of the new permission.
- **kwargs** – Optional arguments. See DriveAPIWrapper.create_permission documentation for details.

permissions
Permissions for this file.

remove_permission(*email_or_domain*, *permission_id=None*)
Remove a permission from this sheet.

All permissions associated with this email or domain are deleted.

Parameters

- **email_or_domain** – Email or domain of the permission.

- **permission_id** – (optional) permission id if a specific permission should be deleted.

export (*file_format=<ExportType.CSV: 'text/csv:.csv'>*, *path=''*, *filename=''*)
Export all worksheets.

The filename must have an appropriate file extension. Each sheet will be exported into a separate file. The filename is extended (before the extension) with the index number of the worksheet to not overwrite each file.

Parameters

- **file_format** – ExportType.<?>
- **path** – Path to the directory where the file will be stored. (default: current working directory)
- **filename** – Filename (default: spreadsheet id)

delete()
Deletes this spreadsheet.

Leaves the local copy intact. The deleted spreadsheet is permanently removed from your drive and not moved to the trash.

get_developer_metadata (*key=None*, *search_sheets=False*)
Fetch developer metadata associated with this spreadsheet

Parameters

- **key** – The key of the metadata to fetch. If unspecified, all metadata will be returned
- **search_sheets** – Set to True to also include worksheets in the metadata search

create_developer_metadata (*key*, *value=None*)
Create a new developer metadata associated with this spreadsheet

Will return None when in batch mode, otherwise will return a DeveloperMetadata object

Parameters

- **key** – the key of the metadata to be created
- **value** – the value of the metadata to be created (optional)

custom_request (*request*, *fields*, ***kwargs*)
Send a custom batch update request to this spreadsheet.

These requests have to be properly constructed. All possible requests are documented in the reference.

Reference: api docs <https://developers.google.com/sheets/api/reference/rest/v4/spreadsheets/request>`__

Parameters

- **request** – One or several requests as dictionaries.
- **fields** – Fields which should be included in the response.
- **kwargs** – Any other params according to refrence.

Returns json response <https://developers.google.com/sheets/api/reference/rest/v4/spreadsheets/response> __

to_json()
Return this spreadsheet as json resource.

Worksheet

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

Parameters

- **spreadsheet** – Spreadsheet object to which this worksheet belongs to
- **jsonSheet** – Contains properties to initialize this worksheet.

  Ref to api details for more info

id
The ID of this worksheet.

index
The index of this worksheet

title
The title of this worksheet.

hidden
Mark the worksheet as hidden.

url
The url of this worksheet.

rows
Number of rows active within the sheet. A new sheet contains 1000 rows.

cols
Number of columns active within the sheet.

frozen_rows
Number of frozen rows.

frozen_cols
Number of frozen columns.

merged_ranges
Ranges of merged cells in this sheet.

linked
If the sheet is linked.

refresh(*update_grid=False*)
refresh worksheet data

link(*syncToCloud=True*)
Link the spreadsheet with cloud, so all local changes will be updated instantly. Data fetches will work only in linked sheet. All the data update calls made when sheet is unlinked will be replayed on linking.

Parameters **syncToCloud** – update the cloud with local changes (data_grid), cached update calls if set to true update the local copy with cloud if set to false

unlink(*save_grid=True*)
Unlink the spreadsheet with cloud, so that any changes made wont be updated instantaneously. All the data update calls are cached and will be called once the sheet is linked again.

> **Warning:** After unlinking, functions which return data won't work.

sync()
sync the worksheet (datagrid, and worksheet properties) to cloud

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

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

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** – can be 'matrix', 'cell', 'range' the corresponding type will be returned

get_value(*addr*, *value_render=<ValueRenderOption.FORMATTED_VALUE: 'FORMATTED_VALUE'>*)
value of a cell at given address

Parameters

- **addr** – cell address as either tuple or label
- **value_render** – how the output values should rendered. api docs

get_values(*start*, *end*, *returnas='matrix'*, *majdim='ROWS'*, *include_tailing_empty=True*, *include_tailing_empty_rows=False*, *value_render=<ValueRenderOption.FORMATTED_VALUE: 'FORMATTED_VALUE'>*, *date_time_render_option=<DateTimeRenderOption.SERIAL_NUMBER: 'SERIAL_NUMBER'>*, *grange=None*, ***kwargs*)
Returns a range of values from start cell to end cell. It will fetch these values from remote and then processes them. Will return either a simple list of lists, a list of Cell objects or a DataRange object with all the cells inside.

Parameters

- **start** – Top left position as tuple or label
- **end** – Bottom right position as tuple or label
- **grange** – give range as grid range object, object of *GridRange*
- **majdim** – The major dimension of the matrix. ('ROWS') ('COLUMNS' not implemented)
- **returnas** – The type to return the fetched values as. ('matrix', 'cell', 'range')
- **include_tailing_empty** – whether to include empty trailing cells/values after last non-zero value in a row
- **include_tailing_empty_rows** – whether to include tailing rows with no values; if include_tailing_empty is false, will return unfilled list for each empty row, else will return rows filled with empty cells

- **value_render** – how the output values should rendered. api docs
- **date_time_render_option** – How dates, times, and durations should be represented in the output. This is ignored if *valueRenderOption* is *FORMATTED_VALUE*. The default dateTime render option is [*DateTimeRenderOption.SERIAL_NUMBER*].

Returns 'range': `DataRange` 'cell': [*Cell*] 'matrix': [[...], [...], ...] append '-unlinked' to get unlinked objects

get_values_batch(*ranges*, *majdim='ROWS'*, *value_render=<ValueRenderOption.FORMATTED_VALUE: 'FORMATTED_VALUE'>*, *date_time_render_option=<DateTimeRenderOption.SERIAL_NUMBER: 'SERIAL_NUMBER'>*, ***kwargs*)

Returns a range of values from start cell to end cell. It will fetch these values from remote and then processes them. Will return either a simple list of lists, a list of Cell objects or a DataRange object with all the cells inside.

Parameters

- **ranges** – list of ranges to get data as - objects of *GridRange*, or tuple (with start and end) or A1 notation string or dict in GridRange format
- **majdim** – The major dimension of the matrix. ('ROWS') ('COLMUNS' not implemented)
- **value_render** – refer get_values
- **date_time_render_option** – refer get_values

Example:

```
>>> wks.get_values_batch( ['A1:A2', 'C1:C2'] )
  [ [['3'], ['4']], [['c']]]
>>> wks.get_values_batch( [('1', None), ('5', None)] )
[ <values of row 1>, <values of row 5> ]
>>> wks.get_values_batch( [('A1', 'B2'), ('5', None), 'Sheet1!D1:F10', 'A'])
[ <values list of lists> ]
```

get_all_values(*returnas='matrix'*, *majdim='ROWS'*, *include_tailing_empty=True*, *include_tailing_empty_rows=True*, ***kwargs*)

Returns a list of lists containing all cells' values as strings.

Parameters

- **majdim** – output as row wise or column-wise
- **returnas** (*'matrix', 'cell', 'range*) – return as list of strings of cell objects
- **include_tailing_empty** – whether to include empty trailing cells/values after last non-zero value
- **include_tailing_empty_rows** – whether to include rows with no values; if include_tailing_empty is false, will return unfilled list for each empty row, else will return rows filled with empty string
- **kwargs** – all parameters of *pygsheets.Worksheet.get_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*, *majdim='ROWS'*, *numericise_data=True*, ***kwargs*)
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 which row to use as keys, starting from 1 following the numeration of the spreadsheet.
- **majdim** – ROW or COLUMN major form
- **numericise_data** – determines if data is converted to numbers or left as string
- **kwargs** – all parameters of *pygsheets.Worksheet.get_values()*

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

Warning: Will work nicely only if there is a single table in the sheet

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

Empty cells in this list will be rendered as empty strings .

Parameters

- **include_tailing_empty** – whether to include empty trailing cells/values after last non-zero value
- **row** – index of row
- **kwargs** – all parameters of *pygsheets.Worksheet.get_values()*
- **returnas** – ('matrix', 'cell', 'range') return as cell objects or just 2d array or range object

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

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

Parameters

- **include_tailing_empty** – whether to include empty trailing cells/values after last non-zero value
- **col** – index of col
- **kwargs** – all parameters of *pygsheets.Worksheet.get_values()*
- **returnas** – ('matrix' or 'cell' or 'range') return as cell objects or just values

get_gridrange(*start*, *end*)
get a range in gridrange format

Parameters

- **start** – start address

- **end** – end address

update_value(*addr*, *val*, *parse=None*)
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 is else as if the user typed them into the UI default is spreadsheet.default_parse

Example:

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

update_values(*crange=None*, *values=None*, *cell_list=None*, *extend=False*, *majordim='ROWS'*, *parse=None*)
Updates a range cell values, it can take either a cell list or a range and its values. cell list is only efficient for small lists. This will only update the cell values not other properties.

Parameters

- **cell_list** – List of a *Cell* objects to update with their values. If you pass a matrix to this, then it is assumed that the matrix is continuous (range), and will just update values based on label of top left and bottom right cells.
- **crange** – range in format A1:A2 or just 'A1' or even (1,2) end cell will be inferred 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. Default is spreadsheet.default_parse

update_values_batch(*ranges*, *values*, *majordim='ROWS'*, *parse=None*)
update multiple ranges of values in a single call.

Parameters

- **ranges** – list of addresses of the range. can be GridRange, label, tuple, etc
- **values** – list of values corresponding to ranges, should be list of matrices
- **majordim** – major dimension of values provided. 'ROWS' or 'COLUMNS'
- **parse** – if the values should be as if the user typed them into the UI else its stored as is. Default is spreadsheet.default_parse

Example: >>> wks.update_values_batch(['A1:A2', 'B1:B2'], [[[1],[2]], [[3],[4]]]) >>> wks.get_values_batch(['A1:A2', 'B1:B2']) [[['1'], ['2']], [['3'], ['4']]]

```
>>> wks.update_values_batch([((1,1), (2,1)), 'B1:B2'], [[[1,2]], [[3,4]]],
↪'COLUMNS')
>>> wks.get_values_batch(['A1:A2', 'B1:B2'])
[[['1'], ['2']], [['3'], ['4']]]
```

update_cells(*cell_list*, *fields='*'*)
update cell properties and data from a list of cell objects

Parameters

- **cell_list** – list of cell objects
- **fields** – cell fields to update, in google FieldMask format

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 number of rows.
- **cols** – New number of columns.

add_rows(*rows*)
Adds new rows to this worksheet.

Parameters **rows** – How many rows to add (integer)

add_cols(*cols*)
Add new columns to this worksheet.

Parameters **cols** – How many columns to add (integer)

delete_cols(*index*, *number=1*)
Delete 'number' of columns from index.

Parameters

- **index** – Index of first column to delete
- **number** – Number of columns to delete

delete_rows(*index*, *number=1*)
Delete 'number' of rows 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 new columns after 'col' and initialize all cells with values. Increases the number of rows if there are more values in values than rows.

Reference: insert request

Parameters

- **col** – Index of the col at which the values will be inserted.
- **number** – Number of columns to be inserted.
- **values** – Content to be inserted into new columns.
- **inherit** – New cells will inherit properties from the column to the left (True) or to the right (False).

insert_rows (*row*, *number=1*, *values=None*, *inherit=False*)
Insert a new row after 'row' and initialize all cells with values.

Widens the worksheet if there are more values than columns.

Reference: insert request

Parameters

- **row** – Index of the row at which the values will be inserted.
- **number** – Number of rows to be inserted.
- **values** – Content to be inserted into new rows.
- **inherit** – New cells will inherit properties from the row above (True) or below (False).

clear (*start='A1'*, *end=None*, *fields='userEnteredValue'*)
Clear all values in worksheet. Can be limited to a specific range with start & end.

Fields specifies which cell properties should be cleared. Use "*" to clear all fields.

Reference:

- CellData Api object
- FieldMask Api object

Parameters

- **start** – Top left cell label.
- **end** – Bottom right cell label.
- **fields** – Comma separated list of field masks.

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

Parameters

- **start** – Index of the first column to be widened.
- **end** – Index of the last column to be widened.
- **pixel_size** – New width in pixels or None to set width automatically based on the size of the column content.

apply_format (*ranges*, *format_info*, *fields='userEnteredFormat'*)
apply formatting for for multiple ranges

Parameters

- **ranges** – list of ranges (any type) to apply the formats to
- **format_info** – list or single pygsheets cell or dict of properties specifying the formats to be updated, see this for available options. if a list is given it should match size of ranges.
- **fields** – fields to be updated in the cell

Example: >>> wks.apply_format('A1:A10', {"numberFormat": {"type": "NUMBER"}}) >>> wks.apply_format('A1:A10', "TEXT") # by default number format is assumed >>> wks.apply_format(ranges=['A1:B1', 'D:E'], format_info={'numberFormat': {"type": "NUMBER"}}) >>> mcell = Cell('A1') # dummy cell >>> mcell.format = (pygsheets.FormatType.PERCENT, '') >>> wks.apply_format(ranges=['A1:B1', 'D:E'], format_info=mcell)

update_dimensions_visibility(*start*, *end=None*, *dimension='ROWS'*, *hidden=True*)
Hide or show one or more rows or columns.

Parameters

- **start** – Index of the first row or column.
- **end** – Index of the last row or column.
- **dimension** – 'ROWS' or 'COLUMNS'
- **hidden** – Hide rows or columns

hide_dimensions(*start*, *end=None*, *dimension='ROWS'*)
Hide one ore more rows or columns.

Parameters

- **start** – Index of the first row or column.
- **end** – Index of the last row or column.
- **dimension** – 'ROWS' or 'COLUMNS'

show_dimensions(*start*, *end=None*, *dimension='ROWS'*)
Show one ore more rows or columns.

Parameters

- **start** – Index of the first row or column.
- **end** – Index of the last row or column.
- **dimension** – 'ROWS' or 'COLUMNS'

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

Parameters

- **start** – Index of first row to be heightened.
- **end** – Index of last row to be heightened.
- **pixel_size** – New height in pixels or None to set height automatically based on the size of the row content.

append_table(*values*, *start='A1'*, *end=None*, *dimension='ROWS'*, *overwrite=False*, ***kwargs*)
Append a row or column of values to an existing table in the sheet. The input range is used to search for existing data and find a "table" within that range. Values will be appended to the next row of the table, starting with the first column of the table. The return value contains the index of the appended table. It is useful to get the index of last row or last column.

Reference: request

Parameters

- **values** – List of values for the new row or column.
- **start** – Top left cell of the range (requires a label).
- **end** – Bottom right cell of the range (requires a label).
- **dimension** – Dimension to which the values will be added ('ROWS' or 'COLUMNS')
- **overwrite** – If true will overwrite data present in the spreadsheet. Otherwise will create new rows to insert the data into.

Returns

A :class:dict containing the result of request

replace(*pattern*, *replacement=None*, ***kwargs*)

Replace values in any cells matched by pattern in this worksheet. Keyword arguments not specified will use the default value.

If the worksheet is

- **Unlinked** : Uses *self.find(pattern, **kwargs)* to find the cells and then replace the values in each cell.
- **Linked** : The replacement will be done by a findReplaceRequest as defined by the Google Sheets API. After the request the local copy is updated.

Reference: request

Parameters

- **pattern** – Match cell values.
- **replacement** – Value used as replacement.
- **searchByRegex** – Consider pattern a regex pattern. (default False)
- **matchCase** – Match case sensitive. (default False)
- **matchEntireCell** – Only match on full match. (default False)
- **includeFormulas** – Match fields with formulas too. (default False)

find(*pattern*, *searchByRegex=False*, *matchCase=False*, *matchEntireCell=False*, *includeFormulas=False*, *cols=None*, *rows=None*, *forceFetch=True*)

Finds all cells matched by the pattern.

Compare each cell within this sheet with pattern and return all matched cells. All cells are compared as strings. If replacement is set, the value in each cell is set to this value. Unless full_match is False in in which case only the matched part is replaced.

Note:

- Formulas are searched as their calculated values and not the actual formula.
- Find fetches all data and then run a linear search on then, so this will be slow if you have a large sheet

---

Parameters

- **pattern** – A string pattern.
- **searchByRegex** – Compile pattern as regex. (default False)

- **matchCase** – Comparison is case sensitive. (default False)
- **matchEntireCell** – Only match a cell if the pattern matches the entire value. (default False)
- **includeFormulas** – Match cells with formulas. (default False)
- **rows** – Range of rows to search in as tuple, example (2, 10)
- **cols** – Range of columns to search in as tuple, example (3, 10)
- **forceFetch** – If the offline data should be updated before search. (default False)

Returns A list of *Cells*.

create_named_range(*name*, *start=None*, *end=None*, *grange=None*, *returnas='range'*)
Create a new named range in this worksheet. Provide either start and end or grange.

Reference: Named range Api object

Parameters

- **name** – Name of the range.
- **start** – Top left cell address (label or coordinates)
- **end** – Bottom right cell address (label or coordinates)
- **grange** – grid range, object of *GridRange*

Returns DataRange

get_named_range(*name*)
Get a named range by name.

Reference: Named range Api object

Parameters **name** – Name of the named range to be retrieved.

Returns DataRange

Raises **RangeNotFound** – if no range matched the name given.

get_named_ranges(*name=''*)
Get named ranges from this worksheet.

Reference: Named range Api object

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

Returns DataRange

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

Reference: Named range Api object

Parameters

- **name** – Name of the range.
- **range_id** – Id of the range (optional)

create_protected_range(*start=None*, *end=None*, *grange=None*, *named_range_id=None*, *returnas='range'*)
Create protected range. Provide either start and end or grange.

Reference: Protected range Api object

Parameters

- **start** – address of the topleft cell
- **end** – address of the bottomright cell
- **grange** – grid range to protect, object of *GridRange*
- **named_range_id** – id of named range to protect
- **returnas** – 'json' or 'range'

remove_protected_range(*range_id*)
Remove protected range.

Reference: Protected range Api object

Parameters **range_id** – ID of the protected range.

get_protected_ranges()
returns protected ranges in this sheet

Returns Protected range objects

Return type Datarange

set_dataframe(*df*, *start*, *copy_index=False*, *copy_head=True*, *extend=False*, *fit=False*, *escape_formulae=False*, ***kwargs*)
Load sheet from Pandas Dataframe.

Will load all data contained within the Pandas data frame into this worksheet. It will begin filling the worksheet at cell start. Supports multi index and multi header datarames.

Parameters

- **df** – Pandas data frame.
- **start** – Address of the top left corner where the data should be added.
- **copy_index** – Copy data frame index (multi index supported).
- **copy_head** – Copy header data into first row.
- **extend** – Add columns and rows to the worksheet if necessary, but won't delete any rows or columns.
- **fit** – Resize the worksheet to fit all data inside if necessary.
- **escape_formulae** – Any value starting with an equal or plus sign (=/+), will be prefixed with an apostrophe (') to avoid value being interpreted as a formula.
- **nan** – Value with which NaN values are replaced. by default it will be replaced with string 'nan'. for converting nan values to empty cells set nan="".

get_as_df(*has_header=True*, *index_column=None*, *start=None*, *end=None*, *numerize=True*, *empty_value=''*, *value_render=<ValueRenderOption.FORMATTED_VALUE: 'FORMATTED_VALUE'>*, ***kwargs*)
Get the content of this worksheet as a pandas data frame.

Parameters

- **has_header** – Interpret first row as data frame header.
- **index_column** – Column to use as data frame index (integer).
- **numerize** – Numerize cell values.
- **empty_value** – Placeholder value to represent empty cells when numerizing.

- **start** – Top left cell to load into data frame. (default: A1)
- **end** – Bottom right cell to load into data frame. (default: (rows, cols))
- **value_render** – How the output values should returned, api docs By default, will convert everything to strings. Setting as UNFORMATTED_VALUE will do numerizing, but values will be unformatted.
- **include_tailing_empty** – whether to include empty trailing cells/values after last non-zero value in a row
- **include_tailing_empty_rows** – whether to include tailing rows with no values; if include_tailing_empty is false, will return unfilled list for each empty row, else will return rows filled with empty cells

Returns pandas.Dataframe

export (*file_format=<ExportType.CSV: 'text/csv:.csv'>*, *filename=None*, *path=''*)
Export this worksheet to a file.

Note:

- Only CSV & TSV exports support single sheet export. In all other cases the entire spreadsheet will be exported.
- This can at most export files with 10 MB in size!

Parameters

- **file_format** – Target file format (default: CSV), enum :class:<pygsheets.ExportType>
- **filename** – Filename (default: spreadsheet id + worksheet index).
- **path** – Directory the export will be stored in. (default: current working directory)

copy_to (*spreadsheet_id*)
Copy this worksheet to another spreadsheet.

This will copy the entire sheet into another spreadsheet and then return the new worksheet. Can be slow for huge spreadsheets.

Reference: request

Parameters **spreadsheet_id** – The id this should be copied to.

Returns Copy of the worksheet in the new spreadsheet.

sort_range (*start*, *end*, *basecolumnindex=0*, *sortorder='ASCENDING'*)
Sorts the data in rows based on the given column index.

Parameters

- **start** – Address of the starting cell of the grid.
- **end** – Address of the last cell of the grid to be considered.
- **basecolumnindex** – Index of the base column in which sorting is to be done (Integer), default value is 0. The index here is the index of the column in worksheet.
- **sortorder** – either "ASCENDING" or "DESCENDING" (String)

Example: If the data contain 5 rows and 6 columns and sorting is to be done in 4th column. In this case the values in other columns also change to maintain the same relative values.

add_chart(*domain*, *ranges*, *title=None*, *chart_type=<ChartType.COLUMN: 'COLUMN'>*, *anchor_cell=None*)

Creates a chart in the sheet and returns a chart object. The X-axis is called the domain and the Y-axis is called range. There can only be a single domain as it is the values against which the ranges are plotted.

You can have multiple ranges, and all of them will be plotted against the values on the domain (depending on chart_type).

For example, suppose you want to plot temperature against the years. Here Year will be the domain and Temperature will be a range. Now suppose you want to add a plot of rainfall also to this chart (given you have the same year range). You can just add the rainfall data as a range.

Parameters

- **domain** – Cell range of the desired chart domain (x-axis) in the form of tuple of adresses (start_address, end_address)
- **ranges** – Cell ranges of the desired ranges (y-axis) in the form of list of tuples of adresses
- **title** – Title of the chart
- **chart_type** – Basic chart type (default: COLUMN)
- **anchor_cell** – position of the left corner of the chart in the form of cell address or cell object

Returns *Chart*

Example:

To plot a chart with x values from 'A1' to 'A6' and y values from 'B1' to 'B6'

```
>>> wks.add_chart(('A1', 'A6'), [('B1', 'B6')], 'TestChart')
<Chart 'COLUMN' 'TestChart'>
```

get_charts(*title=None*)

Returns a list of chart objects, can be filtered by title.

Parameters **title** – title to be matched.

Returns list of *Chart*

set_data_validation(*start=None*, *end=None*, *condition_type=None*, *condition_values=None*, *grange=None*, ***kwargs*)

Sets a data validation rule to every cell in the range. To clear validation in a range, call this with no condition_type specified.

refer to api docs for possible inputs.

Parameters

- **start** – start address
- **end** – end address
- **grange** – address as grid range
- **condition_type** – validation condition type: possible values
- **condition_values** – list of values for supporting condition type. For example , when condition_type is NUMBER_BETWEEN, value should be two numbers indicationg lower and upper bound. See api docs for more info.
- **kwargs** – other options of rule. possible values: inputMessage, strict, showCustomUi ref

set_basic_filter(*start=None*, *end=None*, *grange=None*, *sort_order=None*, *sort_foreground_color=None*, *sort_background_color=None*, *sort_column_index=None*, *filter_column_index=None*, *hidden_values=None*, *condition_type=None*, *condition_values=None*, *filter_foreground_color=None*, *filter_background_color=None*)

Sets a basic filter to a row in worksheet.

refer to api docs for possible inputs.

Parameters

- **start** – start address
- **end** – end address
- **grange** – address as grid range
- **sort_order** – either "ASCENDING" or "DESCENDING" (String)
- **sort_foreground_color** – either Color obj (Tuple) or ThemeColorType (String). please refer to api docs for possible inputs.
- **sort_background_color** – either Color obj (Tuple) or ThemeColorType (String). please refer to api docs for possible inputs.
- **sort_column_index** – the position of column for sort.
- **filter_column_index** – the position of column for filter.
- **hidden_values** – values which are hidden by filter.
- **condition_type** – validation condition type: possible values
- **condition_values** – list of values for supporting condition type. For example , when condition_type is NUMBER_BETWEEN, value should be two numbers indicationg lower and upper bound. It also can be this enum. See api docs for more info.
- **filter_foreground_color** – either Color obj (Tuple) or ThemeColorType (String). please refer to api docs for possible inputs.
- **filter_background_color** – either Color obj (Tuple) or ThemeColorType (String). please refer to api docs for possible inputs.

clear_basic_filter()

Clear a basic filter in worksheet

refer to api docs for possible inputs.

add_conditional_formatting(*start*, *end*, *condition_type*, *format*, *condition_values=None*, *grange=None*)

Adds a new conditional format rule.

Parameters

- **start** – start address
- **end** – end address
- **grange** – address as grid range
- **condition_type** – validation condition type: possible values
- **condition_values** – list of values for supporting condition type. For example , when condition_type is NUMBER_BETWEEN, value should be two numbers indicationg lower and upper bound. It also can be this enum. See api docs for more info.

- **format** – cell format json to apply if condition succeedes. *refer. <https://developers.google.com/sheets/api/reference/rest/v4/spreadsheets/cells#CellFormat>*

merge_cells(*start=None*, *end=None*, *merge_type='MERGE_ALL'*, *grange=None*)
Merge cells in range

! You can't vertically merge cells that intersect an existing filter

Parameters

- **merge_type** – either 'MERGE_ALL' ,'MERGE_COLUMNS' (= merge multiple rows (!) together to make column(s)) ,'MERGE_ROWS' (= merge multiple columns (!) together to make a row(s)) ,'NONE' (unmerge)
- **start** – start Address
- **end** – end Address

get_developer_metadata(*key=None*)
Fetch developer metadata associated with this worksheet

Parameters **key** – the key of the metadata to fetch. If unspecified, all metadata will be returned

create_developer_metadata(*key*, *value=None*)
Create a new developer metadata associated with this worksheet

Will return None when in batch mode, otherwise will return a DeveloperMetadata object

Parameters

- **key** – the key of the metadata to be created
- **value** – the value of the metadata to be created (optional)

DataRange

class pygsheets.datarange.**DataRange**(*start=None*, *end=None*, *worksheet=None*, *name=''*, *data=None*, *name_id=None*, *namedjson=None*, *protectedjson=None*, *grange=None*)
DataRange specifies a range of cells in the sheet. It can be unbounded on one or more axes. DataRange is for storing/manipulating a range of data in worksheet. This class can be used for group operations, e.g. changing format of all cells in a given range. This can also represent named ranges protected ranges, banned ranges etc.

All the proteted range properties are stored in protected_properties.

Parameters

- **start** – top left cell address. can be unbounded.
- **end** – bottom right cell address
- **worksheet** – worksheet where this range belongs
- **name** – name of the named range
- **data** – data of the range in as row major matrix
- **name_id** – id of named range
- **namedjson** – json representing the NamedRange from api

```
>>> drange = Datarange(start='A1', end='B2', worksheet=wks)
<Datarange Sheet1!A1:B2>
>>> drange.name = "my_named_range" # make this datarange a named range
<Datarange my_named_range Sheet1!A1:B2>
>>> drange.protected = True # make the range protected
<Datarange my_named_range Sheet1!A1:B2 protected>
>>> drange.start_addr = 'B' # make the range unbounded on rows
<Datarange my_named_range Sheet1!A:B protected>
>>> drange.end_addr = None # make the range unbounded on both axes
<Datarange my_named_range Sheet1 protected>
```

name
name of the named range. setting a name will make this a range a named range setting this to empty string will delete the named range

name_id
if of the named range

protect_id
id of the protected range

protected
get/set the range as protected setting this to False will make this range unprotected

editors
Lists the editors of the protected range can also set a list of editors, take a tuple ('users' or 'groups', [<editors>]) can also set ('domainUsersCanEdit', Boolean)

requesting_user_can_edit
if the requesting user can edit protected range

description
if the requesting user can edit protected range

start_addr
top-left address of the range

end_addr
bottom-right address of the range

range
Range in format A1:C5

worksheet
linked worksheet

cells
Get cells of this range

link(*update=True*)
link the datarange so that all properties 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 it is changed

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

> **Warning:** Currently only data is fetched not properties, so *only_data* wont work

Parameters **only_data** – fetch only data

apply_format(*cell=None*, *fields=None*, *cell_json=None*)
Change format of all cells in the range

Parameters

- **cell** – a model :class: Cell whose format will be applied to all cells
- **fields** – comma seprated string of fields of cell to apply, refer to google api docs
- **cell_json** – if not providing a cell object, provide a cell json. refer to google api docs

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

Parameters **values** – values as matrix, which has same size as the range

sort(*basecolumnindex=0*, *sortorder='ASCENDING'*)
sort the values in the datarange

Parameters

- **basecolumnindex** – Index of the base column in which sorting is to be done (Integer). The index here is the index of the column in range (first columen is 0).
- **sortorder** – either "ASCENDING" or "DESCENDING" (String)

clear(*fields='userEnteredValue'*)
Clear values in this datarange.

Reference:

- FieldMask Api object

Parameters **fields** – Comma separated list of field masks.

update_named_range()
update the named range properties

update_protected_range(*fields='*'*)
update the protected range properties

update_borders(*top=False*, *right=False*, *bottom=False*, *left=False*, *inner_horizontal=False*, *inner_vertical=False*, *style='NONE'*, *width=1*, *red=0*, *green=0*, *blue=0*)
update borders for range

NB use style='NONE' to erase borders default color is black

Parameters

- **top** – make a top border
- **right** – make a right border
- **bottom** – make a bottom border
- **left** – make a left border
- **style** – either 'SOLID', 'DOTTED', 'DASHED', 'SOLID', 'SOLID_MEDIUM', 'SOLID_THICK', 'DOUBLE' or 'NONE' (String).

- **width** – border width (depreciated) (Integer).
- **red** – 0-255 (Integer).
- **green** – 0-255 (Integer).
- **blue** – 0-255 (Integer).

merge_cells(*merge_type='MERGE_ALL'*)

Merge cells in range

! You can't vertically merge cells that intersect an existing filter

Parameters **merge_type** – either 'MERGE_ALL' ,'MERGE_COLUMNS' (= merge multiple rows (!) together to make column(s)) ,'MERGE_ROWS' (= merge multiple columns (!) together to make a row(s)) ,'NONE' (unmerge)

Address

class pygsheets.**Address**(*value*, *allow_non_single=False*)

Represents the address of a cell. This can also be unbound in an axes. So 'A' is also a valid address but this requires explict setting of param *allow_non_single*. First index correspond to the rows, second index corresponds to columns. Integer Indexes start from 1.

```
>>> a = Address('A2')
>>> a.index
(2, 1)
>>> a.label
'A2'
>>> a[0]
2
>>> a[1]
1
>>> a = Address((1, 4))
>>> a.index
(1, 4)
>>> a.label
D1
>>> b = a + (3,0)
>>> b
<Address D4>
>>> b == (4, 4)
True
>>> column_a = Address((None, 1), True)
>>> column_a
<Address A>
>>> row_2 = Address('2', True)
>>> row_2
<Address 2>
```

label

Label of the current address in A1 format.

row

Row of the address

col

Column of the address

index
Current Address in tuple format. Both axes starts at 1.

class pygsheets.**GridRange**(*label=None*, *worksheet=None*, *start=None*, *end=None*, *worksheet_title=None*, *worksheet_id=None*, *propertiesjson=None*)

Represents a rectangular (can be unbounded) range of adresses on a sheet. All indexes are one-based and are closed, ie the start index and the end index is inclusive Missing indexes indicate the range is unbounded on that side.

A:B, A1:B3, 1:2 are all valid index, but A:1, 2:D are not

grange.start = (1, None) will make the range unbounded on column grange.indexes = ((None, None), (None, None)) will make the range completely unbounded, ie. whole sheet

Example:

```
>>> grange = GridRange(worksheet=wks, start='A1', end='D4')
>>> grange
<GridRange Sheet1!A1:D4>
>>> grange.start = 'A' # will remove bounding in rows
<GridRange Sheet1!A:D>
>>> grange.start = 'A1' # cannot add bounding at just start
<GridRange Sheet1!A:D>
>>> grange.indexes = ('A1', 'D4') # cannot add bounding at just start
<GridRange Sheet1!A1:D4>
>>> grange.end = (3, 5) # tuples will also work
<GridRange Sheet1!A1:C5>
>>> grange.end = (None, 5) # make unbounded on rows
<GridRange Sheet1!1:5>
>>> grange.end = (None, None) # make it unbounded on one index
<GridRange Sheet1!1:1>
>>> grange.start = None # make it unbounded on both indexes
<GridRange Sheet1>
>>> grange.start = 'A1' # make it unbounded on single index,now AZ100 is bottom␣
↪right cell of worksheet
<GridRange Sheet1:A1:AZ100>
>>> 'A1' in grange
True
>>> (100,100) in grange
False
>>> for address in grange:
>>>     print(address)
Address((1,1))
Address((1,2))
...
```

Reference: GridRange API docs

start
address of top left cell (index).

end
address of bottom right cell (index)

indexes
Indexes of this range as a tuple

label
Label in A1 notation format

worksheet_id
Id of woksheet this range belongs to

worksheet_title
Title of woksheet this range belongs to

static create(*data*, *wks=None*)

create a Gridrange from various type of data

Parameters

- **data** – can be string in A format,tuple or list, dict in GridRange format, GridRange object
- **wks** – worksheet to link to (optional)

Returns GridRange object

set_worksheet(*value*)
set the worksheet of this grid range.

to_json()
Get json representation of this grid range.

set_json(*namedjson*)
Apply a Gridrange json to this named range.

Parameters **namedjson** – json object of the GridRange format

Reference: GridRange docs

get_bounded_indexes()
get bounded indexes of this range based on worksheet size, if the indexes are unbounded

height
Height of this gridrange

width
Width of this gridrange

Cell

class pygsheets.**Cell**(*pos*, *val=''*, *worksheet=None*, *cell_data=None*)
Represents a single cell of a sheet.

Each cell is either a simple local value or directly linked to a specific cell of a sheet. When linked any changes to the cell will update the *Worksheet* immediately.

Parameters

- **pos** – Address of the cell as coordinate tuple or label.
- **val** – Value stored inside of the cell.
- **worksheet** – Worksheet this cell belongs to.
- **cell_data** – This cells data stored in json, with the same structure as cellData of the Google Sheets API v4.

borders = None
Border Properties as dictionary. Reference: api object.

parse_value = None
Determines how values are interpreted by Google Sheets (True: USER_ENTERED; False: RAW).

Reference: sheets api

row
Row number of the cell.

col
Column number of the cell.

label
This cells label (e.g. 'A1').

address
Address object representing the cell location.

value
This cells formatted value.

value_unformatted
Unformatted value of this cell.

formula
Get/Set this cells formula if any.

horizontal_alignment
Horizontal alignment of the value in this cell. possible vlaues: *HorizontalAlignment*

vertical_alignment
Vertical alignment of the value in this cell. possible vlaues: *VerticalAlignment*

wrap_strategy
How to wrap text in this cell. Possible wrap strategies: 'OVERFLOW_CELL', 'LEGACY_WRAP', 'CLIP', 'WRAP'. Reference: api docs

note
Get/Set note of this cell.

color
Get/Set background color of this cell as a tuple (red, green, blue, alpha).

simple
Simple cells only fetch the value itself. Set to false to fetch all cell properties.

set_text_format(*attribute*, *value*)
Set a text format property of this cell.

Each format property must be set individually. Any format property which is not set will be considered unspecified.

Attribute:

- foregroundColor: Sets the texts color. (tuple as (red, green, blue, alpha))
- fontFamily: Sets the texts font. (string)
- fontSize: Sets the text size. (integer)
- bold: Set/remove bold format. (boolean)
- italic: Set/remove italic format. (boolean)
- strikethrough: Set/remove strike through format. (boolean)
- underline: Set/remove underline format. (boolean)

Reference: api docs

Parameters

- **attribute** – The format property to set.
- **value** – The value the format property should be set to.

Returns *cell*

set_number_format(*format_type*, *pattern=''*)
Set number format of this cell.

Reference: api docs

Parameters

- **format_type** – The type of the number format. Should be of type `FormatType`.
- **pattern** – Pattern string used for formatting. If not set, a default pattern will be used. See reference for supported patterns.

Returns *cell*

set_text_rotation(*attribute*, *value*)
The rotation applied to text in this cell.

Can be defined as "angle" or as "vertical". May not define both!

angle: [number] The angle between the standard orientation and the desired orientation. Measured in degrees. Valid values are between -90 and 90. Positive angles are angled upwards, negative are angled downwards.

Note: For LTR text direction positive angles are in the counterclockwise direction, whereas for RTL they are in the clockwise direction.

vertical: [boolean] If true, text reads top to bottom, but the orientation of individual characters is unchanged.

Reference: *api_docs <https://developers.google.com/sheets/api/reference/rest/v4/spreadsheets#textrotation>__*

Parameters

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

Returns *cell*

set_horizontal_alignment(*value*)
Set horizondal alignemnt of text in the cell

Parameters **value** – Horizondal alignment value, instance of `HorizontalAlignment`

Returns *cell*

set_vertical_alignment(*value*)
Set vertical alignemnt of text in the cell

Parameters **value** – Vertical alignment value, instance of `VerticalAlignment`

Returns *cell*

set_value(*value*)
Set value of the cell

Parameters **value** – value to be set

Returns *cell*

unlink()
Unlink this cell from its worksheet.

Unlinked cells will no longer automatically update the sheet when changed. Use update or link to update the sheet.

link(*worksheet=None*, *update=False*)
Link cell with the specified worksheet.

Linked cells will synchronize any changes with the sheet as they happen.

Parameters

- **worksheet** – The worksheet to link to. Can be None if the cell was linked to a worksheet previously.
- **update** – Update the cell immediately after linking if the cell has changed

Returns *cell*

neighbour(*position*)
Get a neighbouring cell of this cell.

Parameters **position** – This may be a string 'right', 'left', 'top', 'bottom' or a tuple of relative positions (e.g. (1, 2) will return a cell one below and two to the right).

Returns *neighbouring cell*

fetch(*keep_simple=False*)
Update the value in this cell from the linked worksheet.

refresh()
Refresh the value and properties in this cell from the linked worksheet. Same as fetch.

update(*force=False*, *get_request=False*, *worksheet_id=None*)
Update the cell of the linked sheet or the worksheet given as parameter.

Parameters

- **force** – Force an update from the sheet, even if it is unlinked.
- **get_request** – Return the request object instead of sending the request directly.
- **worksheet_id** – Needed if the cell is not linked otherwise the cells worksheet is used.

get_json()
Returns the cell as a dictionary structured like the Google Sheets API v4.

set_json(*cell_data*)
Reads a json-dictionary returned by the Google Sheets API v4 and initialize all the properties from it.

Parameters **cell_data** – The cells data.

Chart

class pygsheets.**Chart**(*worksheet*, *domain=None*, *ranges=None*, *chart_type=None*, *title=''*, *anchor_cell=None*, *json_obj=None*)
Represents a chart in a sheet.

Parameters

- **worksheet** – Worksheet object in which the chart resides

- **domain** – Cell range of the desired chart domain in the form of tuple of tuples
- **ranges** – Cell ranges of the desired ranges in the form of list of tuple of tuples
- **chart_type** – An instance of `ChartType` Enum.
- **title** – Title of the chart
- **anchor_cell** – Position of the left corner of the chart in the form of cell address or cell object
- **json_obj** – Represents a json structure of the chart as given in api.

title
Title of the chart

domain
Domain of the chart. The domain takes the cell range in the form of tuple of cell adresses. Where first adress is the top cell of the column and 2nd element the last adress of the column.

Example: ((1,1),(6,1)) or ('A1','A6')

chart_type
Type of the chart The specificed as enum of type :class:'ChartType'

The available chart types are given in the api docs .

ranges
Ranges of the chart (y values) A chart can have multiple columns as range. So you can provide them as a list. The ranges are taken in the form of list of tuple of cell adresses. where each tuple inside the list represents a column as staring and ending cell.

Example: [((1,2),(6,2)), ((1,3),(6,3))] or [('B1','B6'), ('C1','C6')]

title_font_family
Font family of the title. (Default: 'Roboto')

font_name
Font name for the chart. (Default: 'Roboto')

legend_position
Legend postion of the chart. (Default: 'RIGHT_LEGEND') The available options are given in the api docs.

id
Id of the this chart.

anchor_cell
Position of the left corner of the chart in the form of cell address or cell object, Changing this will move the chart.

delete()
Deletes the chart.

> **Warning:** Once the chart is deleted the objects of that chart still exist and should not be used.

refresh()
Refreshes the object to incorporate the changes made in the chart through other objects or Google sheet

update_chart()
updates the applied changes to the sheet.

get_json()
Returns the chart as a dictionary structured like the Google Sheets API v4.

set_json(*chart_data*)
Reads a json-dictionary returned by the Google Sheets API v4 and initialize all the properties from it.

Parameters **chart_data** – The chart data as json specified in sheets api.

6.2.4 Helper Classes

The Drive API is wrapped by `DriveAPIWrapper`, and the Sheets API is wrapped by `SheetAPIWrapper`. They Only implements functionality used by this package. You would never need to access this directly.

Also there are many Enums defined for model properties or function parameters.

Custom Types

pygsheets.custom_types

This module contains common Enums used in pygsheets

class pygsheets.custom_types.**WorkSheetProperty**
available properties of worksheets

TITLE = 'title'

ID = 'id'

INDEX = 'index'

class pygsheets.custom_types.**ValueRenderOption**
Determines how values should be rendered in the output.

ValueRenderOption Docs

FORMATTED_VALUE: Values will be calculated & formatted in the reply according to the cell's formatting. Formatting is based on the spreadsheet's locale, not the requesting user's locale. For example, if A1 is 1.23 and A2 is =A1 and formatted as currency, then A2 would return "$1.23".

UNFORMATTED_VALUE : Values will be calculated, but not formatted in the reply. For example, if A1 is 1.23 and A2 is =A1 and formatted as currency, then A2 would return the number 1.23.

FORMULA : Values will not be calculated. The reply will include the formulas. For example, if A1 is 1.23 and A2 is =A1 and formatted as currency, then A2 would return "=A1".

FORMATTED_VALUE = 'FORMATTED_VALUE'

UNFORMATTED_VALUE = 'UNFORMATTED_VALUE'

FORMULA = 'FORMULA'

class pygsheets.custom_types.**DateTimeRenderOption**
Determines how dates should be rendered in the output.

DateTimeRenderOption Doc

SERIAL_NUMBER: Instructs date, time, datetime, and duration fields to be output as doubles in "serial number" format, as popularized by Lotus 1-2-3. The whole number portion of the value (left of the decimal) counts the days since December 30th 1899. The fractional portion (right of the decimal) counts the time as a fraction of the day. For example, January 1st 1900 at noon would be 2.5, 2 because it's 2 days after December 30st 1899,

and .5 because noon is half a day. February 1st 1900 at 3pm would be 33.625. This correctly treats the year 1900 as not a leap year.

FORMATTED_STRING: Instructs date, time, datetime, and duration fields to be output as strings in their given number format (which is dependent on the spreadsheet locale).

SERIAL_NUMBER = 'SERIAL_NUMBER'

FORMATTED_STRING = 'FORMATTED_STRING'

class pygsheets.custom_types.**FormatType**
Enum for cell formats.

CUSTOM = None

TEXT = 'TEXT'

NUMBER = 'NUMBER'

PERCENT = 'PERCENT'

CURRENCY = 'CURRENCY'

DATE = 'DATE'

TIME = 'TIME'

DATE_TIME = 'DATE_TIME'

SCIENTIFIC = 'SCIENTIFIC'

class pygsheets.custom_types.**ExportType**
Enum for possible export types

XLS = 'application/vnd.openxmlformats-officedocument.spreadsheetml.sheet:.xls'

ODT = 'application/x-vnd.oasis.opendocument.spreadsheet:.odt'

PDF = 'application/pdf:.pdf'

CSV = 'text/csv:.csv'

TSV = 'text/tab-separated-values:.tsv'

HTML = 'application/zip:.zip'

class pygsheets.custom_types.**HorizontalAlignment**
Horizontal alignment of the cell.

HorizontalAlignment doc

LEFT = 'LEFT'

RIGHT = 'RIGHT'

CENTER = 'CENTER'

NONE = None

class pygsheets.custom_types.**VerticalAlignment**
Vertical alignment of the cell.

VerticalAlignment doc

TOP = 'TOP'

MIDDLE = 'MIDDLE'

BOTTOM = 'BOTTOM'

NONE = None

class pygsheets.custom_types.**ChartType**
Enum for basic chart types

Reference: insert request

BAR = 'BAR'

LINE = 'LINE'

AREA = 'AREA'

COLUMN = 'COLUMN'

SCATTER = 'SCATTER'

COMBO = 'COMBO'

STEPPED_AREA = 'STEPPED_AREA'

Sheet API Wrapper

class pygsheets.sheet.**SheetAPIWrapper**(*http*, *data_path*, *seconds_per_quota=100*, *retries=1*, *logger=<Logger pygsheets.sheet (WARNING)>*, *check=True*)

batch_update(*spreadsheet_id*, *requests*, ***kwargs*)
Applies one or more updates to the spreadsheet.

Each request is validated before being applied. If any request is not valid then the entire request will fail and nothing will be applied.

Some requests have replies to give you some information about how they are applied. The replies will mirror the requests. For example, if you applied 4 updates and the 3rd one had a reply, then the response will have 2 empty replies, the actual reply, and another empty reply, in that order.

Due to the collaborative nature of spreadsheets, it is not guaranteed that the spreadsheet will reflect exactly your changes after this completes, however it is guaranteed that the updates in the request will be applied together atomically. Your changes may be altered with respect to collaborator changes. If there are no collaborators, the spreadsheet should reflect your changes.

Request body params	Description
includeSpreadsheetInResponse	Determines if the update response should include the spreadsheet resource. (default: False)
responseRanges[]	Limits the ranges included in the response spreadsheet. Only applied if the first param is True.
responseIncludeGridData	True if grid data should be returned. Meaningful only if if includeSpreadsheetInResponse is 'true'. This parameter is ignored if a field mask was set in the request.

Parameters

- **spreadsheet_id** – The spreadsheet to apply the updates to.
- **requests** – A list of updates to apply to the spreadsheet. Requests will be applied in the order they are specified. If any request is not valid, no requests will be applied.
- **kwargs** – Request body params & standard parameters (see reference for details).

Returns

create(*title*, *template=None*, ***kwargs*)

Create a spreadsheet.

Can be created with just a title. All other values will be set to default.

A template can be either a JSON representation of a Spreadsheet Resource as defined by the Google Sheets API or an instance of the Spreadsheet class. Missing fields will be set to default.

Parameters

- **title** – Title of the new spreadsheet.
- **template** – Template used to create the new spreadsheet.
- **kwargs** – Standard parameters (see reference for details).

Returns A Spreadsheet Resource.

get(*spreadsheet_id*, ***kwargs*)

Returns a full spreadsheet with the entire data.

The data returned can be limited with parameters. See reference for details .

Parameters

- **spreadsheet_id** – The Id of the spreadsheet to return.
- **kwargs** – Standard parameters (see reference for details).

Returns Return a SheetResource.

update_sheet_properties_request(*spreadsheet_id*, *properties*, *fields*)
Updates the properties of the specified sheet.

Properties must be an instance of SheetProperties.

Parameters

- **spreadsheet_id** – The id of the spreadsheet to be updated.
- **properties** – The properties to be updated.
- **fields** – Specifies the fields which should be updated.

Returns SheetProperties

developer_metadata_get(*spreadsheet_id*, *metadata_id*)
Returns a dictionary of developer metadata matching the supplied filter

Reference: request

Parameters

- **spreadsheet_id** – The id of the spreadsheet to search.
- **data_filter** – The id of the developer metadata item to get

developer_metadata_search(*spreadsheet_id*, *data_filter*)
Returns a dictionary of developer metadata matching the supplied filter

Reference: request

Parameters

- **spreadsheet_id** – The id of the spreadsheet to search.
- **data_filter** – A dictionary represeting a DeveloperMetadataLookup filter (see reference)

sheets_copy_to(*source_spreadsheet_id*, *worksheet_id*, *destination_spreadsheet_id*, ***kwargs*)
Copies a worksheet from one spreadsheet to another.

Reference: request

Parameters

- **source_spreadsheet_id** – The ID of the spreadsheet containing the sheet to copy.
- **worksheet_id** – The ID of the sheet to copy.
- **destination_spreadsheet_id** – The ID of the spreadsheet to copy the sheet to.
- **kwargs** – Standard parameters (see reference for details).

Returns SheetProperties

values_append(*spreadsheet_id*, *values*, *major_dimension*, *range*, ***kwargs*)
Appends values to a spreadsheet.

The input range is used to search for existing data and find a "table" within that range. Values will be appended to the next row of the table, starting with the first column of the table. See the guide and sample code for specific details of how tables are detected and data is appended.

The caller must specify the spreadsheet ID, range, and a valueInputOption. The valueInputOption only controls how the input data will be added to the sheet (column-wise or row-wise), it does not influence what cell the data starts being written to.

Reference: request

Parameters

- **spreadsheet_id** – The ID of the spreadsheet to update.
- **values** – The values to be appended in the body.
- **major_dimension** – The major dimension of the values provided (e.g. row or column first?)
- **range** – The A1 notation of a range to search for a logical table of data. Values will be appended after the last row of the table.
- **kwargs** – Query & standard parameters (see reference for details).

values_batch_clear(*spreadsheet_id*, *ranges*)

Clear values from sheet.

Clears one or more ranges of values from a spreadsheet. The caller must specify the spreadsheet ID and one or more ranges. Only values are cleared – all other properties of the cell (such as formatting, data validation, etc..) are kept.

Reference: request

Parameters

- **spreadsheet_id** – The ID of the spreadsheet to update.
- **ranges** – A list of ranges to clear in A1 notation.

values_batch_get(*spreadsheet_id*, *value_ranges*, *major_dimension='ROWS'*, *value_render_option=<ValueRenderOption.FORMATTED_VALUE: 'FORMATTED_VALUE'>*, *date_time_render_option=<DateTimeRenderOption.SERIAL_NUMBER: 'SERIAL_NUMBER'>*)

Returns multiple range of values from a spreadsheet. The caller must specify the spreadsheet ID and list of range.

Reference: request

Parameters

- **spreadsheet_id** – The ID of the spreadsheet to retrieve data from.
- **value_ranges** – The list of A1 notation of the values to retrieve.
- **major_dimension** – The major dimension that results should use. For example, if the spreadsheet data is: A1=1,B1=2,A2=3,B2=4, then requesting range=A1:B2,majorDimension=ROWS will return [[1,2],[3,4]], whereas requesting range=A1:B2,majorDimension=COLUMNS will return [[1,3],[2,4]].
- **value_render_option** – How values should be represented in the output. The default render option is ValueRenderOption.FORMATTED_VALUE.
- **date_time_render_option** – How dates, times, and durations should be represented in the output. This is ignored if valueRenderOption is FORMATTED_VALUE. The default dateTime render option is [DateTimeRenderOption.SERIAL_NUMBER].

Returns ValueRange

values_batch_update(*spreadsheet_id*, *body*, *parse=True*)

Impliments batch update

Parameters

- **spreadsheet_id** – id of spreadsheet
- **body** – body of request

- **parse** –

values_get(*spreadsheet_id*, *value_range*, *major_dimension='ROWS'*, *value_render_option=<ValueRenderOption.FORMATTED_VALUE: 'FORMATTED_VALUE'>*, *date_time_render_option=<DateTimeRenderOption.SERIAL_NUMBER: 'SERIAL_NUMBER'>*)

Returns a range of values from a spreadsheet. The caller must specify the spreadsheet ID and a range.

Reference: request

Parameters

- **spreadsheet_id** – The ID of the spreadsheet to retrieve data from.
- **value_range** – The A1 notation of the values to retrieve.
- **major_dimension** – The major dimension that results should use. For example, if the spreadsheet data is: A1=1,B1=2,A2=3,B2=4, then requesting range=A1:B2,majorDimension=ROWS will return [[1,2],[3,4]], whereas requesting range=A1:B2,majorDimension=COLUMNS will return [[1,3],[2,4]].
- **value_render_option** – How values should be represented in the output. The default render option is ValueRenderOption.FORMATTED_VALUE.
- **date_time_render_option** – How dates, times, and durations should be represented in the output. This is ignored if valueRenderOption is FORMATTED_VALUE. The default dateTime render option is [DateTimeRenderOption.SERIAL_NUMBER].

Returns

ValueRange

developer_metadata_delete(*spreadsheet_id*, *data_filter*)

Deletes all developer metadata matching the supplied filter

Reference: request

Parameters

- **spreadsheet_id** – The id of the spreadsheet to search.
- **data_filter** – A dictionary represeting a DeveloperMetadataLookup filter (see reference)

developer_metadata_create(*spreadsheet_id*, *key*, *value*, *location*)

Creates a new developer metadata entry at the specified location

Reference: request

Parameters

- **spreadsheet_id** – The id of the spreadsheet where metadata will be created.
- **key** – They key of the new developer metadata entry to create
- **value** – They value of the new developer metadata entry to create
- **location** – A dictionary represeting the location where metadata will be created

developer_metadata_update(*spreadsheet_id*, *key*, *value*, *location*, *data_filter*)

Updates all developer metadata matching the supplied filter

Reference: request

Parameters

- **spreadsheet_id** – The id of the spreadsheet to search.

- **location** – A dictionary represeting the location where metadata will be created
- **data_filter** – A dictionary represeting a DeveloperMetadataLookup filter (see reference)

Drive API Wrapper

class pygsheets.drive.**DriveAPIWrapper**(*http*, *data_path*, *retries=3*, *logger=<Logger pygsheets.drive (WARNING)>*)

A simple wrapper for the Google Drive API.

Various utility and convenience functions to support access to Google Drive files. By default the requests will access the users personal drive. Use enable_team_drive(team_drive_id) to connect to a TeamDrive instead.

Only functions used by pygsheet are wrapped. All other functionality can be accessed through the service attribute.

See reference for details.

Parameters

- **http** – HTTP object to make requests with.
- **data_path** – Path to the drive discovery file.

include_items_from_all_drive = None
Include files from TeamDrive and My Drive when executing requests.

enable_team_drive(*team_drive_id*)
Access TeamDrive instead of the users personal drive.

disable_team_drive()
Do not access TeamDrive (default behaviour).

get_update_time(*file_id*)
Returns the time this file was last modified in RFC 3339 format.

list(***kwargs*)
Fetch metadata of spreadsheets. Fetches a list of all files present in the users drive or TeamDrive. See Google Drive API Reference for details.

Reference: Files list request

Parameters **kwargs** – Standard parameters (see documentation for details).

Returns List of metadata.

create_folder(*name*, *folder=None*, ***kwargs*)
Create a new folder

Parameters

- **name** – The name to give the new folder
- **folder** – The id of the folder this one will be stored in
- **kwargs** – Standard parameters (see documentation for details).

Returns The new folder id

get_folder_id(*name*)
Fetch the first folder id with a given name

Parameters **name** – The name of the folder to find

folder_metadata (*query=''*, *only_team_drive=False*)
Fetch folder names, ids & and parent folder ids.

The query string can be used to filter the returned metadata.

Reference: search parameters docs.

Parameters **query** – Can be used to filter the returned metadata.

spreadsheet_metadata (*query=''*, *only_team_drive=False*, *fid=None*)
Fetch spreadsheet titles, ids & and parent folder ids.

The query string can be used to filter the returned metadata.

Reference: search parameters docs.

Parameters

- **fid** – id of file [optional]
- **query** – Can be used to filter the returned metadata.

delete (*file_id*, ***kwargs*)
Delete a file by ID.

Permanently deletes a file owned by the user without moving it to the trash. If the file belongs to a Team Drive the user must be an organizer on the parent. If the input id is a folder, all descendants owned by the user are also deleted.

Reference: delete request

Parameters

- **file_id** – The Id of the file to be deleted.
- **kwargs** – Standard parameters (see documentation for details).

move_file (*file_id*, *old_folder*, *new_folder*, *body=None*, ***kwargs*)
Move a file from one folder to another.

Requires the current folder to delete it.

Reference: update request

Parameters

- **file_id** – ID of the file which should be moved.
- **old_folder** – Current location.
- **new_folder** – Destination.
- **body** – Other fields of the file to change. See reference for details.
- **kwargs** – Optional arguments. See reference for details.

copy_file (*file_id*, *title*, *folder*, *body=None*, ***kwargs*)
Copy a file from one location to another

Reference: update request

Parameters

- **file_id** – Id of file to copy.
- **title** – New title of the file.
- **folder** – New folder where file should be copied.

- **body** – Other fields of the file to change. See reference for details.
- **kwargs** – Optional arguments. See reference for details.

update_file(*file_id*, *body=None*, ***kwargs*)
Update file body.

Reference: update request

Parameters

- **file_id** – ID of the file which should be updated.
- **body** – The properties of the file to update. See reference for details.
- **kwargs** – Optional arguments. See reference for details.

export(*sheet*, *file_format*, *path=''*, *filename=''*)
Download a spreadsheet and store it.

Exports a Google Doc to the requested MIME type and returns the exported content.

> **Warning:** This can at most export files with 10 MB in size!

Uses one or several export request to download the files. When exporting to CSV or TSV each worksheet is exported into a separate file. The API cannot put them into the same file. In this case the worksheet index is appended to the file-name.

Reference: request

Parameters

- **sheet** – The spreadsheet or worksheet to be exported.
- **file_format** – File format (`ExportType`)
- **path** – Path to where the file should be stored. (default: current working directory)
- **filename** – Name of the file. (default: Spreadsheet Id)

create_permission(*file_id*, *role*, *type*, ***kwargs*)
Creates a permission for a file or a TeamDrive.

See reference for more details.

Parameters

- **file_id** – The ID of the file or Team Drive.
- **role** – The role granted by this permission.
- **type** – The type of the grantee.
- **emailAddress** – The email address of the user or group to which this permission refers.
- **domain** – The domain to which this permission refers.
- **allowFileDiscovery** – Whether the permission allows the file to be discovered through search. This is only applicable for permissions of type domain or anyone.
- **expirationTime** – The time at which this permission will expire (RFC 3339 date-time). Expiration times have the following restrictions:
  - They can only be set on user and group permissions
  - The time must be in the future

  - The time cannot be more than a year in the future
- **emailMessage** – A plain text custom message to include in the notification email.
- **sendNotificationEmail** – Whether to send a notification email when sharing to users or groups. This defaults to true for users and groups, and is not allowed for other requests. It must not be disabled for ownership transfers.
- **supportsTeamDrives** – Whether the requesting application supports Team Drives. (Default: False)
- **transferOwnership** – Whether to transfer ownership to the specified user and downgrade the current owner to a writer. This parameter is required as an acknowledgement of the side effect. (Default: False)
- **useDomainAdminAccess** – Whether the request should be treated as if it was issued by a domain administrator; if set to true, then the requester will be granted access if they are an administrator of the domain to which the item belongs. (Default: False)

Returns Permission Resource

list_permissions(*file_id*, ***kwargs*)
List all permissions for the specified file.

See reference for more details.

Parameters

- **file_id** – The file to get the permissions for.
- **pageSize** – Number of permissions returned per request. (Default: all)
- **supportsTeamDrives** – Whether the application supports TeamDrives. (Default: False)
- **useDomainAdminAccess** – Request permissions as domain admin. (Default: False)

Returns List of Permission Resources

delete_permission(*file_id*, *permission_id*, ***kwargs*)
Deletes a permission.

See reference for more details.

Parameters

- **file_id** – The ID of the file or Team Drive.
- **permission_id** – The ID of the permission.
- **supportsTeamDrives** – Whether the requesting application supports Team Drives. (Default: false)
- **useDomainAdminAccess** – Whether the request should be treated as if it was issued by a domain administrator; if set to true, then the requester will be granted access if they are an administrator of the domain to which the item belongs. (Default: false)

6.2.5 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

6.3 Examples

Batching of api calls

```
wks.unlink()
for i in range(10):
    wks.update_value((1, i), i) # wont call api
wks.link() # will do all the updates
```

Protect an whole sheet

```
r = Datarange(worksheet=wks)
>>> r # this is a datarange unbounded on both indexes
<Datarange Sheet1>
>>> r.protected = True # this will make the whole sheet protected
```

Formatting columns column A as percentage format, column B as currency format. Then formatting row 1 as white , row 2 as grey colour. By @cwdjankoski

```
model_cell = pygsheets.Cell("A1")

model_cell.set_number_format(
    format_type = pygsheets.FormatType.PERCENT,
    pattern = "0%"
)
# first apply the percentage formatting
pygsheets.DataRange(
    left_corner_cell , right_corner_cell , worksheet = wks
 ).apply_format(model_cell)

# now apply the row-colouring interchangeably
gray_cell = pygsheets.Cell("A1")
gray_cell.color = (0.9529412, 0.9529412, 0.9529412, 0)

white_cell = pygsheets.Cell("A2")
white_cell.color = (1, 1, 1, 0)

cells = [gray_cell, white_cell]
```

(continues on next page)

(continued from previous page)

```
for r in range(start_row, end_row + 1):
    print(f"Doing row {r} ...", flush = True, end = "\r")
    wks.get_row(r, returnas = "range").apply_format(cells[ r % 2 ], fields =
↪"userEnteredFormat.backgroundColor")
```

Conditional Formatting This will apply conditional formatting to cells between A1 to A4. If the value is between 1 and 5, cell background will be red.

```
wks.add_conditional_formatting('A1', 'A4', 'NUMBER_BETWEEN', {'backgroundColor':{'red
↪':1}}, ['1','5'])
```

Note: If you have any interesting examples that you think will be helpful to others, please raise a PR or issue.

6.4 Some Tips

Note that in this article, wks means worksheet, ss means spreadsheet, gc means google client

easier way to acess sheet values:

```
for row in wks:
    print(row[0])
```

Access sheets by id:

```
wks1 = ss[0]
```

Conversion of sheet data

usually all the values are converted to string while using *get_** functions. But if you want then to retain their type, the change the *value_render* option to *ValueRenderOption.UNFORMATTED_VALUE*.

6.5 Versions

6.5.1 Version 2.0.0

This version is not backwards compatible with 1.x There is major rework in the library with this release. Some functions are renamed to have better consistency in naming and clear meaning.

- update_cell() renamed to update_value()
- update_cells() renamed to update_values()
- update_cells_prop() renamed to update_cells()
- changed authorize() params : outh_file -> client_secret, outh_creds_store ->credentials_directory, service_file -> service_account_file, credentials -> custom_credentials
- teamDriveId, enableTeamDriveSupport changed to client.drive.enable_team_drive, include_team_drive_items
- parameter changes for all get_* functions : include_empty, include_all changed to include_tailing_empty, include_tailing_empty_rows
- parameter changes in created_protected_range() : gridrange param changed to start, end
- remoed batch mode

- find() splited into find() and replace()
- removed (show/hide)_(row/column), use (show/hide)_dimensions instead
- removed link/unlink from spreadsheet

New Features added - chart Support added - Sort feature added - Better support for protected ranges - Multi header/index support in dataframes - Removed the dependency on oauth2client and uses google-auth and google-auth-oauth.

Other bug fixes and performance improvements

6.5.2 Version 1.1.4

Changelog not available

CHAPTER 7

Indices and tables

- genindex
- modindex
- search

Python Module Index

p

Index

D

E

F

G

H

I

L

M

N

O

P

R

S

T

U

V

W

X