SheetSync Documentation

Release 0.2.5

Mark Brenig-Jones

Oct 03, 2017

Contents

A python library to create, update and delete rows of data in a google spreadsheet.

CHAPTER 1

Getting Started

SheetSync is a python library to create, update and delete rows of data in a google spreadsheet.

Installation

Install from PyPi using pip:

```
pip install sheetsync
```

Or you can clone the git repo and install from the code:

```
git clone git@github.com:mbrenig/sheetsync.git LocalSheetSync
pip install LocalSheetSync
```

Note, you may need to run the commands above with `sudo`.

Setting up OAuth 2.0 access

In May 2015 Google retired old API access methods, and recommended users migrate to OAuth 2.0. OAuth2.0 is better for security and privacy but it means getting started with sheetsync involves a bit of extra configuration.

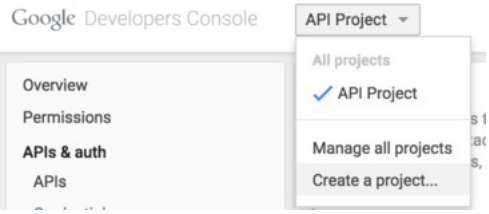
The steps below (written in 2015) guide you through API configuration and a simple script to manipulate a Google sheet. They will take around 20 minutes to complete.

> **Warning:** This tutorial is designed to get you using sheetsync quickly. It is **insecure** because your client secret is stored in plain text. If someone obtains your client secret, they could use it to consume your quota, incur charges or request access to user data.

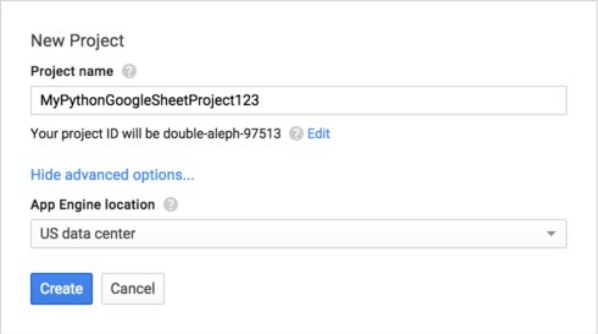
Before using sheetsync in production you should learn about Client IDs and replace the `ia_credentials_helper()` function with your own function that manages authentication and creates an OAuth2Credentials object.

New Project

Start by setting up a new project via Google's developer console, console.developers.google.com:

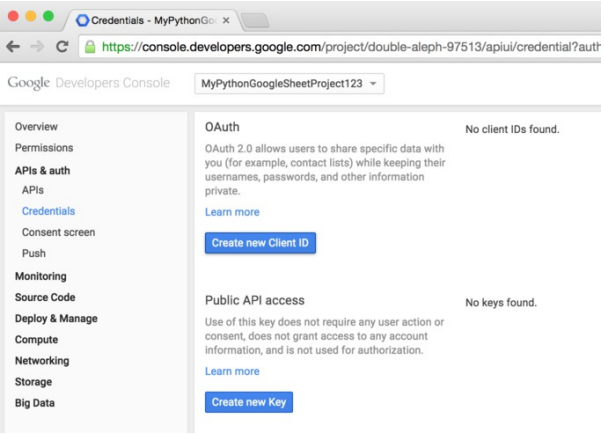


Pick a project name:

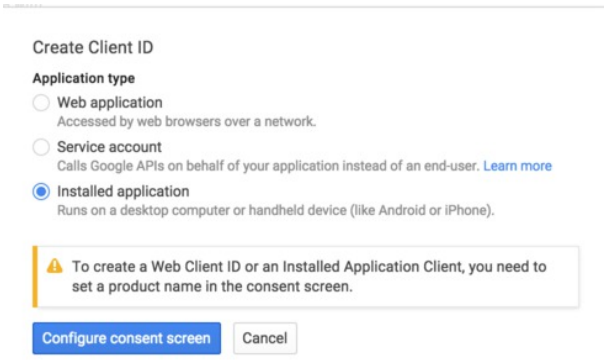


Create a new Client ID

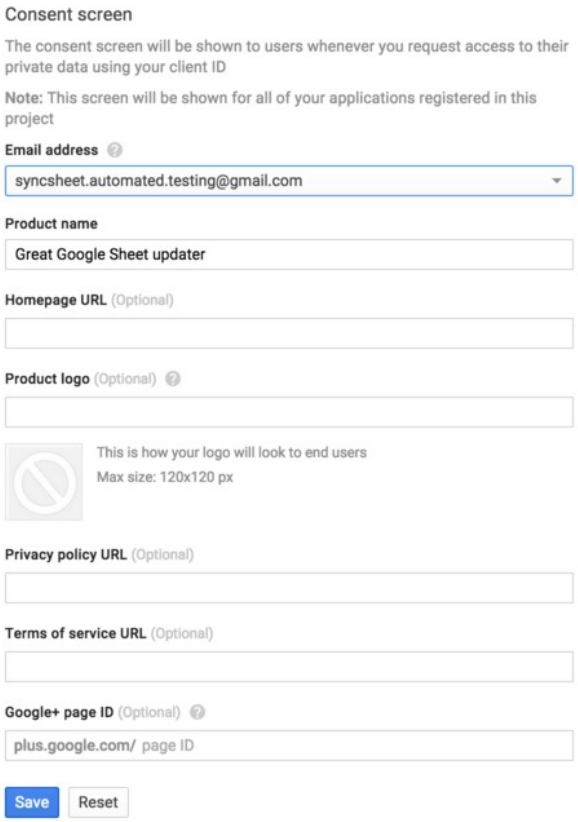
From your new project's configuration panel, in the console, select "Credentials" from the lefthand menu and then "Create new Client ID" for OAuth:



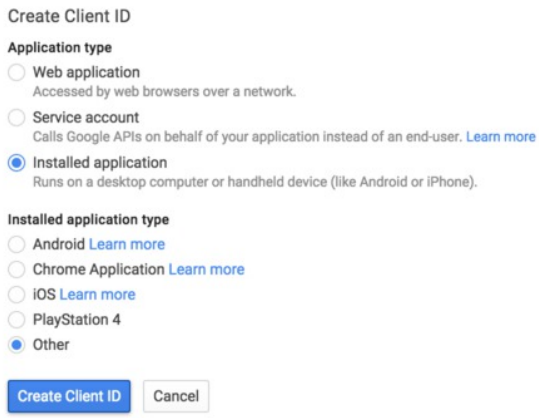
For this tutorial, choose the type Installed application:



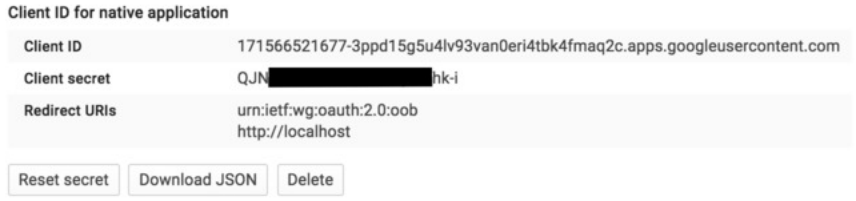
The consent screen is what users will see when the sheetsync script asks for access to their Google drive.



Finally select “Other” for Installed application type:

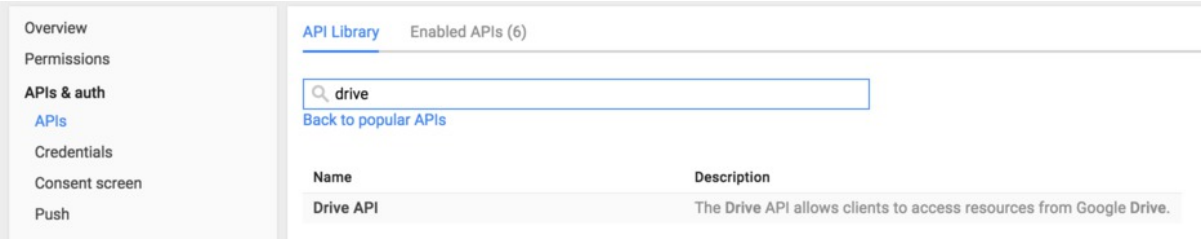


The steps above should have got to you a page that displays your new Client ID and Client Secret. For example:

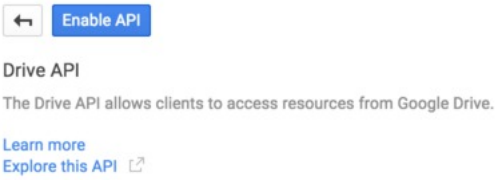


Enable Drive API

Next we need to associate Drive API access with these OAuth credentials. From the lefthand menu choose API and search for Drive:



Click through to the Drive API and "Enable API":



You're now ready to start using this Client ID information with sheetsync.

Injecting data to a Google sheet

sheetsync works with data in a dictionary of dictionaries. Each row is represented by a dictionary, and these are themselves stored in a dictionary indexed by a row-specific key. For example this dictionary represents two rows of

data each with columns “Color” and “Performer”:

```
data = { "Kermit": {"Color" : "Green", "Performer" : "Jim Henson"},
         "Miss Piggy" : {"Color" : "Pink", "Performer" : "Frank Oz"}
       }
```

To insert this data (add or update rows) into a target worksheet in a google spreadsheet doc use this code:

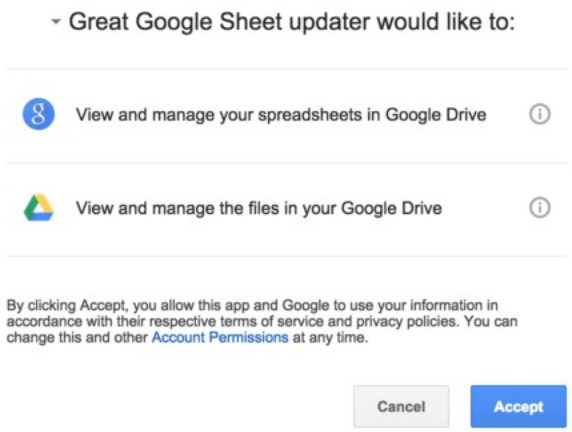
```
import logging
from sheetsync import Sheet, ia_credentials_helper
# Turn on logging so you can see what sheetsync is doing.
logging.getLogger('sheetsync').setLevel(logging.DEBUG)
logging.basicConfig()

# Create OAuth2 credentials, or reload them from a local cache file.
CLIENT_ID = '171566521677-3ppd15g5u4lv93van0eri4tbk4fmaq2c.apps.googleusercontent.com'
CLIENT_SECRET = 'QJN*****************hk-i'
creds = ia_credentials_helper(CLIENT_ID, CLIENT_SECRET,
                              credentials_cache_file='cred_cache.json')

data = { "Kermit": {"Color" : "Green", "Performer" : "Jim Henson"},
         "Miss Piggy" : {"Color" : "Pink", "Performer" : "Frank Oz"} }

# Find or create a spreadsheet, then inject data.
target = Sheet(credentials=creds, document_name="sheetsync Getting Started")
target.inject(data)
print "Spreadsheet created here: %s" % target.document_href
```

The first part of this script (lines 1-11) imports the `Sheet` object and `ia_credentials_helper` function. This function is included to help you quickly generate an OAuth2Credentials object using your Client ID and Secret.

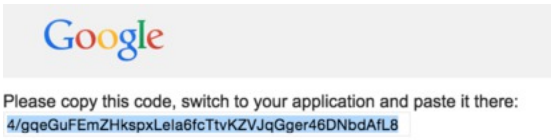
When the `ia_credentials_helper` function runs it will print a URL to allow you to grant the script access, like this:

```
Go to the following link in your browser:
https://accounts.google.com/o/oauth2/auth?scope=https%3A%2F%2Fwww.googleapis.com%2Fauth%2Fdrive+https
%3A%2F%2Fspreadsheets.google.com%2Ffeeds&redirect_uri=urn%3Aietf%3Awg%3Aoauth%3A2.0%3Aoob&response_ty
pe=code&client_id=171566521677-3ppd15g5u4lv93van0eri4tbk4fmaq2c.apps.googleusercontent.com&access_typ
e=offline
Enter verification code:
```

From this URL (you may have to log in to a Google Drive account) you will be prompted to give the API Client you set up in section 1.2, access to your documents:



After accepting you're presented with a verification code that you must paste back into the script:



At this point `ia_credentials_helper` also caches the credentials - so that you don't need to repeat this step on future runs of the script.

The later code defines the table data (lines 13,14) then line 17 creates a new spreadsheet document in your google drive. Finaly line 18 inserts the data resulting in:

	A	B	C
1	Key	Color	Performer
2	Miss Piggy	Pink	Frank Oz
3	Kermit	Green	Jim Henson

It also prints the URL of the google sheet so you can view the result for yourself.

Since you'll probably want to update this spreadsheet, take note of the spreadsheet's document key from the URL:

and then you can inject new data to the existing document by initializing the sheet as follows:

```
1 target = Sheet(credentials=creds,
2                document_key="1bnieREGAyXZ2TnhXgYrlacCIY09Q2IfGXNZbjsvj82M",
3                worksheet_name="Sheet1")
```

Note: The 'inject' method only adds or updates rows. If you want to delete rows from the spreadsheet to keep it in sync with the input data then use the 'sync' method described in the next section.

CHAPTER 2

Tutorial

Let's extend the example from Getting Started, and use more of sheetsync's features. (With apologies in advance to the Muppets involved).

Customizing the spreadsheet

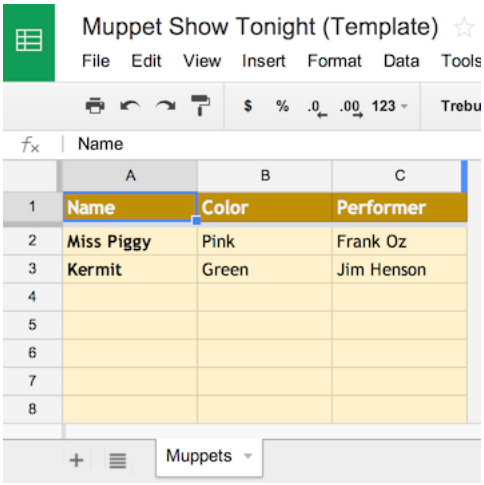
Key Column Headers

The first thing we'll fix is that top-left cell with the value 'Key'. The keys for our data are Names and the column header should reflect that. This is easy enough to do with the key_column_headers field:

```
target = sheetsync.Sheet(credentials=creds,
                         document_name="Muppet Show Tonight",
                         key_column_headers=["Name"])
```

Templates for Formatting

Google's spreadsheet API doesn't currently allow control over cell formatting, but you can specify a template spreadsheet that has the formatting you want - and use sheetsync to add data to a copy of the template. Here's a template spreadsheet created to keep my list of Muppets:



https://docs.google.com/spreadsheets/d/1J__SpvQvI9S4bW-BkA0PmPykH8VVT9bdoWZ-AW7V_0U/edit#gid=0

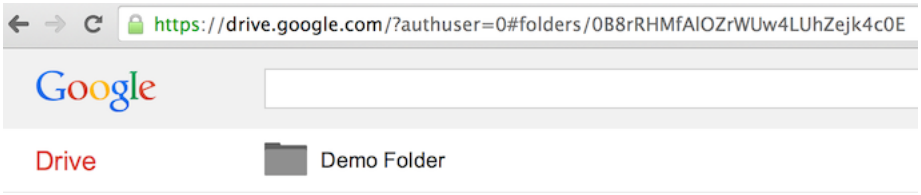
The template's document key is `1J__SpvQvI9S4bW-BkA0PmPykH8VVT9bdoWZ-AW7V_0U` we can instruct sheetsync to use this as a basis for the new spreadsheet it creates as follows:

```
target = sheetsync.Sheet(credentials=creds,
                         document_name="Muppet Show Tonight",
                         worksheet_name="Muppets",
                         template_key="1J__SpvQvI9S4bW-BkA0PmPykH8VVT9bdoWZ-AW7V_0U",
                         key_column_headers=["Name"])
```

Note that I've also specified the worksheet name in that example with the 'worksheet_name' parameter.

Folders

If you use folders to organize your Google drive, you can specify the folder a new spreadsheet will be created in. Use either the 'folder_name' or 'folder_key' parameters. Here for example I have a folder with the key `0B8rRHMfAlOZrWUw4LUhZejk4c0E`:



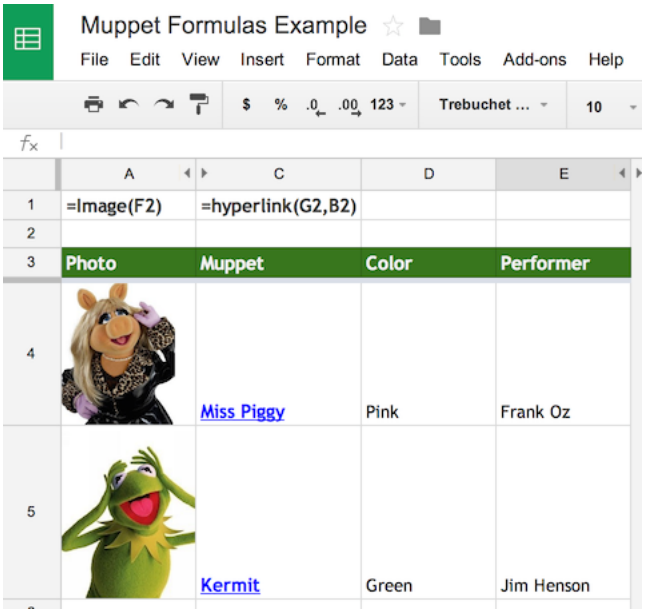
and instruct sheetsync to move the new spreadsheet into that folder with this code:

```
target = sheetsync.Sheet(credentials=creds,
                         document_name="Muppet Show Tonight",
                         worksheet_name="Muppets",
                         key_column_headers=["Name"],
                         template_key="1J__SpvQvI9S4bW-BkA0PmPykH8VVT9bdoWZ-AW7V_0U",
                         folder_key="0B8rRHMfAlOZrWUw4LUhZejk4c0E")
```

Formulas

Often you'll need some columns to contain formulas that depend on data in other columns, and when new rows are inserted by sheetsync, ideally you'd want those formulas to be added too. When initializing the spreadsheet you can

specify a row (typically above the header row) that contains reference formulas. Best illustrated by this example



https://docs.google.com/spreadsheets/d/1tn-lGqGHDrVbnW2PRvwie4LMmC9ZgYHWlbyTjCvwru8/edit#gid=0

Here row 2 contains formulas (Written out in row 1 for readability) that reference hidden columns. Row 3 contains the headers.

When new rows are added to this spreadsheet the 'Photo' and 'Muppet' columns will be populated with a formula similar to the reference row. Here are the parameters to set this up:

```
target = sheetsync.Sheet(credentials=creds,
                         document_key="1tn-lGqGHDrVbnW2PRvwie4LMmC9ZgYHWlbyTjCvwru8",
                         worksheet_name="Muppets",
                         key_column_headers=["Name"],
                         header_row_ix=3,
                         formula_ref_row_ix=2)

animal =  {'Animal': {'Color': 'Red',
                      'Image URL': 'http://upload.wikimedia.org/wikipedia/en/e/e7/
↪Animal_%28Muppet%29.jpg',
                      'Performer': 'Frank Oz',
                      'Wikipedia': 'http://en.wikipedia.org/wiki/Animal_(Muppet)'} }

target.inject(animal)
```

Synchronizing data

Until now all examples have used the 'inject' method to add data into a spreadsheet or update existing rows. As the name suggests, sheetsync also has a 'sync' method which will make sure the rows in the spreadsheet match the rows passed to the function. This might require that rows are deleted from the spreadsheet.

The default behavior is to not actually delete rows, but instead flag them for deletion with the text "(DELETED)" being appended to the values of the Key columns on rows to delete. This is to help recovery from accidental deletions. Full row deletion can be enabled by passing the flag_deletes argument as follows:

```
target = sheetsync.Sheet(credentials=creds,
                         document_key="1J__SABCD1234bW-ABCD1234kH8VABCD1234-AW7V_0U",
                         worksheet_name="Muppets",
                         key_column_headers=["Name"],
                         flag_deletes=False)

new_list = { 'Kermit' : { 'Color' : 'Green',
                          'Performer' : 'Jim Henson' },
             'Fozzie Bear' : {'Color' : 'Orange' } }

target.sync(new_list)
```

With rows for Miss Piggy and Kermit already in the spreadsheet, the sync function (in the example above) would remove Miss Piggy and add Fozzie Bear.

Taking backups

Warning: The sync function could delete a lot of data from your worksheet if the Key values get corrupted somehow. You should use the backup function to protect yourself from errors like this.

Some simple mistakes can cause bad results. For instance, if the key column headers on the spreadsheet don't match those passed to the Sheet constructor the sync method will delete all the existing rows and add new ones! You could protect rows and ranges to guard against this, but perhaps the simplest way to mitigate the risk is by creating a backup of your spreadsheet before syncing data. Here's an example:

```
target.backup("Backup of my important sheet. 16th June",
              folder_name = "sheetsync Backups.")
```

This code would take a copy of the entire spreadsheet that the Sheet instance 'target' belongs to, name it "Backup of my important sheet. 16th June", and move it to a folder named "sheetsync Backups.".

Debugging

sheetsync uses the standard python logging module, the easiest way to find out what's going on under the covers is to turn on all logging:

```
import sheetsync
import logging
# Set all loggers to DEBUG level..
logging.getLogger('').setLevel(logging.DEBUG)
# Register the default log handler to send logs to console..
logging.basicConfig()
```

If you find issues please raise them on github, and if you have fixes please submit pull requests. Thanks!

CHAPTER 3

The `sheetsync` package API

Sheet

class `sheetsync.`**`Sheet`**(*credentials=None*, *document_key=None*, *document_name=None*, *worksheet_name=None*, *key_column_headers=None*, *header_row_ix=1*, *formula_ref_row_ix=None*, *flag_deletes=True*, *protected_fields=None*, *template_key=None*, *template_name=None*, *folder_key=None*, *folder_name=None*)

Represents a single worksheet within a google spreadsheet.

This class tracks the google connection, the reference to the worksheet, as well as options controlling the structure of the data in the worksheet.. for .. rubric:: example

- Which row is used as the table header
- What header names should be used for the key column(s)
- Whether some columns are protected from overwriting

`document_key`

str – The spreadsheet's document key assigned by google drive. If you are using sheetsync to create a spreadsheet then use this attribute to saved the document_key, and make sure you pass it as a parameter in subsequent calls to __init__

`document_name`

str – The title of the google spreadsheet document

`document_href`

str – The HTML href for the google spreadsheet document

`__init__`(*credentials=None*, *document_key=None*, *document_name=None*, *worksheet_name=None*, *key_column_headers=None*, *header_row_ix=1*, *formula_ref_row_ix=None*, *flag_deletes=True*, *protected_fields=None*, *template_key=None*, *template_name=None*, *folder_key=None*, *folder_name=None*)

Creates a worksheet object (also creating a new Google sheet doc if required)

Parameters

- **credentials** (*OAuth2Credentials*) – Credentials object returned by the google authorization server. Described in detail in this article: https://developers.google.com/api-client-library/python/guide/aaa_oauth For testing and development consider using the *ia_credentials_helper* helper function
- **document_key** (*Optional) (str*) – Document key for the existing spreadsheet to sync data to. More info here: https://productforums.google.com/forum/#!topic/docs/XPOR9bTTS50 If this is not provided sheetsync will use document_name to try and find the correct spreadsheet.
- **document_name** (*Optional) (str*) – The name of the spreadsheet document to access. If this is not found it will be created. If you know the document_key then using that is faster and more reliable.
- **worksheet_name** (*str*) – The name of the worksheet inside the spreadsheet that data will be synced to. If omitted then the default name "Sheet1" will be used, and a matching worksheet created if necessary.
- **key_column_headers** (*Optional) (list of str*) – Data in the key column(s) uniquely identifies a row in your data. So, for example, if your data is indexed by a single username string, that you want to store in a column with the header 'Username', you would pass this:

  key_column_headers=['Username']

  However, sheetsync also supports component keys. Python dictionaries can use tuples as keys, for example if you had a tuple key like this:

  ('Tesla', 'Model-S', '2013')

  You can make the column meanings clear by passing in a list of three key_column_headers:

  ['Make', 'Model', 'Year']

  If no value is given, then the default behavior is to name the column "Key"; or "Key-1", "Key-2", ... if your data dictionaries keys are tuples.
- **header_row_ix** (*Optional) (int*) – The row number we expect to see column headers in. Defaults to 1 (the very top row).
- **formula_ref_row_ix** (*Optional) (int*) – If you want formulas to be added to some cells when inserting new rows then use a formula reference row. See *Formulas* for an example use.
- **flag_deletes** (*Optional) (bool*) – Specify if deleted rows should only be flagged for deletion. By default sheetsync does not delete rows of data, it just marks that they are deleted by appending the string " (DELETED)" to key values. If you pass in the value "False" then rows of data will be deleted by the sync method if they are not found in the input data. Note, use the inject method if you only want to add or modify data to in a worksheet.
- **protected_fields** (*Optional) (list of str*) – An list of fields (column headers) that contain protected data. sheetsync will only write to cells in these columns if they are blank. This can be useful if you are expecting users of the spreadsheet to colaborate on the document and edit values in certain columns (e.g. modifying a "Test result" column from "PENDING" to "PASSED") and don't want to overwrite their edits.
- **template_key** (*Optional) (str*) – This optional key references the spreadsheet that will be copied if a new spreadsheet needs to be created. This is useful for copying over formatting, a specific header order, or apps-script functions. See *Templates for Formatting*.

- **template_name** (*Optional) (str*) – As with template_key but the name of the template spreadsheet. If known, using the template_key will be faster.
- **folder_key** (*Optional) (str*) – This optional key references the folder that a new spreadsheet will be moved to if a new spreadsheet needs to be created.
- **folder_name** (*Optional) (str*) – Like folder_key this parameter specifies the optional folder that a spreadsheet will be created in (if required). If a folder matching the name cannot be found, sheetsync will attempt to create it.

backup(*backup_name*, *folder_key=None*, *folder_name=None*)

Copies the google spreadsheet to the backup_name and folder specified.

Parameters

- **backup_name** (*str*) – The name of the backup document to create.
- **folder_key** (*Optional) (str*) – The key of a folder that the new copy will be moved to.
- **folder_name** (*Optional) (str*) – Like folder_key, references the folder to move a backup to. If the folder can't be found, sheetsync will create it.

data(*as_cells=False*)

Reads the worksheet and returns an indexed dictionary of the row objects.

For example:

>>>print sheet.data()

{'Miss Piggy': {'Color': 'Pink', 'Performer': 'Frank Oz'}, 'Kermit': {'Color': 'Green', 'Performer': 'Jim Henson'}}

inject(*raw_data*, *row_change_callback=None*)

Use this function to add rows or update existing rows in the spreadsheet.

Parameters

- **raw_data** (*dict*) – A dictionary of dictionaries. Where the keys of the outer dictionary uniquely identify each row of data, and the inner dictionaries represent the field,value pairs for a row of data.
- **row_change_callback** (*Optional) (func*) – A callback function that you can use to track changes to rows on the spreadsheet. The row_change_callback function must take four parameters like so:

  change_callback(row_key, row_dict_before, row_dict_after, list_of_changed_keys)

Returns

A simple counter object providing statistics about the changes made by sheetsync.

Return type *UpdateResults* (object)

sync(*raw_data*, *row_change_callback=None*)

Equivalent to the inject method but will delete rows from the google spreadsheet if their key is not found in the input (raw_data) dictionary.

Parameters

- **raw_data** (*dict*) – See inject method
- **row_change_callback** (*Optional) (func*) – See inject method

Returns See inject method

Return type *UpdateResults* (object)

UpdateResults

class sheetsync.**UpdateResults**

A lightweight counter object that holds statistics about number of updates made after using the 'sync' or 'inject' method.

added
int – Number of rows added

changed
int – Number of rows changed

nochange
int – Number of rows that were not modified.

deleted
int – Number of rows deleted (which will always be 0 when using the 'inject' function)

ia_credentials_helper

sheetsync.**ia_credentials_helper**(*client_id*, *client_secret*, *credentials_cache_file='credentials.json'*, *cache_key='default'*)

Helper function to manage a credentials cache during testing.

This function attempts to load and refresh a credentials object from a json cache file, using the cache_key and client_id as a lookup.

If this isn't found then it starts an OAuth2 authentication flow, using the client_id and client_secret and if successful, saves those to the local cache. See *Injecting data to a Google sheet*.

Parameters

- **client_id** (*str*) – Google Drive API client id string for an installed app
- **client_secret** (*str*) – The corresponding client secret.
- **credentials_cache_file** (*str*) – Filepath to the json credentials cache file
- **cache_key** (*str*) – Optional string to allow multiple credentials for a client to be stored in the cache.

Returns A google api credentials object. As described here: https://developers.google.com/api-client-library/python/guide/aaa_oauth

Return type OAuth2Credentials

Index

Symbols

A

B

C

D

I

N

S

U