flask-excel Documentation

Release 0.0.3

C.W.

July 11, 2015

Contents

Author C.W.

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

Version 0.0.3

Generated July 11, 2015

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 highlighted features are:

1. excel data import into and export from databases
2. turn uploaded excel file directly into Python data struture
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

Plugins	Supported file formats
xls	xls, xlsx(r), xlsm(r)
xlsx	xlsx
ods3	ods (python 2.6, 2.7, 3.3, 3.4)
ods	ods (python 2.6, 2.7)
text	write only)json, rst, mediawiki, latex, grid, pipe, orgtbl, plain simple

This library makes infomation processing involving various excel files as easy as processing array and dictionary. The information processing job includes 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.

CHAPTER 1

Installation

You can install it via pip:

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

or clone it and install it:

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

Installation of individual plugins , please refer to individual plugin page.

CHAPTER 2

Setup

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

```
from flask.ext import excel
```

or:

```
import flask.ext.excel
```

CHAPTER 3

Quick start

A minimal application may look like this:

```
from flask import Flask, request, jsonify
from flask.ext import 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")

# insert database related code here

if __name__ == "__main__":
    app.run()
```

The tiny application exposes two urls: one for file upload and the other for file donload. The former url presents a simple file upload html and responds back in json with the content of the uploaded file. Here is an *example file <https://github.com/chfw/Flask-Excel/blob/master/examples/example_for_upload.csv>* 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 latter simply throws back a csv file whenever a http request is made to http://localhost:50000/download/. *excel.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

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 include one or all of the following import lines right after **Flask-Excel** is imported:

```
import pyexcel.ext.xls
import pyexcel.ext.xlsx
import pyexcel.ext.ods
```

CHAPTER 5

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
```

Alernatively, you can find the complete example on github

Now let's add the following imports first:

```
from flask.ext.sqlalchemy import SQLAlchemy # sql operations
import pyexcel.ext.xls # import it to be able to handle xls file format
```

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>
    '''
```

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 6

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

CHAPTER 7

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()*	*make_response_from_a_table()*
a list of database tables	*save_book_to_database()*	*make_response_from_tables()*
a database query sets		*make_response_from_query_sets()*

See more examples of the data structures in pyexcel documentation

CHAPTER 8

API Reference

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

class `flask_excel.`**`ExcelRequest`**(*environ*, *populate_request=True*, *shallow=False*)

`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>
```

`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

`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

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

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

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

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()`

save_book_to_database (*field_name=None*, *session=None*, *tables=None*, *initializers=None*, *mapdicts=None*, ***keywords*)

Parameters

- **field_name** – save as *get_sheet()*
- **session** – a SQLAlchemy sessio
- **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()`

CHAPTER 9

Response methods

flask_excel.**make_response**(*pyexcel_instance*, *file_type*, *status=200*)

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.

flask_excel.**make_response_from_array**(*array*, *file_type*, *status=200*)

Parameters

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

flask_excel.**make_response_from_dict**(*dict*, *file_type*, *status=200*)

Parameters

- **dict** – a dictinary of lists
- **file_type** – same as *make_response()*
- **status** – same as *make_response()*

flask_excel.**make_response_from_records**(*records*, *file_type*, *status=200*)

Parameters

- **records** – a list of dictionaries

- **file_type** – same as *make_response()*
- **status** – same as *make_response()*

flask_excel.**make_response_from_book_dict**(*book_dict*, *file_type*, *status=200*)

Parameters

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

flask_excel.**make_response_from_a_table**(*session*, *table*, *file_type status=200*)

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()*

flask_excel.**make_response_from_query_sets**(*query_sets*, *column_names*, *file_type status=200*)

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()*

flask_excel.**make_response_from_tables**(*session*, *tables*, *file_type status=200*)

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()*

CHAPTER 10

Indices and tables

- genindex
- modindex
- search

f

E

F

G

M

S