Flask-Excel Documentation

Release 0.0.7

Onni Software Ltd.

Jul 20, 2017

Contents

Author C.W.

Source code http://github.com/pyexcel/Flask-Excel.git

Issues http://github.com/pyexcel/Flask-Excel/issues

License New BSD License

Released 0.0.7

Generated Jul 20, 2017

Here is a typical conversation between the developer and the user:

```
User: "I have uploaded an excel file"
      "but your application says un-supported file format"
Developer: "Did you upload an xlsx file or a csv file?"
User: "Well, I am not sure. I saved the data using "
      "Microsoft Excel. Surely, it must be in an excel format."
Developer: "OK. Here is the thing. I were not told to support"
           "all available excel formats in day 1. Live with it"
           "or delay the project x number of days."
```

Flask-Excel is based on pyexcel and makes it easy to consume/produce information stored in excel files over HTTP protocol as well as on file system. This library can turn the excel data into a list of lists, a list of records(dictionaries), dictionaries of lists. And vice versa. Hence it lets you focus on data in Flask based web development, instead of file formats.

The idea originated from the common usability problem when developing an excel file driven web applications for non-technical office workers: such as office assistant, human resource administrator. The fact is that not all people know the difference among various excel formats: csv, xls, xlsx. Instead of training those people about file formats, this library helps web developers to handle most of the excel file formats by providing a common programming interface. To add a specific excel file format to you application, all you need is to install an extra pyexcel plugin. No code change to your application. Looking at the community, this library and its associated ones try to become a small and easy to install alternative to Pandas.

The highlighted features are:

1. excel data import into and export from databases
2. turn uploaded excel file directly into Python data structure
3. pass Python data structures as an excel file download
4. provide data persistence as an excel file in server side
5. supports csv, tsv, csvz, tsvz by default and other formats are supported via the following plugins:

Table 1: A list of file formats supported by external plugins

Package name	Supported file formats	Dependencies	Python versions
pyexcel-io	csv, csvz[1], tsv, tsvz[2]		2.6, 2.7, 3.3, 3.4, 3.5, 3.6 pypy
pyexcel-xls	xls, xlsx(read only), xlsm(read only)	xlrd, xlwt	same as above
pyexcel-xlsx	xlsx	openpyxl	same as above
pyexcel-xlsxw	xlsx(write only)	XlsxWriter	same as above
pyexcel-ods3	ods	ezodf, lxml	2.6, 2.7, 3.3, 3.4 3.5, 3.6
pyexcel-ods	ods	odfpy	same as above
pyexcel-odsr	ods(read only)	lxml	same as above
pyexcel-text	(write only)json, rst, mediawiki, html, latex, grid, pipe, orgtbl, plain simple	tabulate	2.6, 2.7, 3.3, 3.4 3.5, 3.6, pypy
pyexcel-handsontable	handsontable in html	handsontable	same as above
pyexcel-pygal	svg chart	pygal	2.7, 3.3, 3.4, 3.5 3.6, pypy
pyexcel-sortable	sortable table in html	csvtotable	same as above
pyexcel-gantt	gantt chart in html	frappe-gantt	except pypy, same as above

In order to manage the list of plugins installed, you need to use pip to add or remove a plugin. When you use virtualenv, you can have different plugins per virtual environment. In the situation where you have multiple plugins that does the same thing in your environment, you need to tell pyexcel which plugin to use per function call. For example, pyexcel-ods and pyexcel-odsr, and you want to get_array to use pyexcel-odsr. You need to append get_array(..., library='pyexcel-odsr').

This library makes information processing involving various excel files as easy as processing array, dictionary when processing file upload/download, data import into and export from SQL databases, information analysis and persistence. It uses **pyexcel** and its plugins:

1. to provide one uniform programming interface to handle csv, tsv, xls, xlsx, xlsm and ods formats.
2. to provide one-stop utility to import the data in uploaded file into a database and to export tables in a database as excel files for file download.
3. to provide the same interface for information persistence at server side: saving a uploaded excel file to and loading a saved excel file from file system.

[1] zipped csv file
[2] zipped tsv file

CHAPTER 1

Installation

You can install it via pip:

```
$ pip install Flask-Excel
```

or clone it and install it:

```
$ git clone https://github.com/pyexcel/Flask-Excel.git
$ cd Flask-Excel
$ python setup.py install
```

Installation of individual plugins , please refer to individual plugin page. For example, if you need xls file support, please install pyexcel-xls:

```
$ pip install pyexcel-xls
```

CHAPTER 2

Setup

In your application, you must import it before using it:

```
import flask_excel as excel

...
excel.init_excel(app) # required since version 0.0.7
```

CHAPTER 3

Quick start

A minimal application may look like this:

```
# -*- coding: utf-8 -*-
"""
tiny_example.py
:copyright: (c) 2015 by C. W.
:license: GPL v3 or BSD
"""
from flask import Flask, request, jsonify
import flask_excel as excel

app = Flask(__name__)


@app.route("/upload", methods=['GET', 'POST'])
def upload_file():
    if request.method == 'POST':
        return jsonify({"result": request.get_array(field_name='file')})
    return '''
    <!doctype html>
    <title>Upload an excel file</title>
    <h1>Excel file upload (csv, tsv, csvz, tsvz only)</h1>
    <form action="" method=post enctype=multipart/form-data><p>
    <input type=file name=file><input type=submit value=Upload>
    </form>
    '''


@app.route("/download", methods=['GET'])
def download_file():
    return excel.make_response_from_array([[1, 2], [3, 4]], "csv")


@app.route("/export", methods=['GET'])
def export_records():
```

```
    return excel.make_response_from_array([[1, 2], [3, 4]], "csv",
                                          file_name="export_data")


@app.route("/download_file_named_in_unicode", methods=['GET'])
def download_file_named_in_unicode():
    return excel.make_response_from_array([[1, 2], [3, 4]], "csv",
                                          file_name=u"")


# insert database related code here
if __name__ == "__main__":
    excel.init_excel(app)
    app.run()
```

The tiny application exposes four urls:

1. one for file upload
2. three urls for file download.

The first url presents a simple file upload html and responds back in json with the content of the uploaded file. Here is an example file for testing but you can upload any other excel file. The file upload handler uses *request.get_array* to parse the uploaded file and gets an array back. The parameter **file** is coded in the html form:

```
<input ... name=file>
```

> **Warning:** If 'field_name' was not specified, for example *request.get_array('file')* in upload_file() function, your browser would display "Bad Request: The browser (or proxy) sent a request that this server could not understand."

The rest of the links simply throw back a csv file whenever a http request is made to http://localhost:50000/download/. `make_response_from_array()` takes a list of lists and a file type as parameters and sets up the mime type of the http response. If you would like to give 'tsvz' a go, please change "csv" to "tsvz".

CHAPTER 4

Support the project

If your company has embedded pyexcel and its components into a revenue generating product, please support me on patreon to maintain the project and develop it further.

If you are an individual, you are welcome to support me too on patreon and for however long you feel like to. As a patreon, you will receive early access to pyexcel related contents.

With your financial support, I will be able to invest a little bit more time in coding, documentation and writing interesting posts.

CHAPTER 5

More excel file formats

The example application understands csv, tsv and its zipped variants: csvz and tsvz. If you would like to expand the list of supported excel file formats (see *A list of file formats supported by external plugins*) for your own application, you could install one or all of the following:

```
pip install pyexcel-xls
pip install pyexcel-xlsx
pip install pyexcel-ods
```

Warning: If you are using pyexcel <=0.2.1, you still need to import each plugin manually, e.g. *import pyexcel.ext.xls* and Your IDE or pyflakes may highlight it as un-used but it is used. The registration of the extra file format support happens when the import action is performed

CHAPTER 6

Data import and export

Continue with the previous example, the data import and export will be explained. You can copy the following code in their own appearing sequence and paste them after the place holder:

```
# insert database related code here
```

Alternatively, you can find the complete example on github

Now let's add the following imports first:

```
from flask_sqlalchemy import SQLAlchemy # sql operations
```

Now configure the database connection. Sqllite will be used and **tmp.db** will be used and can be found in your current working directory:

```
app.config['SQLALCHEMY_DATABASE_URI'] = 'sqlite:///tmp.db'
db = SQLAlchemy(app)
```

And paste some models from Flask-SQLAlchemy's documentation:

```
class Post(db.Model):
    id = db.Column(db.Integer, primary_key=True)
    title = db.Column(db.String(80))
    body = db.Column(db.Text)
    pub_date = db.Column(db.DateTime)

    category_id = db.Column(db.Integer, db.ForeignKey('category.id'))
    category = db.relationship('Category',
                               backref=db.backref('posts',
                                                  lazy='dynamic'))

    def __init__(self, title, body, category, pub_date=None):
        self.title = title
        self.body = body
        if pub_date is None:
            pub_date = datetime.utcnow()
```

```
        self.pub_date = pub_date
        self.category = category

    def __repr__(self):
        return '<Post %r>' % self.title


class Category(db.Model):
    id = db.Column(db.Integer, primary_key=True)
    name = db.Column(db.String(50))

    def __init__(self, name):
        self.name = name

    def __repr__(self):
        return '<Category %r>' % self.name
```

Now let us create the tables in the database:

```
db.create_all()
```

Write up the view functions for data import:

```
@app.route("/import", methods=['GET', 'POST'])
def doimport():
    if request.method == 'POST':

        def category_init_func(row):
            c = Category(row['name'])
            c.id = row['id']
            return c

        def post_init_func(row):
            c = Category.query.filter_by(name=row['category']).first()
            p = Post(row['title'], row['body'], c, row['pub_date'])
            return p
        request.save_book_to_database(
            field_name='file', session=db.session,
            tables=[Category, Post],
            initializers=[category_init_func, post_init_func])
        return "Saved"
    return '''
    <!doctype html>
    <title>Upload an excel file</title>
    <h1>Excel file upload (xls, xlsx, ods please)</h1>
    <form action="" method=post enctype=multipart/form-data><p>
    <input type=file name=file><input type=submit value=Upload>
    </form>
    '''
```

In the code, *category_init_func* and *post_init_func* are custom initialization functions for Category and Post respectively. In the situation where you want to skip certain rows, None should should be returned and flask_excel will ignore the row.

Write up the view function for data export:

```
@app.route("/export", methods=['GET'])
def doexport():
    return excel.make_response_from_tables(db.session, [Category, Post], "xls")
```

Then run the example again. Visit http://localhost:5000/import and upload sample-data.xls. Then visit http://localhost:5000/export to download the data back.

CHAPTER 7

Export filtered query sets

Previous example shows you how to dump one or more tables over http protocol. Hereby, let's look at how to turn a query sets into an excel sheet. You can pass a query sets and an array of selected column names to *make_response_from_query_sets()* and generate an excel sheet from it:

```
@app.route("/custom_export", methods=['GET'])
def docustomexport():
    query_sets = Category.query.filter_by(id=1).all()
    column_names = ['id', 'name']
    return excel.make_response_from_query_sets(query_sets, column_names, "xls")
```

Then visit http://localhost:5000/custom_export to download the data .. _data-types-and-its-conversion-funcs:

CHAPTER 8

All supported data types

The example application likes to have array but it is not just about arrays. Here is table of functions for all supported data types:

data structure	from file to data structures	from data structures to response
dict	*get_dict()*	*make_response_from_dict()*
records	*get_records()*	*make_response_from_records()*
a list of lists	*get_array()*	*make_response_from_array()*
dict of a list of lists	*get_book_dict()*	*make_response_from_book_dict()*
`pyexcel.Sheet`	*get_sheet()*	*make_response()*
`pyexcel.Book`	*get_book()*	*make_response()*
database table	*save_to_database()* *isave_to_database()*	*make_response_from_a_table()*
a list of database tables	*save_book_to_database()* *isave_book_to_database()*	*make_response_from_tables()*
a database query sets		*make_response_from_query_sets()*
a generator for records	`iget_records()`	
a generator of lists	`iget_array()`	

See more examples of the data structures in pyexcel documentation

CHAPTER 9

API Reference

Flask-Excel attaches **pyexcel** functions to **Request** class.

ExcelRequest

`flask_excel.ExcelRequest.`**`get_sheet`**(*field_name=None*, *sheet_name=None*, ***keywords*)

Parameters

- **`field_name`** – the file field name in the html form for file upload
- **`sheet_name`** – For an excel book, there could be multiple sheets. If it is left unspecified, the sheet at index 0 is loaded. For 'csv', 'tsv' file, *sheet_name* should be None anyway.
- **`keywords`** – additional keywords to `pyexcel.get_sheet()`

Returns A sheet object

The following html form, the *field_name* should be "file":

```
<!doctype html>
<title>Upload an excel file</title>
<h1>Excel file upload (csv, tsv, csvz, tsvz only)</h1>
<form action="" method=post enctype=multipart/form-data><p>
<input type=file name=file><input type=submit value=Upload>
</form>
```

`flask_excel.ExcelRequest.`**`get_array`**(*field_name=None*, *sheet_name=None*, ***keywords*)

Parameters

- **`field_name`** – same as *`get_sheet()`*
- **`sheet_name`** – same as *`get_sheet()`*
- **`keywords`** – additional keywords to pyexcel library

Returns a two dimensional array, a list of lists

`flask_excel.ExcelRequest.get_dict`(*field_name=None*, *sheet_name=None*, *name_columns_by_row=0*, ***keywords*)

Parameters

- **field_name** – same as `get_sheet()`
- **sheet_name** – same as `get_sheet()`
- **name_columns_by_row** – uses the first row of the sheet to be column headers by default.
- **keywords** – additional keywords to pyexcel library

Returns a dictionary of the file content

`flask_excel.ExcelRequest.get_records`(*field_name=None*, *sheet_name=None*, *name_columns_by_row=0*, ***keywords*)

Parameters

- **field_name** – same as `get_sheet()`
- **sheet_name** – same as `get_sheet()`
- **name_columns_by_row** – uses the first row of the sheet to be record field names by default.
- **keywords** – additional keywords to pyexcel library

Returns a list of dictionary of the file content

`flask_excel.ExcelRequest.get_book`(*field_name=None*, ***keywords*)

Parameters

- **field_name** – same as `get_sheet()`
- **keywords** – additional keywords to pyexcel library

Returns a two dimensional array, a list of lists

`flask_excel.ExcelRequest.get_book_dict`(*field_name=None*, ***keywords*)

Parameters

- **field_name** – same as `get_sheet()`
- **keywords** – additional keywords to pyexcel library

Returns a two dimensional array, a list of lists

`flask_excel.ExcelRequest.save_to_database`(*field_name=None*, *session=None*, *table=None*, *initializer=None*, *mapdict=None **keywords*)

Parameters

- **field_name** – same as `get_sheet()`
- **session** – a SQLAlchemy session
- **table** – a database table
- **initializer** – a custom table initialization function if you have one
- **mapdict** – the explicit table column names if your excel data do not have the exact column names
- **keywords** – additional keywords to `pyexcel.Sheet.save_to_database()`

`flask_excel.ExcelRequest.`**`isave_to_database`**(*field_name=None*, *session=None*, *table=None*, *initializer=None*, *mapdict=None* ***keywords*)

similar to `:meth:`~flask_excel.ExcelRequest.isave_to_database()`. But it requires less memory.

This requires column names must be at the first row.

`flask_excel.ExcelRequest.`**`save_book_to_database`**(*field_name=None*, *session=None*, *tables=None*, *initializers=None*, *mapdicts=None*, ***keywords*)

Parameters

- **`field_name`** – same as *`get_sheet()`*
- **`session`** – a SQLAlchemy session
- **`tables`** – a list of database tables
- **`initializers`** – a list of model initialization functions.
- **`mapdicts`** – a list of explicit table column names if your excel data sheets do not have the exact column names
- **`keywords`** – additional keywords to `pyexcel.Book.save_to_database()`

`flask_excel.ExcelRequest.`**`isave_book_to_database`**(*field_name=None*, *session=None*, *tables=None*, *initializers=None*, *mapdicts=None*, ***keywords*)

similar to `:meth:`~flask_excel.ExcelRequest.isave_book_to_database()`. But it requires less memory.

This requires column names must be at the first row in each sheets

Response methods

`flask_excel.`**`make_response`**(*pyexcel_instance*, *file_type*, *status=200*, *file_name=None*)

Parameters

- **`pyexcel_instance`** – `pyexcel.Sheet` or `pyexcel.Book`
- **`file_type`** – one of the following strings:
  - 'csv'
  - 'tsv'
  - 'csvz'
  - 'tsvz'
  - 'xls'
  - 'xlsx'
  - 'xlsm'
  - 'ods'
- **`status`** – unless a different status is to be returned.
- **`file_name`** – provide a custom file name for the response, excluding the file extension

`flask_excel.`**`make_response_from_array`**(*array*, *file_type*, *status=200*, *file_name=None*)

Parameters

- **array** – a list of lists
- **file_type** – same as `make_response()`
- **status** – same as `make_response()`
- **file_name** – same as `make_response()`

`flask_excel.`**`make_response_from_dict`**(*dict*, *file_type*, *status=200*, *file_name=None*)

Parameters

- **dict** – a dictionary of lists
- **file_type** – same as `make_response()`
- **status** – same as `make_response()`
- **file_name** – same as `make_response()`

`flask_excel.`**`make_response_from_records`**(*records*, *file_type*, *status=200*, *file_name=None*)

Parameters

- **records** – a list of dictionaries
- **file_type** – same as `make_response()`
- **status** – same as `make_response()`
- **file_name** – same as `make_response()`

`flask_excel.`**`make_response_from_book_dict`**(*book_dict*, *file_type*, *status=200*, *file_name=None*)

Parameters

- **book_dict** – a dictionary of two dimensional arrays
- **file_type** – same as `make_response()`
- **status** – same as `make_response()`
- **file_name** – same as `make_response()`

`flask_excel.`**`make_response_from_a_table`**(*session*, *table*, *file_type* *status=200*, *file_name=None*)

Produce a single sheet Excel book of *file_type*

Parameters

- **session** – SQLAlchemy session
- **table** – a SQLAlchemy table
- **file_type** – same as `make_response()`
- **status** – same as `make_response()`
- **file_name** – same as `make_response()`

`flask_excel.`**`make_response_from_query_sets`**(*query_sets*, *column_names*, *file_type* *status=200*, *file_name=None*)

Produce a single sheet Excel book of *file_type* from your custom database queries

Parameters

- **query_sets** – a query set
- **column_names** – a nominated column names. It could not be None, otherwise no data is returned.
- **file_type** – same as `make_response()`
- **status** – same as `make_response()`
- **file_name** – same as `make_response()`

`flask_excel.`**`make_response_from_tables`**(*session*, *tables*, *file_type* *status=200*, *file_name=None*)

Produce a multiple sheet Excel book of *file_type*. It becomes the same as `make_response_from_a_table()` if you pass *tables* with an array that has a single table

Parameters

- **session** – SQLAlchemy session
- **tables** – SQLAlchemy tables
- **file_type** – same as `make_response()`
- **status** – same as `make_response()`
- **file_name** – same as `make_response()`

CHAPTER 10

Change log

0.0.7 - 20.07.2017

Updated

1. the intialization method has been modified. please call init_excel(app) before you do anything else. This change was made in order to apply for approved flask extension status. And by doing this change, it will support multiple Flask apps and only the app that was initialized with init_excel gets Flask-Excel and other apps in your BIG app won't get affected.

0.0.6 - 22.06.2017

Updated

1. **#22: support download** file name in unicode(including Chinese texts)

0.0.5 - 21.08.2016

Updated

1. compatibility with pyexcel v0.2.2: automatic discovery of pyexcel plugins.
2. #15: file name may have more than one dot

0.0.4 - 15.01.2016

Updated

1. #8: set file name in response

0.0.3 - 01.07.2015

Updated

1. code refactoring. less code lines in Flask-Excel and more reusable code in pyexcel-webio

0.0.2 - 21.05.2015

Added

1. turn query sets into a response

0.0.1 - 22.01.2015

Mix pyexcel into Flask.request and bring more make_response functions.

Python Module Index

f

Index