gspread Documentation

Release 4.0.0

Anton Burnashev

Sep 27, 2021

Contents

gspread is a Python API for Google Sheets.

Features:

- Google Sheets API v4.
- Open a spreadsheet by title, key or url.
- Read, write, and format cell ranges.
- Sharing and access control.
- Batching updates.

CHAPTER 1

Installation

```
pip install gspread
```

Requirements: Python 3+.

CHAPTER 2

Quick Example

```
import gspread

gc = gspread.service_account()

# Open a sheet from a spreadsheet in one go
wks = gc.open("Where is the money Lebowski?").sheet1

# Update a range of cells using the top left corner address
wks.update('A1', [[1, 2], [3, 4]])

# Or update a single cell
wks.update('B42', "it's down there somewhere, let me take another look.")

# Format the header
wks.format('A1:B1', {'textFormat': {'bold': True}})
```

CHAPTER 3

Getting Started

3.1 Authentication

To access spreadsheets via Google Sheets API you need to authenticate and authorize your application.

- If you plan to access spreadsheets on behalf of a bot account use *Service Account*.
- If you'd like to access spreadsheets on behalf of end users (including yourself) use *OAuth Client ID*.

3.1.1 Enable API Access for a Project

1. Head to Google Developers Console and create a new project (or select the one you already have).
2. In the box labeled "Search for APIs and Services", search for "Google Drive API" and enable it.
3. In the box labeled "Search for APIs and Services", search for "Google Sheets API" and enable it.

3.1.2 For Bots: Using Service Account

A service account is a special type of Google account intended to represent a non-human user that needs to authenticate and be authorized to access data in Google APIs [sic].

Since it's a separate account, by default it does not have access to any spreadsheet until you share it with this account. Just like any other Google account.

Here's how to get one:

1. *Enable API Access for a Project* if you haven't done it yet.
2. Go to "APIs & Services > Credentials" and choose "Create credentials > Service account key".
3. Fill out the form
4. Click "Create" and "Done".
5. Press "Manage service accounts" above Service Accounts.

6. Press on near recently created service account and select "Manage keys" and then click on "ADD KEY > Create new key".
7. Select JSON key type and press "Create".

You will automatically download a JSON file with credentials. It may look like this:

```
{
    "type": "service_account",
    "project_id": "api-project-XXX",
    "private_key_id": "2cd ... ba4",
    "private_key": "-----BEGIN PRIVATE KEY-----\nNrDyLw ... jINQh/9\n-----END PRIVATE␣
↪KEY-----\n",
    "client_email": "473000000000-yoursisdifferent@developer.gserviceaccount.com",
    "client_id": "473 ... hd.apps.googleusercontent.com",
    ...
}
```

Remember the path to the downloaded credentials file. Also, in the next step you'll need the value of *client_email* from this file.

6. Very important! Go to your spreadsheet and share it with a *client_email* from the step above. Just like you do with any other Google account. If you don't do this, you'll get a `gspread.exceptions.SpreadsheetNotFound` exception when trying to access this spreadsheet from your application or a script.
7. Move the downloaded file to `~/.config/gspread/service_account.json`. Windows users should put this file to `%APPDATA%\gspread\service_account.json`.
8. Create a new Python file with this code:

```
import gspread

gc = gspread.service_account()

sh = gc.open("Example spreadsheet")

print(sh.sheet1.get('A1'))
```

Ta-da!

Note: If you want to store the credentials file somewhere else, specify the path to *service_account.json* in *`service_account()`*:

```
gc = gspread.service_account(filename='path/to/the/downloaded/file.json')
```

Make sure you store the credentials file in a safe place.

For the curious, under the hood *`service_account()`* loads your credentials and authorizes gspread. Similarly to the code that has been used for authentication prior to the gspread version 3.6:

```
from google.oauth2.service_account import Credentials

scopes = [
    'https://www.googleapis.com/auth/spreadsheets',
    'https://www.googleapis.com/auth/drive'
]

credentials = Credentials.from_service_account_file(
```

(continues on next page)

(continued from previous page)

```
    'path/to/the/downloaded/file.json',
    scopes=scopes
)

gc = gspread.authorize(credentials)
```

There is also the option to pass credentials as a dictionary:

```
import gspread

credentials = {
    "type": "service_account",
    "project_id": "api-project-XXX",
    "private_key_id": "2cd ... ba4",
    "private_key": "-----BEGIN PRIVATE KEY-----\nNrDyLw ... jINQh/9\n-----END PRIVATE␣
↪KEY-----\n",
    "client_email": "473000000000-yoursisdifferent@developer.gserviceaccount.com",
    "client_id": "473 ... hd.apps.googleusercontent.com",
    ...
}

gc = gspread.service_account_from_dict(credentials)

sh = gc.open("Example spreadsheet")

print(sh.sheet1.get('A1'))
```

Note: Older versions of gspread have used oauth2client. Google has deprecated it in favor of *google-auth*. If you're still using *oauth2client* credentials, the library will convert these to *google-auth* for you, but you can change your code to use the new credentials to make sure nothing breaks in the future.

3.1.3 For End Users: Using OAuth Client ID

This is the case where your application or a script is accessing spreadsheets on behalf of an end user. When you use this scenario, your application or a script will ask the end user (or yourself if you're running it) to grant access to the user's data.

1. *Enable API Access for a Project* if you haven't done it yet.
2. Go to "APIs & Services > OAuth Consent Screen." Click the button for "Configure Consent Screen".

a. In the "1 OAuth consent screen" tab, give your app a name and fill the "User support email" and "Developer contact information". Click "SAVE AND CONTINUE".

b. There is no need to fill in anything in the tab "2 Scopes", just click "SAVE AND CONTINUE".

c. In the tab "3 Test users", add the Google account email of the end user, typically your own Google email. Click "SAVE AND CONTINUE".

d. Double check the "4 Summary" presented and click "BACK TO DASHBOARD".

3. Go to "APIs & Services > Credentials"
4. Click "+ Create credentials" at the top, then select "OAuth client ID".
5. Select "Desktop app", name the credentials and click "Create". Click "Ok" in the "OAuth client created" popup.

6. Download the credentials by clicking the Download JSON button in "OAuth 2.0 Client IDs" section.
7. Move the downloaded file to `~/.config/gspread/credentials.json`. Windows users should put this file to `%APPDATA%\gspread\credentials.json`.

Create a new Python file with this code:

```
import gspread

gc = gspread.oauth()

sh = gc.open("Example spreadsheet")

print(sh.sheet1.get('A1'))
```

When you run this code, it launches a browser asking you for authentication. Follow the instruction on the web page. Once finished, gspread stores authorized credentials in the config directory next to *credentials.json*. You only need to do authorization in the browser once, following runs will reuse stored credentials.

Note: If you want to store the credentials file somewhere else, specify the path to *credentials.json* and *authorized_user.json* in *oauth()*:

```
gc = gspread.oauth(
    credentials_filename='path/to/the/credentials.json',
    authorized_user_filename='path/to/the/authorized_user.json'
)
```

Make sure you store the credentials file in a safe place.

Attention: Security Credentials file and authorized credentials contain sensitive data. **Do not share these files with others** and treat them like private keys.

If you are concerned about giving the application access to your spreadsheets and Drive, use Service Accounts.

Note: The user interface of Google Developers Console may be different when you're reading this. If you find that this document is out of sync with the actual UI, please update it. Improvements to the documentation are always welcome. Click **Edit on GitHub** in the top right corner of the page, make it better and submit a PR.

CHAPTER 4

Usage

4.1 Examples of gspread Usage

If you haven't yet authorized your app, read *Authentication* first.

4.1.1 Opening a Spreadsheet

You can open a spreadsheet by its title as it appears in Google Docs:

```
sh = gc.open('My poor gym results')
```

If you want to be specific, use a key (which can be extracted from the spreadsheet's url):

```
sht1 = gc.open_by_key('0BmgG6nO_6dprdS1MN3d3MkdPa142WFRrdnRRUWl1UFE')
```

Or, if you feel really lazy to extract that key, paste the entire spreadsheet's url

```
sht2 = gc.open_by_url('https://docs.google.com/spreadsheet/ccc?key=0Bm...FE&hl')
```

4.1.2 Creating a Spreadsheet

Use *create()* to create a new blank spreadsheet:

```
sh = gc.create('A new spreadsheet')
```

Note: If you're using a *service account*, this new spreadsheet will be visible only to this account. To be able to access newly created spreadsheet from Google Sheets with your own Google account you *must* share it with your email. See how to share a spreadsheet in the section below.

4.1.3 Sharing a Spreadsheet

If your email is *otto@example.com* you can share the newly created spreadsheet with yourself:

```
sh.share('otto@example.com', perm_type='user', role='writer')
```

See `share()` documentation for a full list of accepted parameters.

4.1.4 Selecting a Worksheet

Select worksheet by index. Worksheet indexes start from zero:

```
worksheet = sh.get_worksheet(0)
```

Or by title:

```
worksheet = sh.worksheet("January")
```

Or the most common case: *Sheet1*:

```
worksheet = sh.sheet1
```

To get a list of all worksheets:

```
worksheet_list = sh.worksheets()
```

4.1.5 Creating a Worksheet

```
worksheet = sh.add_worksheet(title="A worksheet", rows="100", cols="20")
```

4.1.6 Deleting a Worksheet

```
sh.del_worksheet(worksheet)
```

4.1.7 Getting a Cell Value

Using A1 notation:

```
val = worksheet.acell('B1').value
```

Or row and column coordinates:

```
val = worksheet.cell(1, 2).value
```

If you want to get a cell formula:

```
cell = worksheet.acell('B1', value_render_option='FORMULA').value

# or

cell = worksheet.cell(1, 2, value_render_option='FORMULA').value
```

4.1.8 Getting All Values From a Row or a Column

Get all values from the first row:

```
values_list = worksheet.row_values(1)
```

Get all values from the first column:

```
values_list = worksheet.col_values(1)
```

Note: So far we've been fetching a limited amount of data from a sheet. This works great until you need to get values from hundreds of cells or iterating over many rows or columns.

Under the hood, gspread uses Google Sheets API v4. Most of the time when you call a gspread method to fetch or update a sheet gspread produces one HTTP API call.

HTTP calls have performance costs. So if you find your app fetching values one by one in a loop or iterating over rows or columns you can improve the performance of the app by fetching data in one go.

What's more, Sheets API v4 introduced Usage Limits (as of this writing, 500 requests per 100 seconds per project, and 100 requests per 100 seconds per user). When your application hits that limit, you get an *`APIError`* *429 RESOURCE_EXHAUSTED*.

Here are the methods that may help you to reduce API calls:

- `get_all_values()` fetches values from all of the cells of the sheet.
- `get()` fetches all values from a range of cells.
- `batch_get()` can fetch values from multiple ranges of cells with one API call.
- `update()` lets you update a range of cells with a list of lists.
- `batch_update()` lets you update multiple ranges of cells with one API call.

4.1.9 Getting All Values From a Worksheet as a List of Lists

```
list_of_lists = worksheet.get_all_values()
```

4.1.10 Getting All Values From a Worksheet as a List of Dictionaries

```
list_of_dicts = worksheet.get_all_records()
```

4.1.11 Finding a Cell

Find a cell matching a string:

```
cell = worksheet.find("Dough")

print("Found something at R%sC%s" % (cell.row, cell.col))
```

Find a cell matching a regular expression

```
amount_re = re.compile(r'(Big|Enormous) dough')
cell = worksheet.find(amount_re)
```

find returns *None* if value is not Found

4.1.12 Finding All Matched Cells

Find all cells matching a string:

```
cell_list = worksheet.findall("Rug store")
```

Find all cells matching a regexp:

```
criteria_re = re.compile(r'(Small|Room-tiering) rug')
cell_list = worksheet.findall(criteria_re)
```

4.1.13 Clear A Worksheet

Clear one or multiple celle ranges at once:

```
worksheet.batch_clear(["A1:B1", "C2:E2", "my_named_range"])
```

Clear the entire workshseet:

```
worksheet.clear()
```

4.1.14 Cell Object

Each cell has a value and coordinates properties:

```
value = cell.value
row_number = cell.row
column_number = cell.col
```

4.1.15 Updating Cells

Using A1 notation:

```
worksheet.update('B1', 'Bingo!')
```

Or row and column coordinates:

```
worksheet.update_cell(1, 2, 'Bingo!')
```

Update a range

```
worksheet.update('A1:B2', [[1, 2], [3, 4]])
```

4.1.16 Formatting

Here's an example of basic formatting.

Set **A1:B1** text format to bold:

```
worksheet.format('A1:B1', {'textFormat': {'bold': True}})
```

Color the background of **A2:B2** cell range in black, change horizontal alignment, text color and font size:

```
worksheet.format("A2:B2", {
    "backgroundColor": {
      "red": 0.0,
      "green": 0.0,
      "blue": 0.0
    },
    "horizontalAlignment": "CENTER",
    "textFormat": {
      "foregroundColor": {
        "red": 1.0,
        "green": 1.0,
        "blue": 1.0
      },
      "fontSize": 12,
      "bold": True
    }
})
```

The second argument to `format()` is a dictionary containing the fields to update. A full specification of format options is available at CellFormat in Sheet API Reference.

Tip: gspread-formatting offers extensive functionality to help you when you go beyond basics.

4.1.17 Using gspread with pandas

pandas is a popular library for data analysis. The simplest way to get data from a sheet to a pandas DataFrame is with `get_all_records()`:

```
import pandas as pd

dataframe = pd.DataFrame(worksheet.get_all_records())
```

Here's a basic example for writing a dataframe to a sheet. With `update()` we put the header of a dataframe into the first row of a sheet followed by the values of a dataframe:

```
import pandas as pd

worksheet.update([dataframe.columns.values.tolist()] + dataframe.values.tolist())
```

For advanced pandas use cases check out these libraries:

- gspread-pandas
- gspread-dataframe

4.1.18 Using gspread with NumPy

NumPy is a library for scientific computing in Python. It provides tools for working with high performance multi-dimensional arrays.

Read contents of a sheet into a NumPy array:

```
import numpy as np
array = np.array(worksheet.get_all_values())
```

The code above assumes that your data starts from the first row of the sheet. If you have a header row in the first row, you need replace `worksheet.get_all_values()` with `worksheet.get_all_values()[1:]`.

Write a NumPy array to a sheet:

```
import numpy as np

array = np.array([[1, 2, 3], [4, 5, 6]])

# Write the array to worksheet starting from the A2 cell
worksheet.update('A2', array.tolist())
```

CHAPTER 5

Advanced

5.1 Advanced Usage

5.1.1 Custom Authentication

Google Colaboratory

If you familiar with the Jupyter Notebook, Google Colaboratory is probably the easiest way to get started using gspread:

```
from google.colab import auth
auth.authenticate_user()

import gspread
from oauth2client.client import GoogleCredentials

gc = gspread.authorize(GoogleCredentials.get_application_default())
```

See the full example in the External data: Local Files, Drive, Sheets, and Cloud Storage notebook.

Using Authlib

Using `Authlib` instead of `google-auth`. Similar to google.auth.transport.requests.AuthorizedSession Authlib's `AssertionSession` can automatically refresh tokens.:

```
import json
from gspread import Client
from authlib.integrations.requests_client import AssertionSession

def create_assertion_session(conf_file, scopes, subject=None):
    with open(conf_file, 'r') as f:
        conf = json.load(f)
```

(continues on next page)

(continued from previous page)

```
    token_url = conf['token_uri']
    issuer = conf['client_email']
    key = conf['private_key']
    key_id = conf.get('private_key_id')

    header = {'alg': 'RS256'}
    if key_id:
        header['kid'] = key_id

    # Google puts scope in payload
    claims = {'scope': ' '.join(scopes)}
    return AssertionSession(
        grant_type=AssertionSession.JWT_BEARER_GRANT_TYPE,
        token_url=token_url,
        issuer=issuer,
        audience=token_url,
        claims=claims,
        subject=subject,
        key=key,
        header=header,
    )

scopes = [
    'https://www.googleapis.com/auth/spreadsheets',
    'https://www.googleapis.com/auth/drive',
]
session = create_assertion_session('your-google-conf.json', scopes)
gc = Client(None, session)

wks = gc.open("Where is the money Lebowski?").sheet1

wks.update_acell('B2', "it's down there somewhere, let me take another look.")

# Fetch a cell range
cell_list = wks.range('A1:B7')
```

CHAPTER 6

API Documentation

6.1 API Reference

6.1.1 Top level

gspread.**oauth**(*scopes=['https://www.googleapis.com/auth/spreadsheets', 'https://www.googleapis.com/auth/drive'], flow=<function local_server_flow>, credentials_filename='/home/docs/.config/gspread/credentials.json', authorized_user_filename='/home/docs/.config/gspread/authorized_user.json'*)

Authenticate with OAuth Client ID.

By default this function will use the local server strategy and open the authorization URL in the user's browser:

```
gc = gspread.oauth()
```

Another option is to run a console strategy. This way, the user is instructed to open the authorization URL in their browser. Once the authorization is complete, the user must then copy & paste the authorization code into the application:

```
gc = gspread.oauth(flow=gspread.auth.console_flow)
```

`scopes` parameter defaults to read/write scope available in `gspread.auth.DEFAULT_SCOPES`. It's read/write for Sheets and Drive API:

```
DEFAULT_SCOPES =[
    'https://www.googleapis.com/auth/spreadsheets',
    'https://www.googleapis.com/auth/drive'
]
```

You can also use `gspread.auth.READONLY_SCOPES` for read only access. Obviously any method of `gspread` that updates a spreadsheet **will not work** in this case:

```
gc = gspread.oauth(scopes=gspread.auth.READONLY_SCOPES)

sh = gc.open("A spreadsheet")
sh.sheet1.update('A1', '42')    # <-- this will not work
```

If you're storing your user credentials in a place other than the default, you may provide a path to that file like so:

```
gc = gspread.oauth(
    credentials_filename='/alternative/path/credentials.json',
    authorized_user_filename='/alternative/path/authorized_user.json',
)
```

Parameters

- **scopes** (*list*) – The scopes used to obtain authorization.
- **flow** (*function*) – OAuth flow to use for authentication. Defaults to *local_server_flow()*
- **credentials_filename** (*str*) – Filepath (including name) pointing to a credentials *.json* file. Defaults to DEFAULT_CREDENTIALS_FILENAME:
  - *%APPDATA%gspreadcredentials.json* on Windows
  - *~/.config/gspread/credentials.json* everywhere else
- **authorized_user_filename** (*str*) – Filepath (including name) pointing to an authorized user *.json* file. Defaults to DEFAULT_AUTHORIZED_USER_FILENAME:
  - *%APPDATA%gspreadauthorized_user.json* on Windows
  - *~/.config/gspread/authorized_user.json* everywhere else

Return type *gspread.Client*

gspread.**service_account**(*filename='/home/docs/.config/gspread/service_account.json', scopes=['https://www.googleapis.com/auth/spreadsheets', 'https://www.googleapis.com/auth/drive']*)

Authenticate using a service account.

`scopes` parameter defaults to read/write scope available in `gspread.auth.DEFAULT_SCOPES`. It's read/write for Sheets and Drive API:

```
DEFAULT_SCOPES =[
    'https://www.googleapis.com/auth/spreadsheets',
    'https://www.googleapis.com/auth/drive'
]
```

You can also use `gspread.auth.READONLY_SCOPES` for read only access. Obviously any method of `gspread` that updates a spreadsheet **will not work** in this case.

Parameters

- **filename** (*str*) – The path to the service account json file.
- **scopes** (*list*) – The scopes used to obtain authorization.

Return type *gspread.Client*

gspread.**authorize**(*credentials*, *client_class=<class 'gspread.client.Client'>*)
Login to Google API using OAuth2 credentials. This is a shortcut function which instantiates *client_class*. By default *gspread.Client* is used.

Returns *client_class* instance.

6.1.2 Client

class gspread.**Client**(*auth*, *session=None*)
An instance of this class communicates with Google API.

Parameters

- **auth** – An OAuth2 credential object. Credential objects created by google-auth.
- **session** – (optional) A session object capable of making HTTP requests while persisting some parameters across requests. Defaults to google.auth.transport.requests.AuthorizedSession.

```
>>> c = gspread.Client(auth=OAuthCredentialObject)
```

copy(*file_id*, *title=None*, *copy_permissions=False*, *folder_id=None*)
Copies a spreadsheet.

Parameters

- **file_id** (*str*) – A key of a spreadsheet to copy.
- **title** (*str*) – (optional) A title for the new spreadsheet.
- **copy_permissions** (*bool*) – (optional) If True, copy permissions from the original spreadsheet to the new spreadsheet.
- **folder_id** (*str*) – Id of the folder where we want to save the spreadsheet.

Returns a `Spreadsheet` instance.

New in version 3.1.0.

Note: If you're using custom credentials without the Drive scope, you need to add `https://www.googleapis.com/auth/drive` to your OAuth scope in order to use this method.

Example:

```
scope = [
    'https://www.googleapis.com/auth/spreadsheets',
    'https://www.googleapis.com/auth/drive'
]
```

Otherwise, you will get an `Insufficient Permission` error when you try to copy a spreadsheet.

---

create(*title*, *folder_id=None*)
Creates a new spreadsheet.

Parameters

- **title** (*str*) – A title of a new spreadsheet.
- **folder_id** (*str*) – Id of the folder where we want to save the spreadsheet.

Returns a `Spreadsheet` instance.

del_spreadsheet(*file_id*)
Deletes a spreadsheet.

Parameters **file_id** (*str*) – a spreadsheet ID (a.k.a file ID).

import_csv(*file_id*, *data*)
Imports data into the first page of the spreadsheet.

Parameters **data** (*str*) – A CSV string of data.

Example:

```
# Read CSV file contents
content = open('file_to_import.csv', 'r').read()

gc.import_csv(spreadsheet.id, content)
```

Note: This method removes all other worksheets and then entirely replaces the contents of the first worksheet.

insert_permission(*file_id*, *value*, *perm_type*, *role*, *notify=True*, *email_message=None*, *with_link=False*)
Creates a new permission for a file.

Parameters

- **file_id** (*str*) – a spreadsheet ID (aka file ID).
- **value** (*str, None*) – user or group e-mail address, domain name or None for 'default' type.
- **perm_type** (*str*) – (optional) The account type. Allowed values are: `user`, `group`, `domain`, `anyone`
- **role** (*str*) – (optional) The primary role for this user. Allowed values are: `owner`, `writer`, `reader`
- **notify** (*str*) – (optional) Whether to send an email to the target user/domain.
- **email_message** (*str*) – (optional) An email message to be sent if `notify=True`.
- **with_link** (*bool*) – (optional) Whether the link is required for this permission to be active.

Examples:

```
# Give write permissions to otto@example.com

gc.insert_permission(
    '0BmgG6nO_6dprnRRUWl1UFE',
    'otto@example.org',
    perm_type='user',
    role='writer'
)

# Make the spreadsheet publicly readable

gc.insert_permission(
    '0BmgG6nO_6dprnRRUWl1UFE',
    None,
```

(continues on next page)

(continued from previous page)

```
    perm_type='anyone',
    role='reader'
)
```

list_permissions(*file_id*)
Retrieve a list of permissions for a file.

Parameters **file_id** (*str*) – a spreadsheet ID (aka file ID).

open(*title*)
Opens a spreadsheet.

Parameters **title** (*str*) – A title of a spreadsheet.

Returns a `Spreadsheet` instance.

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

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

```
>>> gc.open('My fancy spreadsheet')
```

open_by_key(*key*)
Opens a spreadsheet specified by *key* (a.k.a Spreadsheet ID).

Parameters **key** (*str*) – A key of a spreadsheet as it appears in a URL in a browser.

Returns a `Spreadsheet` instance.

```
>>> gc.open_by_key('0BmgG6nO_6dprdS1MN3d3MkdPa142WFRrdnRRUWl1UFE')
```

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

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

Returns a `Spreadsheet` instance.

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

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

openall(*title=None*)
Opens all available spreadsheets.

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

Returns a list of `Spreadsheet` instances.

remove_permission(*file_id*, *permission_id*)
Deletes a permission from a file.

Parameters

- **file_id** (*str*) – a spreadsheet ID (aka file ID.)
- **permission_id** (*str*) – an ID for the permission.

6.1.3 Models

The models represent common spreadsheet entities: `a spreadsheet`, `a worksheet` and `a cell`.

Note: The classes described below should not be instantiated by the end-user. Their instances result from calling other objects' methods.

6.1.4 Utils

gspread.utils

This module contains utility functions.

gspread.utils.**rowcol_to_a1**(*row*, *col*)
Translates a row and column cell address to A1 notation.

Parameters

- **row** (*int, str*) – The row of the cell to be converted. Rows start at index 1.
- **col** – The column of the cell to be converted. Columns start at index 1.

Returns a string containing the cell's coordinates in A1 notation.

Example:

```
>>> rowcol_to_a1(1, 1)
A1
```

gspread.utils.**a1_to_rowcol**(*label*)
Translates a cell's address in A1 notation to a tuple of integers.

Parameters **label** (*str*) – A cell label in A1 notation, e.g. 'B1'. Letter case is ignored.

Returns a tuple containing *row* and *column* numbers. Both indexed from 1 (one).

Return type tuple

Example:

```
>>> a1_to_rowcol('A1')
(1, 1)
```

gspread.utils.**a1_range_to_grid_range**(*name*, *sheet_id=None*)
Converts a range defined in A1 notation to a dict representing a GridRange.

All indexes are zero-based. Indexes are half open, e.g the start index is inclusive and the end index is exclusive: [startIndex, endIndex).

Missing indexes indicate the range is unbounded on that side.

Examples:

```
>>> a1_range_to_grid_range('A1:A1')
```

{'startRowIndex': 0, 'endRowIndex': 1, 'startColumnIndex': 0, 'endColumnIndex': 1}

```
>>> a1_range_to_grid_range('A3:B4')
{'startRowIndex': 2, 'endRowIndex': 4, 'startColumnIndex': 0, 'endColumnIndex': 2}
```

```
>>> a1_range_to_grid_range('A:B')
{'startColumnIndex': 0, 'endColumnIndex': 2}
```

```
>>> a1_range_to_grid_range('A5:B')
{'startRowIndex': 4, 'startColumnIndex': 0, 'endColumnIndex': 2}
```

```
>>> a1_range_to_grid_range('A1')
{'startRowIndex': 0, 'endRowIndex': 1, 'startColumnIndex': 0, 'endColumnIndex': 1}
```

```
>>> a1_range_to_grid_range('A')
{'startColumnIndex': 0, 'endColumnIndex': 1}
```

```
>>> a1_range_to_grid_range('1')
{'startRowIndex': 0, 'endRowIndex': 1}
```

```
>>> a1_range_to_grid_range('A1', sheet_id=0)
{'sheetId': 0, 'startRowIndex': 0, 'endRowIndex': 1, 'startColumnIndex': 0,
↪'endColumnIndex': 1}
```

gspread.utils.**cast_to_a1_notation**(*method*)

Decorator function casts wrapped arguments to A1 notation in range method calls.

gspread.utils.**absolute_range_name**(*sheet_name*, *range_name=None*)

Return an absolutized path of a range.

```
>>> absolute_range_name("Sheet1", "A1:B1")
"'Sheet1'!A1:B1"
```

```
>>> absolute_range_name("Sheet1", "A1")
"'Sheet1'!A1"
```

```
>>> absolute_range_name("Sheet1")
"'Sheet1'"
```

```
>>> absolute_range_name("Sheet'1")
"'Sheet''1'"
```

```
>>> absolute_range_name("Sheet''1")
"'Sheet''''1'"
```

```
>>> absolute_range_name("'''sheet12'''", "A1:B2")
"'''''sheet12'''''!A1:B2"
```

gspread.utils.**is_scalar**(*x*)

Return True if the value is scalar.

A scalar is not a sequence but can be a string.

```
>>> is_scalar([])
False
```

```
>>> is_scalar([1, 2])
False
```

```
>>> is_scalar(42)
True
```

```
>>> is_scalar('nice string')
True
```

```
>>> is_scalar({})
True
```

```
>>> is_scalar(set())
True
```

gspread.utils.**filter_dict_values**(*D*)

Return a shallow copy of D with all *None* values excluded.

```
>>> filter_dict_values({'a': 1, 'b': 2, 'c': None})
{'a': 1, 'b': 2}
```

```
>>> filter_dict_values({'a': 1, 'b': 2, 'c': 0})
{'a': 1, 'b': 2, 'c': 0}
```

```
>>> filter_dict_values({})
{}
```

```
>>> filter_dict_values({'imnone': None})
{}
```

gspread.utils.**accepted_kwargs**(***default_kwargs*)

```
>>> @accepted_kwargs(d='d', e=None)
... def foo(a, b, c='c', **kwargs):
...     return {
...         'a': a,
...         'b': b,
...         'c': c,
...         'd': kwargs['d'],
...         'e': kwargs['e'],
...     }
...
```

```
>>> foo('a', 'b')
{'a': 'a', 'b': 'b', 'c': 'c', 'd': 'd', 'e': None}
```

```
>>> foo('a', 'b', 'NEW C')
{'a': 'a', 'b': 'b', 'c': 'NEW C', 'd': 'd', 'e': None}
```

```
>>> foo('a', 'b', e='Not None')
{'a': 'a', 'b': 'b', 'c': 'c', 'd': 'd', 'e': 'Not None'}
```

```
>>> foo('a', 'b', d='NEW D')
{'a': 'a', 'b': 'b', 'c': 'c', 'd': 'NEW D', 'e': None}
```

```
>>> foo('a', 'b', a_typo='IS DETECTED')
Traceback (most recent call last):
...
TypeError: foo got unexpected keyword arguments: ['a_typo']
```

```
>>> foo('a', 'b', d='NEW D', c='THIS DOES NOT WORK BECAUSE OF d')
Traceback (most recent call last):
...
TypeError: foo got unexpected keyword arguments: ['c']
```

6.1.5 Auth

gspread.auth

Simple authentication with OAuth.

gspread.auth.**local_server_flow**(*scopes*, *port=0*, *filename='/home/docs/.config/gspread/credentials.json'*)
Run an OAuth flow using a local server strategy.

Creates an OAuth flow and runs google_auth_oauthlib.flow.InstalledAppFlow.run_local_server. This will start a local web server and open the authorization URL in the user's browser.

Pass this function to `flow` parameter of *`oauth()`* to run a local server flow.

gspread.auth.**console_flow**(*scopes*, *filename='/home/docs/.config/gspread/credentials.json'*)
Run an OAuth flow using a console strategy.

Creates an OAuth flow and runs google_auth_oauthlib.flow.InstalledAppFlow.run_console.

Pass this function to `flow` parameter of *`oauth()`* to run a console strategy.

6.1.6 Exceptions

exception gspread.exceptions.**GSpreadException**
A base class for gspread's exceptions.

exception gspread.exceptions.**APIError**(*response*)

CHAPTER 7

How to Contribute

Please make sure to take a moment and read the Code of Conduct.

7.1 Ask Questions

The best way to get an answer to a question is to ask on Stack Overflow with a gspread tag.

7.2 Report Issues

Please report bugs and suggest features via the GitHub Issues.

Before opening an issue, search the tracker for possible duplicates. If you find a duplicate, please add a comment saying that you encountered the problem as well.

7.3 Contribute code

Please make sure to read the Contributing Guide before making a pull request.

CHAPTER 8

Indices and tables

- genindex
- modindex
- search

Python Module Index

g

Index