pygsheets Documentation

Release 1.1

nithinmurali

May 05, 2018

Contents

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

CHAPTER 1

Features

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

CHAPTER 2

Small Example

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

```
import pygsheets

gc = pygsheets.authorize()

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

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

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

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

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

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

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

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

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

(continues on next page)

(continued from previous page)

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

CHAPTER 3

Installation

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

CHAPTER 4

Overview

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

CHAPTER 5

Authors and License

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

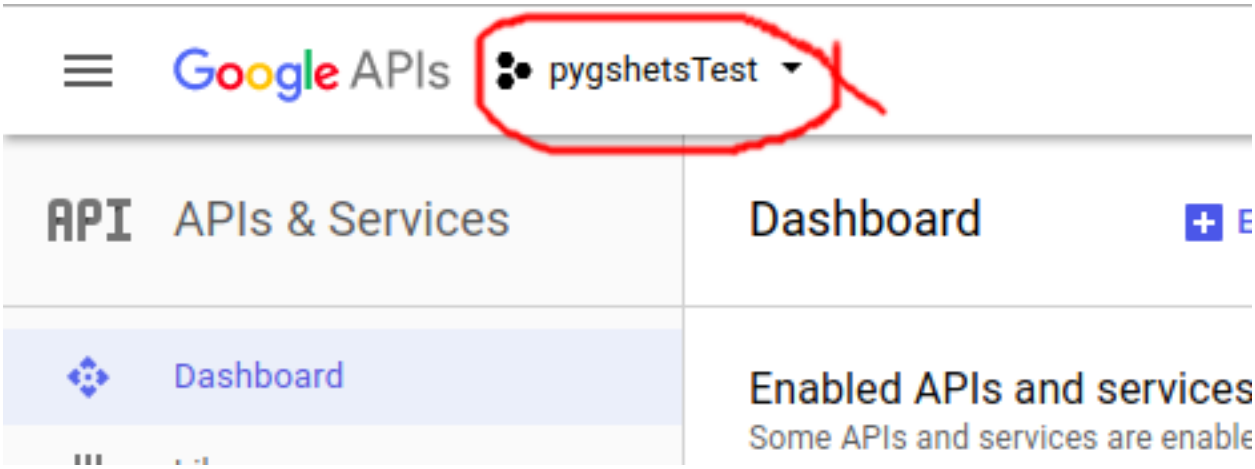
Feel free to improve this package and send a pull request to GitHub.

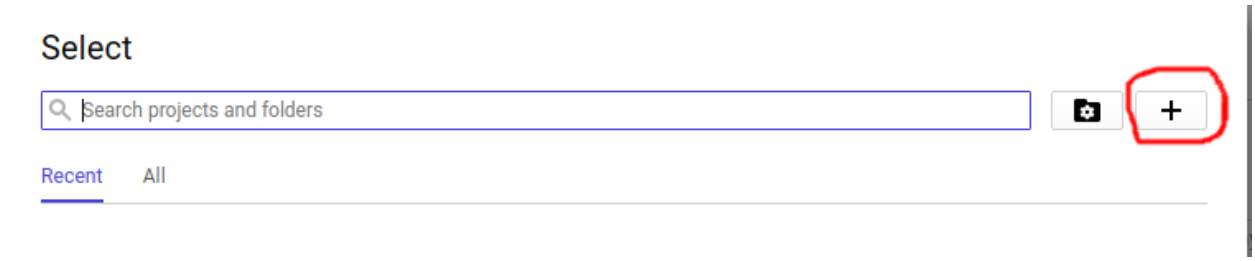
Contents:

5.1 Authorizing pygsheets

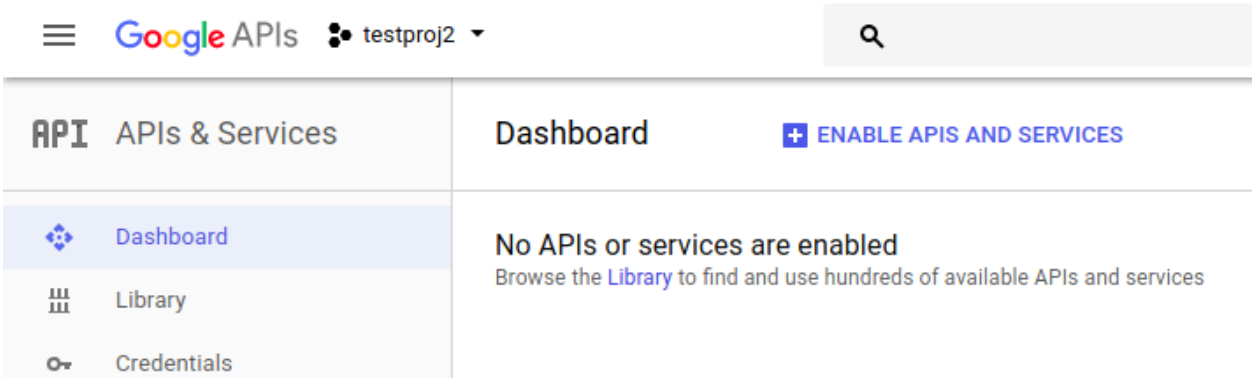
There are multiple ways to authorize google sheets. First you should create a developer account (follow below steps) and create the type of credentials depending on your need. But remember not to give away any of there credentials, as your usage quota is limited.

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

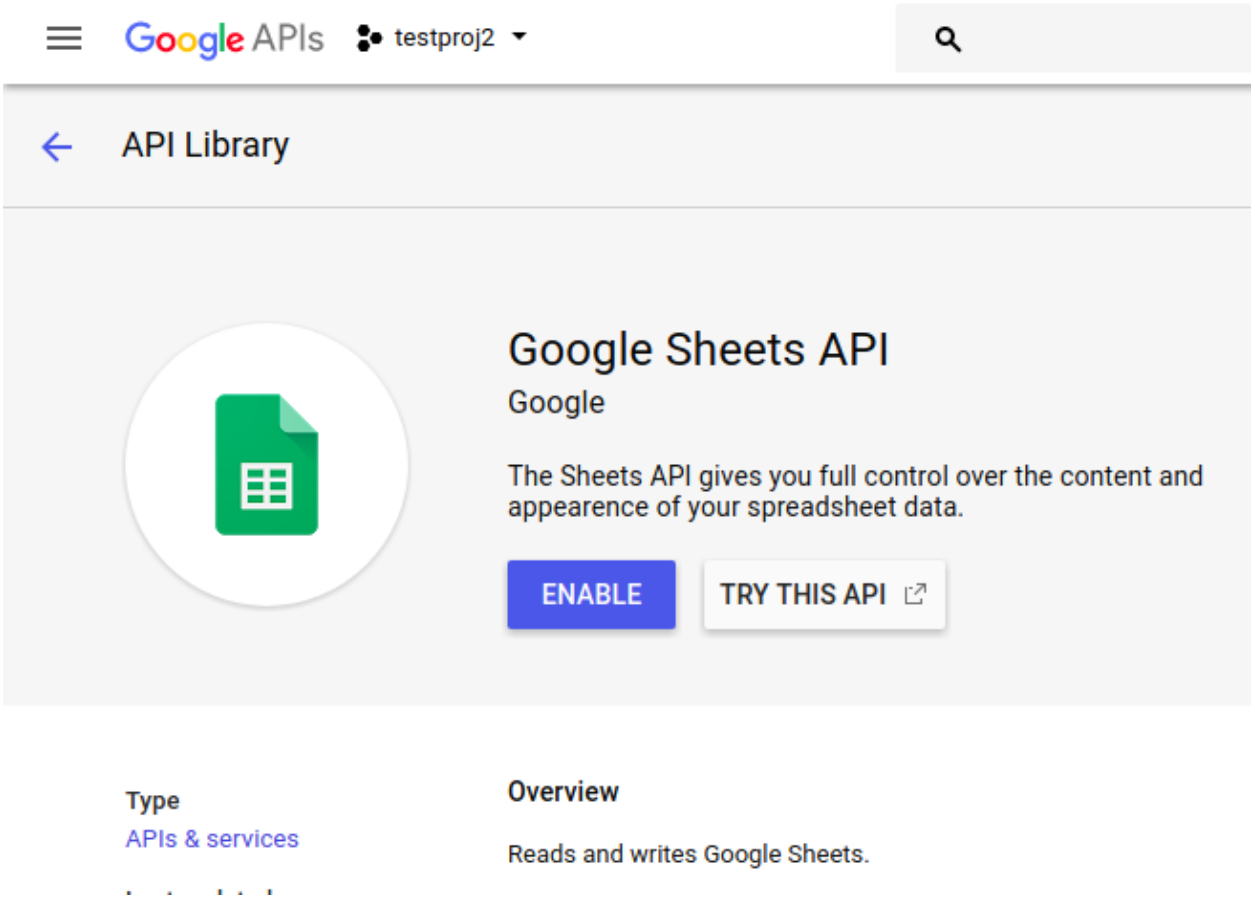




2. You will be redirected to the Project Dashboard, there click on “Enable Apis and services”, search for “Sheets API”.



3. In the API screen click on ‘ENABLE’ to enable this API



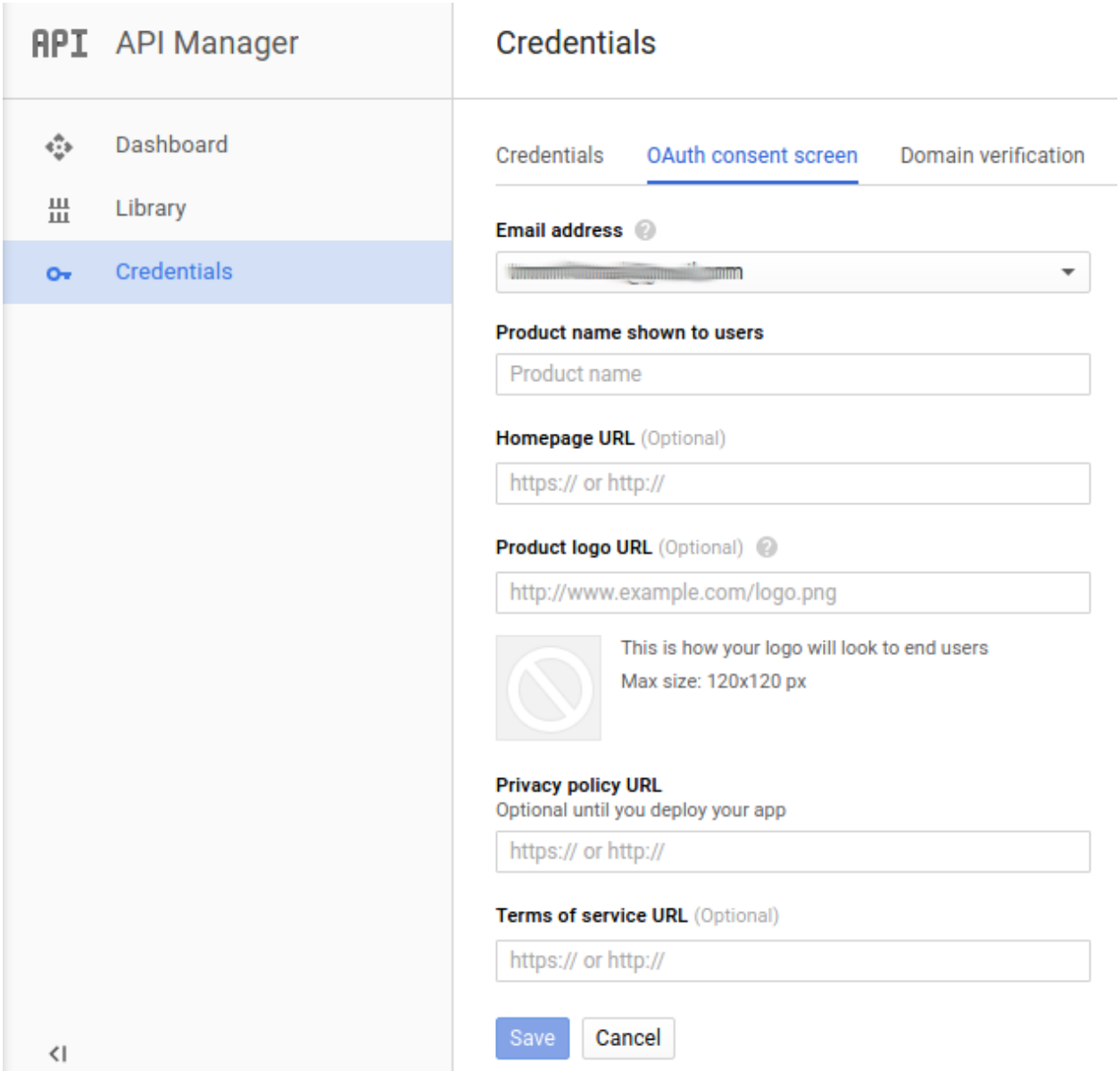
4. Similarly enable the "Drive API". We require drives api for getting list of spreadsheets, deleting them etc.

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

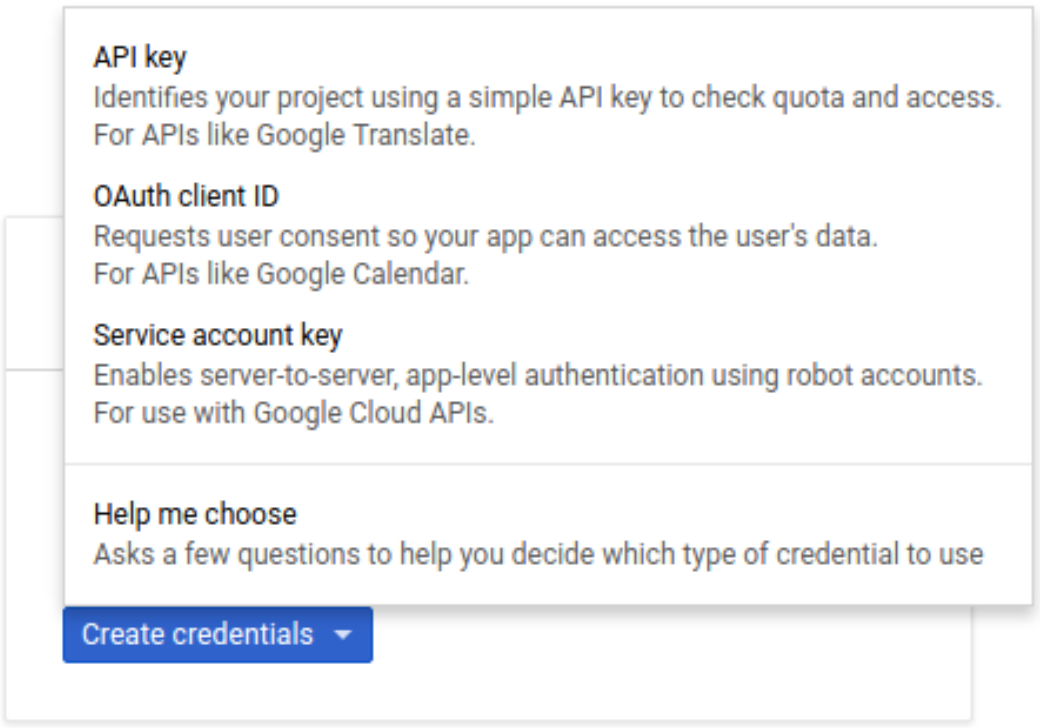
5.1.1 OAuth Credentials

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

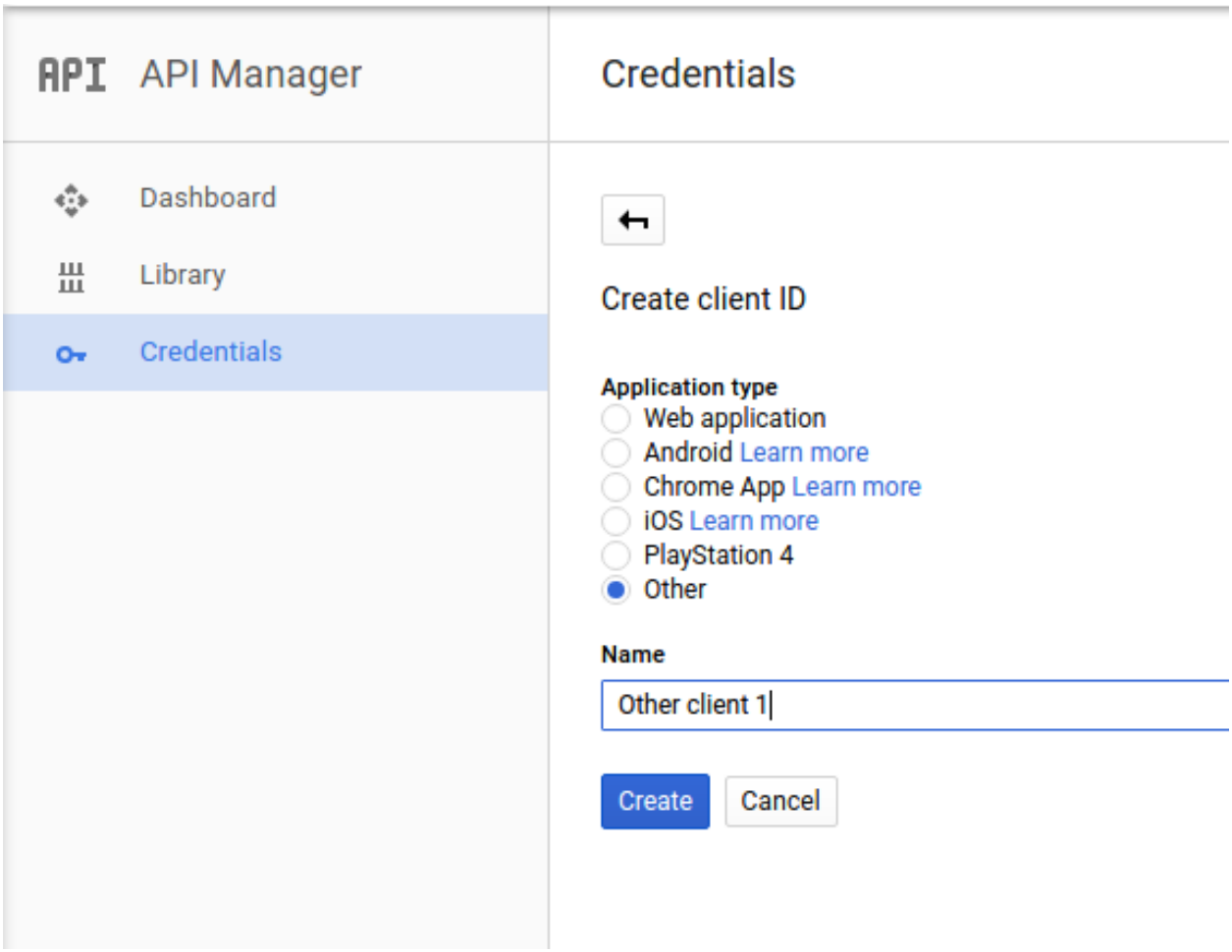
5. First you need to configure how the conscent sceen will look while asking for autorization. Go to "Credentials" Side Tab and choose "OAuth Conscent screen". There inset all the data you need to show while asking for authorization and save it



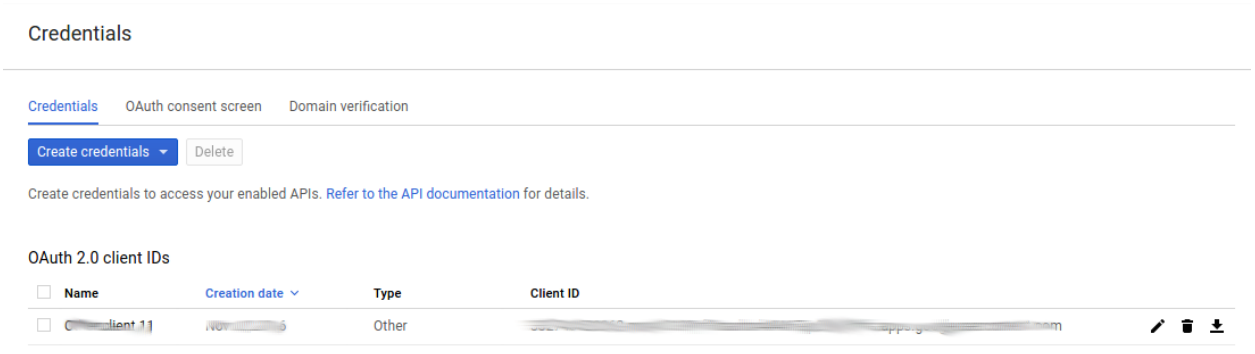
6. Go to "Credentials" Tab and choose "Create Credentials > OAuth Client ID".



7. Now choose the Application Type as ‘Other’



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



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

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

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

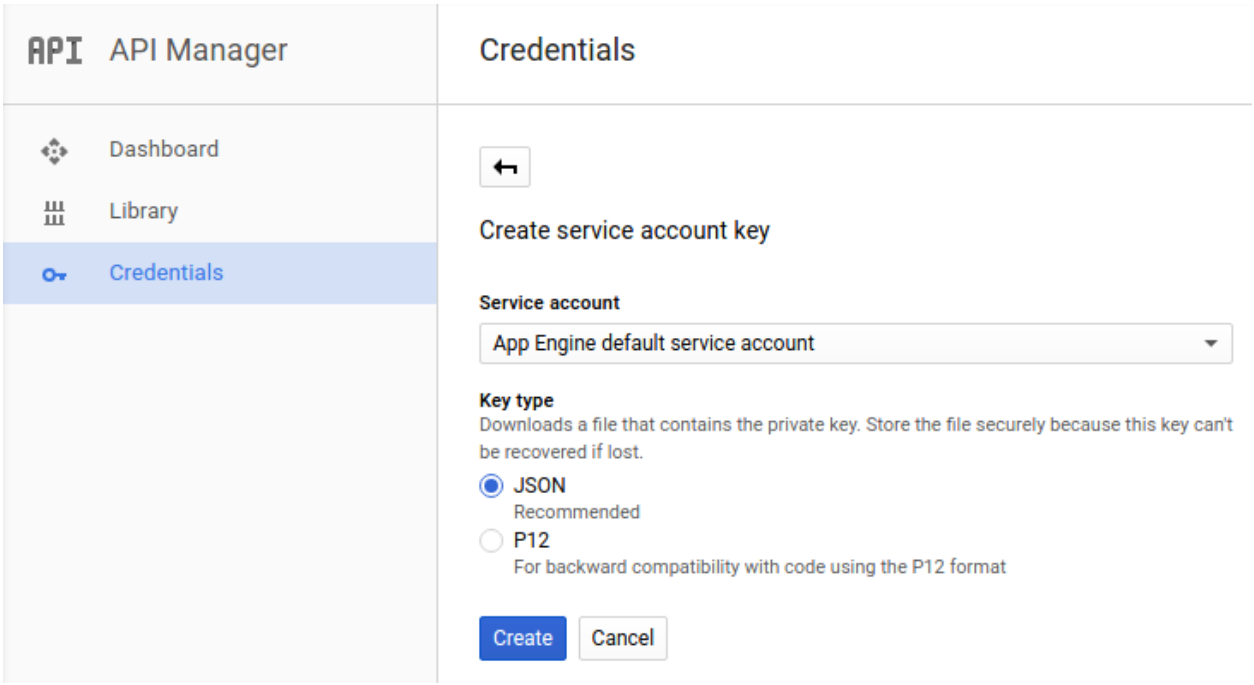
Incase you are running the script in a headless server where it can't open a browser, you can enbale *non-local* authorization. Hence instead of opening a browser in the same meachine, it will provide a link which you can run on your local meachine and authorize the application.

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

5.1.2 Signed Credentials

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

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



You will automatically download a JSON file with this data.

This is how this file may look like:

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

(continues on next page)

(continued from previous page)

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

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

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

5.1.3 Custom Credentials Objects

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

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

    def refresh (self, http):
        # get new access_token
        # this only gets called if access_token is None
```

Then you could pass this for authorization as

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

5.2 pygsheets Reference

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

5.2.1 Top Level Interface

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

Login to Google API using OAuth2 credentials.

This function instantiates *Client* and performs authentication.

Parameters

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

Returns *Client* instance.

class pygsheets.**Client**(*oauth*, *http_client=None*, *retries=1*, *no_cache=False*)
An instance of this class communicates with Google API.

Parameters

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

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

create(*title*, *parent_id=None*)
Creates a spreadsheet, returning a *Spreadsheet* instance.

Parameters

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

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

Parameters

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

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

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

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

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

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

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

Parameters

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

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

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

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

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

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

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

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

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

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

:param filter_query : (optional) either title or query, see https://developers.google.com/drive/v3/web/search-parameters

Returns list of *Spreadsheet* instances

list_ssheets (*parent_id=None*)
Lists all spreadsheets

Parameters **parent_id** – (optional) If specified filters spreadsheets by parent_id.

Returns list of dictionaries with id and name for each spreadsheet

5.2.2 Models

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

Spreadsheet

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

worksheet_cls
alias of pygsheets.worksheet.Worksheet

id
Id of the spreadsheet.

title
Title 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 (*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.

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.

Request: https://developers.google.com/sheets/api/reference/rest/v4/spreadsheets/request#findreplacerequest

Parameters

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

Key searchByRegex Consider pattern a regex pattern. (default False)

Key matchCase Match case sensitive. (default False)

Key matchEntireCell Only match on full match. (default False)

Key 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.

Key searchByRegex Consider pattern a regex pattern. (default False)

Key matchCase Match case sensitive. (default False)

Key matchEntireCell Only match on full match. (default False)

Key includeFormulas Match fields with formulas too. (default False)

:returns A list of lists of *Cells*

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

Parameters

- **addr** – email for user/group, domain address for domains or 'anyone'

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

list_permissions()
List all permissions for this spreadsheet.

Returns Permissions (list)

remove_permissions(*addr*)
Remove user from permissions list.

Parameters **addr** – User email.

batch_start()
Start batch mode.

This will begin batch mode. All requests made to the sheet will instead be collected and executed once done. This should speed up processing of local file and reduce the number of API calls.

bath_stop(*discard=False*)
Stop batch mode.

This will end batch mode and all changes made during batch mode will be either synched with the remote spreadsheet or discarded.

Parameters **discard** – Discard all changes made during batch mode.

link(*syncToCloud=False*)
Link spreadsheet with remote.

Linked spreadsheets will upload each change to the remote. This ensures that the local copy will always be up to date. This will link all sheets and cause an update. Either local or remote data will be overwritten.

Parameters **syncToCloud** – True -> Overwrite remote with local changes. False -> Overwrite local with remote changes.

unlink()
Unlink spreadsheet from remote.

Unlinked spreadsheets will no longer update the remote. All changes will only apply to the local copy. Use link() to re-link this spreadsheet with remote.

export(*fformat='text/csv:.csv'*, *filename=None*)
Export all the worksheets to the file.

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

- **fformat** – ExportType.<?>
- **filename** – File name with path. Otherwise file will be stored in working directory.

custom_request(*request*, *fields*)
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: 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.

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

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.

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.

refresh(*update_grid=False*)

refresh worksheet data

link(*syncToCloud=True*)

Link the spread sheet with cloud, so all local changes will be updated instantly, so does all data fetches

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

unlink()

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

sync()

sync the worksheet to cloud

cell(*addr*)

Returns cell object at given address.

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

Returns an instance of a *Cell*

Example:

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

range(*crange*, *returnas='cells'*)

Returns a list of *Cell* objects from specified range.

Parameters

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

get_value(*addr*)

value of a cell at given address

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

get_values(*start*, *end*, *returnas='matrix'*, *majdim='ROWS'*, *include_tailing_empty=True*, *include_empty_rows=False*, *value_render='FORMATTED_VALUE'*)

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
- **majdim** – The major dimension of the matrix. ('ROWS') ('COLMUNS' not implimented)
- **returnas** – The type to return the fetched values as. ('matrix', 'cell', 'range')
- **include_tailing_empty** – Wheather to include empty trailing cells/values after last non-zero value
- **include_empty_rows** – Wheather 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
- **value_render** – format of output values

Returns 'range' *DataRange* 'cell': [*Cell*] 'matrix': [[...], [...], ...]

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

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

Parameters

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

Example:

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

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

Returns a list of dictionaries, all of them having:

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

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

Parameters

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

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

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

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

Parameters

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

get_col (*col*, *returnas='matrix'*, *include_tailing_empty=True*)
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 values
- **col** – index of col
- **returnas** – ('matrix' or 'cell') return as cell objects or just values

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

Parameters

- **start** – start adress
- **end** – end adress

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 cell values in batch, it can take either a cell list or a range and values. cell list is only efficient for large lists. This will only update the cell values not other properties.

Parameters

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

update_cells(*cell_list*, *fields='*'*)

update cell properties and data from a list of cell obejcts

Parameters

- **cell_list** – list of cell objects
- **fields** – cell fields to update, in google FieldMask format(see api docs)

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 colums 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: https://developers.google.com/sheets/api/reference/rest/v4/spreadsheets/request#insertdimensionrequest

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: https://developers.google.com/sheets/api/reference/rest/v4/spreadsheets/request#insertdimensionrequest

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: https://developers.google.com/sheets/api/reference/rest/v4/spreadsheets#CellData
Reference FieldMask: https://developers.google.com/protocol-buffers/docs/reference/google.protobuf#google.protobuf.FieldMask

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=100*)
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.

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 first 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 first row or column.
- **dimension** – 'ROWS' or 'COLUMNS'

hide_rows (*start*, *end=None*)
Hide one or more rows.

Parameters

- **start** – Index of the first row.
- **end** – Index of the last row.

show_rows(*start*, *end=None*)
Show one or more rows.

Parameters

- **start** – Index of the first row.
- **end** – Index of the last row.

hide_columns(*start*, *end=None*)
Hide one or more columns.

Parameters

- **start** – Index of the first column.
- **end** – Index of the last column.

show_columns(*start*, *end=None*)
Show one or more columns.

Parameters

- **start** – Index of the first column.
- **end** – Index of the last column.

adjust_row_height(*start*, *end=None*, *pixel_size=100*)
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.

append_table(*start='A1'*, *end=None*, *values=None*, *dimension='ROWS'*, *overwrite=False*)
Append values to the sheet.

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

Parameters

- **start** – Top left cell of the range (requires a label).
- **end** – Bottom right cell of the range (requires a label).
- **values** – List of values for the new row or column.
- **dimension** – Dimension to which the values will be added ('ROWS' or 'COLUMNS')
- **overwrite** – Overwrite existing values.

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.

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.

Request: https://developers.google.com/sheets/api/reference/rest/v4/spreadsheets/request#findreplacerequest

Parameters

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

Key searchByRegex Consider pattern a regex pattern. (default False)

Key matchCase Match case sensitive. (default False)

Key matchEntireCell Only match on full match. (default False)

Key includeFormulas Match fields with formulas too. (default False)

find(*pattern*, *searchByRegex=False*, *matchCase=False*, *matchEntireCell=False*, *includeFormulas=False*)

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.

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)

:returns A list of *Cells*.

create_named_range(*name*, *start*, *end*)

Create a new named range in this worksheet.

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

Parameters

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

:return *DataRange*

get_named_range(*name*)

Get a named range by name.

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

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

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

Parameters

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

create_protected_range(*gridrange*)
Create protected range.

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

Parameters **gridrange** – Grid range of the cells to be protected.

remove_protected_range(*range_id*)
Remove protected range.

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

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

set_dataframe(*df*, *start*, *copy_index=False*, *copy_head=True*, *fit=False*, *escape_formulae=False*, *nan='NaN'*)
Load sheet from Pandas data frame.

Will load all data contained within the Pandas data frame into this worksheet. It will begin filling the worksheet at cell start.

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.
- **fit** – Resize the worksheet to fit all data inside if necessary.
- **escape_formulae** – Any value starting with an equal sign (=), will be prefixed with an apostroph (‘) to avoid value being interpreted as a formula.
- **nan** – Value with which NaN values are replaced.

get_as_df(*has_header=True*, *index_colum=None*, *start=None*, *end=None*, *numerize=True*, *empty_value=''*)
Get the content of this worksheet as a pandas data frame.

Parameters

- **has_header** – Interpret first row as data frame header.
- **index_colum** – Column to use as data frame index (integer).
- **numerize** – Numerize cell values.

- **empty_value** – Placeholder value to represent empty cells.
- **start** – Top left cell to load into data frame. (default: A1)
- **end** – Bottom right cell to load into data frame. (default: (rows, cols))

Returns pandas.Dataframe

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

A worksheet can be exported as CSV, MS_Excel, Open_Office_sheet or PDF.

Parameters

- **fformat** – Format of the exported file.
- **filename** – File name of the exported file (incl. appropriate file ending).

copy_to(*spreadsheet_id*)
Copy this worksheet to the specified spreadsheet.

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

Parameters **spreadsheet_id** – Id of the spreadsheet this worksheet should be copied to.

DataRange

class pygsheets.**DataRange**(*start=None*, *end=None*, *worksheet=None*, *name=''*, *data=None*, *name_id=None*, *namedjson=None*, *protect_id=None*, *protectedjson=None*)
DataRange specifies a range of cells in the sheet

Parameters

- **start** – top left cell address
- **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

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

protected
get/set range protection

start_addr
top-left address of the range

end_addr
bottom-right address of the range

range
Range in format A1:C5

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

Parameters **only_data** – fetch only data

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

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

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

Parameters **values** – values as matrix

update_named_range()
update the named properties

Cell

class pygsheets.**Cell**(*pos*, *val=''*, *worksheet=None*, *cell_data=None*)
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: https://developers.google.com/sheets/api/reference/rest/v4/spreadsheets#borders.

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

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

row
Row number of the cell.

col
Column number of the cell.

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

value
This cells formatted value.

value_unformatted
Unformatted value of this cell.

formula
Get/Set this cells formula if any.

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.

The following formats can be set:

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 strikethrough format. (boolean) underline: Set/remove underline format. (boolean)

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

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

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: 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*

horizontal_alignment
Horizontal alignment of the value in this cell.

vertical_alignment
Vertical alignment of the value in this cell.

wrap_strategy
How to wrap text in this cell.

Possible wrap strategies: 'OVERFLOW_CELL', 'LEGACY_WRAP', 'CLIP', 'WRAP'.

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

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?

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.

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.

5.2.3 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

5.3 Indices and tables

- genindex
- modindex
- search

CHAPTER 6

Indices and tables

- genindex
- modindex
- search

Python Module Index

p

Index

L

N

O

P

R

S

T

U

V

W