Hyou Documentation

Release 3.0b2

Shuhei Takahashi

May 07, 2017

Contents

Hyou provides a simple Pythonic interface to access your Google Spreadsheet data.

CHAPTER 1

Synopsis

```
import hyou

# Login to Google Spreadsheet with credentials
collection = hyou.login('/path/to/credentails.json')

# Open a spreadsheet by ID
spreadsheet = collection['1ZYeIFccacgHkL0TPfdgXiMfPCuEEWUtbhXvaB9HBDzQ']
print spreadsheet.title          # => "Hyou Test Sheet"

# Open a worksheet in a spreadsheet by sheet name
worksheet = spreadsheet['Sheet1']
print worksheet.title            # => "Sheet1"
print worksheet.rows             # => 5
print worksheet.cols             # => 3

# Worksheet objects can be accessed just like two-dimensional lists
print worksheet[1][0]            # => "banana"
print worksheet[1][1]            # => "50"

# Call Worksheet.commit() to apply changes
worksheet[2][0] = 'cinamon'
worksheet[2][1] = 40
worksheet.commit()
```

CHAPTER 2

User Guide

Installation

Hyou can be installed from pypi with pip.

```
$ sudo pip install hyou
or
$ pip install --user hyou
```

Source code is available on GitHub.

https://github.com/google/hyou

Preparing Credentials

The first step is to prepare a credential you access Google Spreadsheet with.

There are three options:

1. Authorize as your Google account with OAuth2 using a shared application project.
2. Authorize as your Google account with OAuth2 using your own application project.
3. Authorize as a service account (a bot account not associated with any Google account).

If you just want to access your spreadsheet programatically, 1 is the safe and easy way. In other options, you need some steps to register an application at Google Developer Console. (TODO(nya): Describe those options too)

To authorize as your Google account with a shared application project, run `generate_oauth2_credentials.py`.

```
$ generate_oauth2_credentials.py ~/.drive.json
Please visit this URL to get the authorization code:
https://accounts.google.com/o/oauth2/auth?scope=...
```

```

Code:_
```

Open the URL with a web browser, click "Accept" button, copy-and-paste the authorization code to the console and hit enter. Then the credential JSON is saved to the specified file.

Keep the credential file in a safe location. With the credentials, all your Google Drive documents can be accessed.

Once you prepared a credential JSON file, it is very simple to connect to Google Spreadsheet service using it:

```
collection = hyou.login('/path/to/credentails.json')
```

Working with Collections

A `Collection` object represents a set of Google Spreadsheet documents. It is a dictionary-like object, whose key is spreadsheet ID and value is a `Spreadsheet` object.

You can enumerate the spreadsheets you own by accessing a `Collection` object like a dictionary.

```
for id, spreadsheet in collection.iteritems():
    print id, spreadsheet.title
```

If you know a spreadsheet ID, you can open it just by indexing. This is faster than iterating through `Collection` because it does not fetch the list of spreadsheets. For example, to open https://docs.google.com/spreadsheets/d/1ZYeIFccacgHkL0TPfdgXiMfPCuEEWUtbhXvaB9HBDzQ/edit :

```
spreadsheet = collection['1ZYeIFccacgHkL0TPfdgXiMfPCuEEWUtbhXvaB9HBDzQ']
```

Working with Spreadsheets

A `Spreadsheet` object is an ordered dictionary-like object, whose key is a worksheet title and value is a `Worksheet` object.

```
worksheet = spreadsheet['Sheet1']
```

It also behaves just like a list when accessed with integer indices since it is ordered.

```
worksheet = spreadsheet[0]  # Open the first worksheet
```

To add or delete worksheets, use `Spreadsheet.add_worksheet()` and `Spreadsheet.delete_worksheet()`.

```
new_worksheet = spreadsheet.add_worksheet('worksheet title', rows=1000, cols=26)
spreadsheet.delete_worksheet('worksheet title')
```

`Spreadsheet.title` read-write property holds the title of the spreadsheet.

```
print spreadsheet.title  # => "Current spreadsheet name"
spreadsheet.title = 'New spreadsheet name'
```

Working with Worksheets

A *Worksheet* object can be accessed just like two-dimensional string lists.

```
for i, row in enumerate(worksheet):
    print i, row[0], '/'.join(row[1:])
```

A cell value is a bare input string, represented as a unicode string (`str` in Python 3, `unicode` in Python 2).

- Numbers are converted to strings.
- Formulas (e.g. *"=SUM(A2:A)"*) are never expanded, and returned as-is.

Inversely, you can create a formula cell by writing a formula string like *"=SUM(A2:A)"*.

If you attempt to write a non-string value (e.g. numbers) to a cell, it is automatically converted to a string.

```
worksheet[0][0] = 7
print type(worksheet[0][0])  # => str in Python 3, unicode in Python 2
```

Writes to cells are never committed until *Worksheet.commit()* is called. You can use *with statements* to make sure *Worksheet.commit()* is called:

```
with worksheet:
    worksheet[0][0] = 'apple'
    worksheet[1][0] = 'banana'
    worksheet[2][0] = 'cinamon'
# Changes have been committed at this point
```

Cache Behavior

To reduce network traffic and round-trips, data is fetched on demand and cached. For example, calling `Worksheet.values()` first time takes some time to fetch data to servers, but subsequent calls return immediately because the server response is cached.

To clear the cache to access the up-to-date data, call `refresh()`.

Please be aware that any uncommitted writes to worksheet cells are discarded when `refresh()` is called.

As for *Worksheet*, all worksheet cells are fetched when a cell is attempted to read for the first time. This can be waste of time and bandwidth if you are interested in a subrange of a worksheet. In such case, you can use views described next.

Using Views

If you are interested in a subrange of a worksheet, you can use *WorksheetView* for efficiency to reduce the number of fetched cells. For example, this code snippet will create a 20x10 view of a worksheet:

```
view = worksheet.view(start_row=100, end_row=120, start_col=200, end_col=210)
assert view[0][0] == worksheet[100][200]
```

Each view has independent cache. Reading a cell of a view will fetch contained cells only, instead of all cells in the worksheet.

CHAPTER 3

API Reference

`hyou.`**`SCOPES`**

A tuple of strings representing the scopes needed to access spreadsheets. Use this constant to request OAuth2 credentials.

`hyou.`**`login`**(*json_path=None*, *json_text=None*)

Logs in to Google Spreadsheet, and returns a new `Collection` object.

Parameters

- **`json_path`** (`str`) – The filesystem path to a credential JSON file.
- **`json_text`** (`str`) – A credential JSON in text format.

Either one of *json_path* or *json_text* should be given.

This method accepts two formats of credential JSONs:

1. JSON file that serialized `oauth2client.client.Credentials`.
2. JSON file downloaded from Google Developer Console (for service accounts)

class `hyou.`**`Collection`**

Representation of your spreadsheet collection.

This is a dictionary-like object, implementing several dictionary methods like `keys()`, `values()`, `items()`, `iterkeys()`, `itervalues()`, `iteritems()`, `__len__()`, `__iter__()`. In contrast to usual `dict`, it is immutable (unless `refresh()` is called).

classmethod **`login`**(*json_path=None*, *json_text=None*)

An alias of `login()`.

`create_spreadsheet`(*title*, *rows=1000*, *cols=26*)

Creates a new spreadsheet, and returns a `Spreadsheet` instance.

Parameters

- **`title`** (`str`) – The title of a new spreadsheet.
- **`rows`** (`int`) – The number of rows of a new spreadsheet.

- **`cols`** (*`int`*) – The number of cols of a new spreadsheet.

Addition of a spreadsheet is committed immediately and *`refresh()`* is automatically called to reflect changes.

`refresh``()`

Discards the associated cache. See *Cache Behavior* for details.

class `hyou.`**`Spreadsheet`**

Representation of a spreadsheet.

This is a dictionary-like object, implementing several dictionary methods like `keys()`, `values()`, `items()`, `iterkeys()`, `itervalues()`, `iteritems()`, `__len__()`, `__iter__()`. In contrast to usual `dict`, it is immutable (unless *`refresh()`* is called), and elements are ordered.

Ordered values can by accessed by indices. That is, `obj[i]` is equivalent to `obj.values()[i]` when `i` is an integer.

`key`

The spreadsheet ID.

This property is read-only.

`title`

The title of the spreadsheet.

This property is writable. Writes are committed immediately and *`refresh()`* is automatically called to reflect changes.

`url`

The URL of the spreadsheet.

This property is read-only.

`updated`

The last update time of the spreadsheet as a `datetime.datetime` object.

This property is read-only.

`add_worksheet``(`*title*, *rows=100*, *cols=26*`)`

Adds a new worksheet and returns a new *`Worksheet`* object.

Parameters

- **`title`** (*`str`*) – The title of a new worksheet.
- **`rows`** (*`int`*) – The number of rows of a new worksheet.
- **`cols`** (*`int`*) – The number of cols of a new worksheet.

Addition of a worksheet is committed immediately and *`refresh()`* is automatically called to reflect changes.

`delete_worksheet``(`*title*`)`

Deletes a worksheet.

Parameters **`title`** (*`str`*) – The title of the worksheet to be deleted.

Deletion of a worksheet is committed immediately and *`refresh()`* is automatically called to reflect changes.

`refresh``()`

Discards the associated cache. See *Cache Behavior* for details.

class `hyou.`**`Worksheet`**

Representation of a worksheet.

This object behaves just like two-dimensional string lists. The first dimension is rows and the second is columns.

`title`

The title of the worksheet.

This property is writable. Writes are committed immediately and *`refresh()`* is automatically called to reflect changes.

`rows`

The number of rows of the worksheet.

This property is writable. Writes are committed immediately and *`refresh()`* is automatically called to reflect changes.

Use *`set_size()`* to change the number of both rows and columns simultaneously.

`cols`

The number of columns of the worksheet.

This property is writable. Writes are committed immediately and *`refresh()`* is automatically called to reflect changes.

Use *`set_size()`* to change the number of both rows and columns simultaneously.

`commit``()`

Commits writes to cells. Until this method is called, writes to cells never take effect.

`__enter__``()`

`__exit__``()`

These methods implements context manager protocol to make sure *`commit()`* is called.

`set_size``(`*rows*, *cols*`)`

Changes the dimension of the worksheet.

Parameters

- **`rows`** (*`int`*) – The new number of rows.
- **`cols`** (*`int`*) – The new number of cols.

Changes are committed immediately and *`refresh()`* is automatically called to reflect changes.

`view``(`*start_row=None*, *end_row=None*, *start_col=None*, *end_col=None*`)`

Creates a new *`WorksheetView`* representing a subrange of the worksheet.

Parameters

- **`start_row`** (*`integer`*) – The index of the first row included in a new view. Defaults to 0 if not specified.
- **`end_row`** (*`integer`*) – The index of the first row NOT included in a new view. Default to *`rows`* if not specified.
- **`start_col`** (*`integer`*) – The index of the first column included in a new view. Defaults to 0 if not specified.
- **`end_col`** (*`integer`*) – The index of the first column NOT included in a new view. Default to *`cols`* if not specified.

refresh()
Discards the associated cache. Please be aware that any uncommitted writes to cells are also discarded. See *Cache Behavior* for details.

class hyou.**WorksheetView**
Representation of a subrange of a worksheet.

Similarly as `Worksheet`, this object behaves just like two-dimensional string lists.

rows
The number of rows in this view. Read-only.

cols
The number of columns in this view. Read-only.

commit()
Commits writes to cells. Until this method is called, writes to cells never take effect.

__enter__()

__exit__()
These methods implements context manager protocol to make sure `commit()` is called.

refresh()
Discards the associated cache. Please be aware that any uncommitted writes to cells are also discarded. See *Cache Behavior* for details.

CHAPTER 4

Changelog

3.0.0 (2017-02-XX)

- Added Python 3.3+ support.
- Dropped Python 2.6 support.
- Switched to Sheets API v4.
- Now cell values are always represented as a unicode string even in Python 2.

2.1.1 (2016-07-04)

- Support oauth2client v2.0.0+.

2.1.0 (2015-10-28)

- Worksheets emulate standard lists better.
- Support Python 2.6.
- Bugfixes.

2.0.0 (2015-08-14)

- First stable release with 100% test coverage.

1.x

- Beta releases.

CHAPTER 5

Notices

Author

Shuhei Takahashi

- Website: https://nya3.jp/
- Twitter: https://twitter.com/nya3jp/

Disclaimer

This library is authored by a Googler and copyrighted by Google, but is not an official Google product.

License

Index