easy-google-docs Documentation

Release 0.2.3

David W Pettifor

Aug 08, 2023

Contents

CHAPTER 1

Quick Start

Install with:

```
pip install easy-google-docs
```

Import with:

```
from easygoogledocs import GoogleAPI
```

Authorize with either web-browser-based authentication:

```
from easygoogledocs import AUTH_TYPE_BROWSER
api = GoogleAPI(credentials_file='oauth_credentials.json')
api.authorize(authentication_type=AUTH_TYPE_BROWSER)
```

Or authorize with a service account:

```
from easygoogledocs import AUTH_TYPE_SERVICE_ACCOUNT
api = GoogleAPI(credentials_file='service_account_credentials.json')
api.authorize(authentication_type=AUTH_TYPE_SERVICE_ACCOUNT)
```

CHAPTER 2

Examples

2.1 Google APIs and Account Authentication Types

There are two types of authentication methods you can use for Google API access:

- Web-based authentication (authenticates as _your_ account)
- Service-account authentication (allows for non-interfacing scripting)

2.1.1 Web-Based Authentication

Pros:

- Everything you do will be done as *you* - no need to mess with permissions, etc.
- All files you upload will be to *your* drive space.
- More secure as you must authenticate as yourself every time the script runs

Cons:

- Requires human-interface for authentication every time you authenticate
- Cannot automate script/run on GUI-less server

2.1.2 Service-Account Authentication

Pros:

- *Can* automate scripts/requires no interfacing
- Gets its own drive space (also a *con*)

Cons:

- Gets its own drive space - this means all files uploaded are to *this* account and not *yours*. You then must share the file with yourself.

- All files existing in *your* drive space you want to work with must be shared with the service account before it can *see* them.

2.1.3 Steps to Get Started

Create A Project

1. Go to your Google Console to create a new project: [https://console.cloud.google.com/projectcreate]
2. Give your project a name ('Example Project') and click "Create"
3. You will be taken to the main dashboard [https://console.cloud.google.com/home/dashboard].

Enable APIs

1. Go to the "Google Drive API" dashboard: [https://console.cloud.google.com/apis/library/drive.googleapis.com] Make sure your project ('Example Project') is selected in the drop-down menu at the top.
2. Click "Enable"
3. Go to the "Google Sheets API" dashboard: [https://console.cloud.google.com/apis/library/sheets.googleapis.com] Make sure your project ('Example Project') is selected in the drop-down menu at the top.
4. Click "Enable"

Create Credentials

Seriously consider what kind of authentication method you would like to use. This will heavily depend on your project's needs, and may mean more or less work in your code.

1. Go to [https://console.cloud.google.com/apis/credentials] to start creating credentials.
    - If you want to create a web-based authentication credential, select "OAuth client ID".
    - If you want to create a service-account credential, select "Service account key"

OAuth Client (Web Browser-based Authentication)

1. If this is your first OAuth client for this project, you must configure the consent screen here: [https://console.cloud.google.com/apis/credentials/consent?createClient]

2. Fill out the "Product name shown to users" field and click "Save".
3. This should take you back to the OAuth Client screen: [https://console.cloud.google.com/apis/credentials/oauthclient]
4. Select "Other" and give it a name (example: "My Script") and click "Create"
5. Close the pop-up window that shows your "Client ID" and "Secret"
6. On the far-right side of your new credential, click on the Download link to download your client credentails JSON file.
7. Keep this JSON file someplace secret! It will be needed to use this API.

Service-Account

1. On the "Create Service Account" page [https://console.cloud.google.com/apis/credentials/serviceaccountkey], click "Create Service Account".
2. Give your service account a name ('myserviceaccount'), verify the service account ID and copy the Email address generated under the service account ID field. This will be the email address you share folders with.
3. Click "Create and Continue"
4. Under "Role", select "Service Account Admin" and click "Done".
5. Click on the new service account email that now shows in the list, and click the "KEYS" tab.
6. Click "ADD KEY" -> "Create new key".
7. Make sure "JSON" is selected, and click "Create".
5. Save the returned credentials JSON file someplace safe! It will be needed to use this API.

2.2 Installation and Initialization

Once you have your credentials, you can install the package with:

```
pip install easy-google-docs
```

2.2.1 Importing and authentication

Using JSON File

Knowing where your credentails.json file is, you can initialize and authenticate with:

```
from easygoogledocs import GoogleAPI, AUTH_TYPE_BROWSER, AUTH_TYPE_SERVICE_ACCOUNT
api = GoogleAPI(credentials_file='credentials.json')

# To authorize with a web-browser based OAuth Token:
api.authorize(authentication_type=AUTH_TYPE_BROWSER)

# To authorize with a service account:
api.authorize(authentication_type=AUTH_TYPE_SERVICE_ACCOUNT)
```

Using JSON Object

Recommended for production deployment. In this case, we don't want to store our private key and ID in the JSON file, especially in plain text (and *especially* in a code repository!). In this case, we can load in the content of the JSON file (which has had the private key and ID fields stripped) and add them in using environment variables:

```
import json
from easygoogledocs import GoogleAPI, AUTH_TYPE_BROWSER, AUTH_TYPE_SERVICE_ACCOUNT

# load the cleaned JSON credentials file
json_auth_file = open('myproject-creds-clean.json', 'r')
json_credentials = json.loads(json_auth_file.read())
json_auth_file.close()
```

(continues on next page)

(continued from previous page)

```
# update the private key ID from an environment variable
json_credentials['private_key_id'] = os.getenv('GOOGLE_API_PRIVATE_KEY_ID')

# note: in some cases (GitHub stored secrets, for example) the `\n` character is␣
↪stored with an escaped backslash,
# so we need to clean this up as we load the private key, or authentication won't␣
↪work.
json_credentials['private_key'] = os.getenv('GOOGLE_API_PRIVATE_KEY').replace('\\n',
↪'\n')

# instead of using the `credentials_file` parameter, we can use the `credentials_
↪json` and pass in our JSON object
api = GoogleAPI(credentials_json=json_credentials)

# To authorize with a web-browser based OAuth Token:
api.authorize(authentication_type=AUTH_TYPE_BROWSER)

# To authorize with a service account:
api.authorize(authentication_type=AUTH_TYPE_SERVICE_ACCOUNT)
```

2.3 Uploading Documents to Drive

The document uploader will let Google try to figure out which kind of document it is. For most cases, this will work fine.

You can easily upload a file with:

```
from easygoogledocs import GoogleAPI, AUTH_TYPE_BROWSER, AUTH_TYPE_SERVICE_ACCOUNT
api = GoogleAPI(credentials_file='credentials.json')

# To authorize with a web-browser based OAuth Token:
api.authorize(authentication_type=AUTH_TYPE_BROWSER)

# To authorize with a service account:
api.authorize(authentication_type=AUTH_TYPE_SERVICE_ACCOUNT)

# Upload the file "MyFinalPaper.docx" to my root drive folder:
file_meta = api.upload_document(document_location="MyFinalPaper.docx")

# Or if you want to upload it to a specific folder, you'll need to know the folder's␣
↪ID:
file_meta = api.upload_document(document_location="MyFinalPaper.docx", parent_id=
↪'FOLDER_ID')
```

Or if you want to upload specifically to Google Sheets:

```
from easygoogledocs import GoogleAPI, AUTH_TYPE_BROWSER, AUTH_TYPE_SERVICE_ACCOUNT
api = GoogleAPI(credentials_file='credentials.json')

# To authorize with a web-browser based OAuth Token:
api.authorize(authentication_type=AUTH_TYPE_BROWSER)

# To authorize with a service account:
```

(continues on next page)

(continued from previous page)

```
api.authorize(authentication_type=AUTH_TYPE_SERVICE_ACCOUNT)

# Upload the file "Expenses.xlsx" to my root drive folder:
file_meta = api.upload_spreadsheet(document_location="Expenses.xlsx")

# Or if you want to upload it to a specific folder, you'll need to know the folder's␣
↪ID:
file_meta = api.upload_spreadsheet(document_location="Expenses.xlsx", parent_id=
↪'FOLDER_ID')
```

2.3.1 Specifying Mimetypes

You can upload a file and specify which mime-type the file is if you would rather not let Google try to figure it out:

```
from easygoogledocs import GoogleAPI, AUTH_TYPE_BROWSER, AUTH_TYPE_SERVICE_ACCOUNT
api = GoogleAPI(credentials_file='credentials.json')

# To authorize with a web-browser based OAuth Token:
api.authorize(authentication_type=AUTH_TYPE_BROWSER)

# To authorize with a service account:
api.authorize(authentication_type=AUTH_TYPE_SERVICE_ACCOUNT)

# Upload the file "leafy_autumn.jpg" to my root drive folder:
file_meta = api.upload_file(document_location="leafy_autumn.jpg", mime_type=
↪'application/vnd.google-apps.photo')
```

Supported Mime-Types

You can view a compelte list of supported mime-types (and where Google will default the opening application) at:

https://developers.google.com/drive/v3/web/mime-types

2.4 Deleting Files from Google Drive

You can delete any file from Google Drive so long as you know the File ID:

```
from easygoogledocs import GoogleAPI, AUTH_TYPE_BROWSER, AUTH_TYPE_SERVICE_ACCOUNT
api = GoogleAPI(credentials_file='credentials.json')

# To authorize with a web-browser based OAuth Token:
api.authorize(authentication_type=AUTH_TYPE_BROWSER)

# To authorize with a service account:
api.authorize(authentication_type=AUTH_TYPE_SERVICE_ACCOUNT)

# Delete the file "budget1999.xlsx", which has a file ID of "XXXXXXXXXXXXXXXXXXX":
api.delete_file(file_id='XXXXXXXXXXXXXXXXXXX')
```

There is also one for spreadsheets, which does nothing more than call the above function:

```
from easygoogledocs import GoogleAPI, AUTH_TYPE_BROWSER, AUTH_TYPE_SERVICE_ACCOUNT
api = GoogleAPI(credentials_file='credentials.json')

# To authorize with a web-browser based OAuth Token:
api.authorize(authentication_type=AUTH_TYPE_BROWSER)

# To authorize with a service account:
api.authorize(authentication_type=AUTH_TYPE_SERVICE_ACCOUNT)

# Delete the file "budget1999.xlsx", which has a file ID of "XXXXXXXXXXXXXXXXXX":
api.delete_spreadsheet(spreadsheet_id='XXXXXXXXXXXXXXXXXX')
```

2.5 Folders

Folders in Google Drive are nothing more than blank files with a specific type. They are given an ID so any other file can use that ID as part of their "parent" field.

2.5.1 Creating Folders

To create a folder:

```
from easygoogledocs import GoogleAPI, AUTH_TYPE_BROWSER, AUTH_TYPE_SERVICE_ACCOUNT
api = GoogleAPI(credentials_file='credentials.json')

# To authorize with a web-browser based OAuth Token:
api.authorize(authentication_type=AUTH_TYPE_BROWSER)

# To authorize with a service account:
api.authorize(authentication_type=AUTH_TYPE_SERVICE_ACCOUNT)

# Create a folder named "My Stuff":
mystuff_meta = api.create_folder(folder_name='My Stuff')
```

2.5.2 Creating Sub-folders

Knowing the folder's meta data, we can also create sub-folders by passing in the "Parent ID" of the previous folder. So to create a sub-folder "Photos" under the previous "My Stuff":

```
photo_folder_meta = api.create_folder(folder_name='Photos', parent_id=mystuff_meta['id
↪'])
```

2.5.3 Getting Parent Folder Information:

You may also obtain the parent folder's information if you know your file's data. In this example, we can verify that the "Photos" folder has the "My Stuff" folder as its parent:

```
parent_meta = api.get_parent_ids(file_id=photo_folder_meta['id'])

print(parent_meta)  # Should print the same ID as the one we find in `mystuff_meta['id
↪']`
```

2.6 Spreadsheets (Google Sheets)

2.6.1 Downloading/Exporting Spreadsheets

Exporting spreadsheets from Google Sheets is actually done through the Google Drive API (and not the Google Sheets API as one might expect). Only two things are required for this:

- Spreadsheet ID (File ID)
- Format (which can be any of the following defined formats in this library):
  - FORMAT_MS_EXCEL (format: '.xlsx')
  - FORMAT_OPEN_OFFICE_SHEET (format: '.ods')
  - FORMAT_PDF (format: '.pdf')
  - FORMAT_CSV (format: '.csv' - single sheet/tab only)
  - FORMAT_TSV (format: '.tsv' - single sheet/tab only)
  - FORMAT_HTML (format: '.zip' file of each sheet/tab as an HTML file/table)

Download As Excel File

You can download the entire file as a MS Excel Spreadsheet File:

```
from easygoogledocs import GoogleAPI, AUTH_TYPE_BROWSER, AUTH_TYPE_SERVICE_ACCOUNT,␣
↪FORMAT_MS_EXCEL
api = GoogleAPI(credentials_file='credentials.json')

# To authorize with a web-browser based OAuth Token:
api.authorize(authentication_type=AUTH_TYPE_BROWSER)

# To authorize with a service account:
api.authorize(authentication_type=AUTH_TYPE_SERVICE_ACCOUNT)

# Download the spreadsheet with a File ID "XXXXXXXXXXXXXXXXXX" to the Desktop
file_path = api.download_spreadsheet_to_file(spreadsheet_id='XXXXXXXXXXXXXXXXXX',␣
↪download_location='~/Desktop/', format=FORMAT_MS_EXCEL)
```

Download Specific Tab as CSV

Sometimes we only one a single sheet/tab of our workbook, especially when we're downloading it as a CSV file. You can do this by specifying either the tab index (0-based index, so "tab_index=0" is the first tab), or by the tab's name. Here, we download it by index:

```
from easygoogledocs import GoogleAPI, AUTH_TYPE_BROWSER, AUTH_TYPE_SERVICE_ACCOUNT,␣
↪FORMAT_CSV
api = GoogleAPI(credentials_file='credentials.json')

# To authorize with a web-browser based OAuth Token:
api.authorize(authentication_type=AUTH_TYPE_BROWSER)

# To authorize with a service account:
api.authorize(authentication_type=AUTH_TYPE_SERVICE_ACCOUNT)

```

(continues on next page)

(continued from previous page)

```
# Download the second tab/sheet of the spreadsheet with a File ID "XXXXXXXXXXXXXXXXXX
↪" to the Desktop
file_path = api.download_spreadsheet_to_file(spreadsheet_id='XXXXXXXXXXXXXXXXXX',␣
↪download_location='~/Desktop/', format=FORMAT_CSV, tab_index=1)
```

The above code works around a bit of what the Google Drive API v3 supports (as they do not support exporting other tabs as CSV natively). It actually downloads the file as an Excel file (to get all tabs), and uses the Python library xlrd to extract the specified tab's data (all in memory), then writes it to disk.

Download Spreadsheet in Memory

If you would rather get the data from your spreadsheet into memory-only (and avoid writing to disk immediately), you can:

```
from easygoogledocs import GoogleAPI, AUTH_TYPE_BROWSER, AUTH_TYPE_SERVICE_ACCOUNT,␣
↪FORMAT_CSV
api = GoogleAPI(credentials_file='credentials.json')

# To authorize with a web-browser based OAuth Token:
api.authorize(authentication_type=AUTH_TYPE_BROWSER)

# To authorize with a service account:
api.authorize(authentication_type=AUTH_TYPE_SERVICE_ACCOUNT)

# Download the second tab/sheet of the spreadsheet with a File ID "XXXXXXXXXXXXXXXXXX
↪" to memory (returns a io.ByteIO object
sheet_data = api.download_spreadsheet(spreadsheet_id='XXXXXXXXXXXXXXXXXX',␣
↪format=FORMAT_CSV, tab_index=1)
```

Note: the above function does support all export types, not just CSV! Also, before returning, the ByteIO object returned is reset with *.seek(0)* so it is ready to be read at the beginning of the byte data.

2.6.2 Clearing Spreadsheets

If you want to clear the contents of a specific tab/sheet of a spreadsheet, this can easily be done with:

```
from easygoogledocs import GoogleAPI, AUTH_TYPE_BROWSER, AUTH_TYPE_SERVICE_ACCOUNT
api = GoogleAPI(credentials_file='credentials.json')

# To authorize with a web-browser based OAuth Token:
api.authorize(authentication_type=AUTH_TYPE_BROWSER)

# To authorize with a service account:
api.authorize(authentication_type=AUTH_TYPE_SERVICE_ACCOUNT)

# Clear the contents of the second tab/sheet:
api.clear_spreadsheet(spreadsheet_id='XXXXXXXXXXXXXXXXXX', tab_index=1)
```

Note: this clears *only the contents* and not the formatting of each cell. All cell formatting (Date/Dollars/Numeric formatting) will remain.

2.6.3 Replacing Spreadsheets with Content

Sometimes we have updates we want to push to a specific spreadsheet tab and it's just easiest to replace the entire sheet's tab with this new data.

Replacing with Raw Data

If you have row data as a 2-dimensional array (a list of lists...the first list representing rows, each second list the column data):

```
from easygoogledocs import GoogleAPI, AUTH_TYPE_BROWSER, AUTH_TYPE_SERVICE_ACCOUNT
api = GoogleAPI(credentials_file='credentials.json')

# To authorize with a web-browser based OAuth Token:
api.authorize(authentication_type=AUTH_TYPE_BROWSER)

# To authorize with a service account:
api.authorize(authentication_type=AUTH_TYPE_SERVICE_ACCOUNT)

# Example of what your row data could look like:
row_data = [
    ['Name', 'Address', 'State'],    # first row -- header
    ['John', '123 Fake Street', 'AL']    # second row
    ['Jane', '456 Fake Street', 'AK']    # third row
]

# Replace the second tab with our row data:
api.replace_spreadsheet_with_rows(spread_sheet_id='XXXXXXXXXXXXXXXXXX', row_data=row_
↪data, tab_index=1)
```

Replacing with CSV

Or sometimes you may just have a CSV file you want to replace the sheet with:

```
from easygoogledocs import GoogleAPI, AUTH_TYPE_BROWSER, AUTH_TYPE_SERVICE_ACCOUNT
api = GoogleAPI(credentials_file='credentials.json')

# To authorize with a web-browser based OAuth Token:
api.authorize(authentication_type=AUTH_TYPE_BROWSER)

# To authorize with a service account:
api.authorize(authentication_type=AUTH_TYPE_SERVICE_ACCOUNT)

# Replace the second tab with the CSV file on our desktop:
api.replace_spreadsheet_with_csv(spread_sheet_id='XXXXXXXXXXXXXXXXXX', csv_file_
↪location='~/Desktop/data.csv', tab_index=1)
```

Note: for both _replacing_ functions, you can pass in an *input_type* parameter of:

- INPUT_TYPE_RAW - forces Google Sheets to not try to analyze the input (leaves formatting alone)
- INPUT_TYPE_AUTO - lets Google Sheets auto-format things like dates, decimals, etc.

2.6.4 Appending to Sheets

Appending Raw Data to a Specific Spreadsheet Tab

You can append to a spreadsheet (bottom of the table) with:

```
from easygoogledocs import GoogleAPI, AUTH_TYPE_BROWSER, AUTH_TYPE_SERVICE_ACCOUNT
api = GoogleAPI(credentials_file='credentials.json')

# To authorize with a web-browser based OAuth Token:
api.authorize(authentication_type=AUTH_TYPE_BROWSER)

# To authorize with a service account:
api.authorize(authentication_type=AUTH_TYPE_SERVICE_ACCOUNT)

# Example of what your row data may look like:
row_data = [
    ['John', '123 Fake Street', 'AL']   # first row to add
    ['Jane', '456 Fake Street', 'AK']   # second row to add
]

# Append the row_data to the second tab of the sheet "XXXXXXXXXXXXXXXXXX"
api.append_rows_to_spreadsheet(spreadsheet_id = 'XXXXXXXXXXXXXXXXXX' row_data=row_
↪data, tab_index=1)
```

If you want to append to a sheet with a different starting column (default is *A1*), you can specify the starting range:

```
# Example of what your row data may look like:
row_data = [
    ['John', '123 Fake Street', 'AL']   # first row to add
    ['Jane', '456 Fake Street', 'AK']   # second row to add
]

# Append the row_data to the second tab of the sheet "XXXXXXXXXXXXXXXXXX" starting in␣
↪the "B" column
api.append_rows_to_spreadsheet(spreadsheet_id = 'XXXXXXXXXXXXXXXXXX' row_data=row_
↪data, tab_index=1, starting_range='B1')
```

Note: we still use "x1" to define the starting range even though we are not actually starting at *Row 1*. With the *append* function, it will always insert into the spreadsheet after the last row which contains data.

Appending a CSV to a Specific Spreadsheet Tab

You can append to a spreadsheet (bottom of the table) with:

```
from easygoogledocs import GoogleAPI, AUTH_TYPE_BROWSER, AUTH_TYPE_SERVICE_ACCOUNT
api = GoogleAPI(credentials_file='credentials.json')

# To authorize with a web-browser based OAuth Token:
api.authorize(authentication_type=AUTH_TYPE_BROWSER)

# To authorize with a service account:
api.authorize(authentication_type=AUTH_TYPE_SERVICE_ACCOUNT)

# Append the CSV "data.csv" to the second tab of the sheet "XXXXXXXXXXXXXXXXXX"
api.append_rows_to_spreadsheet(spreadsheet_id = 'XXXXXXXXXXXXXXXXXX' csv_file='data.
↪csv', tab_index=1)
```

Note: Both *append* functions also allow for *tab_name* to be used instead of *tab_index*. Also, you can specify

an *input_type* if you would rather the data be inserted as raw (INPUT_TYPE_RAW rather than the default INPUT_TYPE_AUTO)

2.7 Sharing

There are three levels of sharing a file:

- Read-only
- Read/Write
- Ownership – Note! Owernship cannot be transferred to different domains!

2.7.1 Sharing as Read-Only

You can share a file with a specific user (knowing their email address) as read-only with:

```
from easygoogledocs import GoogleAPI, AUTH_TYPE_BROWSER, AUTH_TYPE_SERVICE_ACCOUNT,␣
↪PERMISSION_READ
api = GoogleAPI(credentials_file='credentials.json')

# To authorize with a web-browser based OAuth Token:
api.authorize(authentication_type=AUTH_TYPE_BROWSER)

# To authorize with a service account:
api.authorize(authentication_type=AUTH_TYPE_SERVICE_ACCOUNT)

# share with "johndoe@gmail.com":
api.share_document(file_id='XXXXXXXXXXXXXXXXXX', recipient_email='johndoe@gmail.com',␣
↪share_permissions=PERMISSION_READ)
```

The user will then receive an email shortly with a link to view your document, but they will not be able to make changes to it.

2.7.2 Sharing with Read/Write

You can also give a person permission to make changes to your document with:

```
from easygoogledocs import GoogleAPI, AUTH_TYPE_BROWSER, AUTH_TYPE_SERVICE_ACCOUNT,␣
↪PERMISSION_WRITE
api = GoogleAPI(credentials_file='credentials.json')

# To authorize with a web-browser based OAuth Token:
api.authorize(authentication_type=AUTH_TYPE_BROWSER)

# To authorize with a service account:
api.authorize(authentication_type=AUTH_TYPE_SERVICE_ACCOUNT)

# share with "johndoe@gmail.com":
api.share_document(file_id='XXXXXXXXXXXXXXXXXX', recipient_email='johndoe@gmail.com',␣
↪share_permissions=PERMISSION_WRITE)
```

The user will then receive an email shortly with a link to edit your document.

2.7.3 Changing Ownership

You can give ownership to a file to a specific user (but it *must* be within the same domain as your account) with:

```
from easygoogledocs import GoogleAPI, AUTH_TYPE_BROWSER, AUTH_TYPE_SERVICE_ACCOUNT,␣
↪PERMISSION_OWNER
api = GoogleAPI(credentials_file='credentials.json')

# To authorize with a web-browser based OAuth Token:
api.authorize(authentication_type=AUTH_TYPE_BROWSER)

# To authorize with a service account:
api.authorize(authentication_type=AUTH_TYPE_SERVICE_ACCOUNT)

# move ownership to "johndoe@gmail.com":
api.share_document(file_id='XXXXXXXXXXXXXXXXXX', recipient_email='johndoe@gmail.com',␣
↪share_permissions=PERMISSION_OWNER)
```

Note: only the user will then be able to control sharing access and change ownerships. Once you let go of ownership, you cannot take it back yourself!

2.7.4 Service Account Ownership

It might be recommended to set the *auto_ownership* flag in the initialization to *True* and define your email when using a service account. When you upload a file using a service account authentication method, the file will be *owned by that account, and not you!*. In fact, unless you specifically share the file with yourself, you will not be able to see the file you just uploaded.

However, you can have any uploaded file be automatically transferred to you being the owner by setting the right flags upon initialization of your API:

```
from easygoogledocs import GoogleAPI
api = GoogleAPI(credentials_file='credentials.json', auto_ownership=True, default_
↪owner_email='YOUREMAIL@gmail.com')
```

Doing this will automatically move ownership to the email address in *default_owner_email* any time a new file is created/uploaded.

2.7.5 Revoking Access

You can revoke access from a user to a specific file with:

```
from easygoogledocs import GoogleAPI, AUTH_TYPE_BROWSER, AUTH_TYPE_SERVICE_ACCOUNT
api = GoogleAPI(credentials_file='credentials.json')

# To authorize with a web-browser based OAuth Token:
api.authorize(authentication_type=AUTH_TYPE_BROWSER)

# To authorize with a service account:
api.authorize(authentication_type=AUTH_TYPE_SERVICE_ACCOUNT)

# Revoke permission for "johndoe@gmail.com" from accessing the file
api.revoke_permission(file_id='XXXXXXXXXXXXXXXXXX', user_emal='johndoe@gmail.com')
```

Note: this will remove *ALL* permissions that *johndoe@gmail.com* may have on the file. You cannot revoke Ownership.

CHAPTER 3

Code Documentation

3.1 GoogleAPI Code Documentation

CHAPTER 4

Indices and tables

- genindex
- search