pyramid-excel Documentation

Release 0.0.5

Onni Software Ltd.

Oct 16, 2020

Contents

1 Plugin shopping guide **3**

2 Installation **5**

3 Setup **7**

4 Quick Start **9**

4.1 Support the project 10

4.2 More excel file formats 10

4.3 Data import and export 12

4.4 Export filtered query sets 14

4.5 All supported data types 14

4.6 API Reference 15

Python Module Index **19**

Index **21**

Author chfw

Source code http://github.com/pyexcel-webwares/pyramid-excel.git

Issues http://github.com/pyexcel-webwares/pyramid-excel/issues

License New BSD License

Released 0.0.5

Generated Oct 16, 2020

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."
```

pyramid-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 Pyramid based web development, instead of file formats.

The idea originated from the common usability problem: when an excel file driven web application is delivered for non-developer users (ie: team assistant, human resource administrator etc). The fact is that not everyone knows (or cares) about the differences between various excel formats: csv, xls, xlsx are all the same to them. Instead of training those users 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 type to you application, all you need is to install an extra pyexcel plugin. Hence no code changes to your application and no issues with excel file formats any more. 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
pyexcel-io	csv, csvz[1], tsv, tsvz[2]	
pyexcel-xls	xls, xlsx(read only), xlsm(read only)	xlrd, xlwt
pyexcel-xlsx	xlsx	openpyxl
pyexcel-ods3	ods	pyexcel-ezodf, lxml
pyexcel-ods	ods	odfpy

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

Table 2: Dedicated file reader and writers

Package name	Supported file formats	Dependencies
pyexcel-xlsxw	xlsx(write only)	XlsxWriter
pyexcel-libxlsxw	xlsx(write only)	libxlsxwriter
pyexcel-xlsxr	xlsx(read only)	lxml
pyexcel-xlsbr	xlsb(read only)	pyxlsb
pyexcel-odsr	read only for ods, fods	lxml
pyexcel-odsw	write only for ods	loxun
pyexcel-htmlr	html(read only)	lxml,html5lib
pyexcel-pdfr	pdf(read only)	camelot

CHAPTER 1

Plugin shopping guide

Since 2020, all pyexcel-io plugins have dropped the support for python version lower than 3.6. If you want to use any python verions, please use pyexcel-io and its plugins version lower than 0.6.0.

Except csv files, xls, xlsx and ods files are a zip of a folder containing a lot of xml files

The dedicated readers for excel files can stream read

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').

Table 1: Other data renderers

Package name	Supported file formats	Dependencies	Python versions
pyexcel-text	write only:rst, mediawiki, html, latex, grid, pipe, orgtbl, plain simple read only: ndjson r/w: json	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

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.

CHAPTER 2

Installation

You can install pyramid-excel via pip:

```
$ pip install pyramid-excel
```

or clone it and install it:

```
$ git clone https://github.com/pyexcel-webwares/pyramid-excel.git
$ cd pyramid-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 3

Setup

Once the pyramid_excel is installed, you must use the config.include mechanism to include it into your Pyramid project's configuration:

```
config = Configurator(.....)
config.include('pyramid_excel')
```

Alternately, you may activate the extension by changing your application's .ini file by adding it to the pyramid.includes list:

```
pyramid.includes = pyramid_excel
```

CHAPTER 4

Quick Start

Here is the quick demonstration code for pyramid-excel:

```
from wsgiref.simple_server import make_server
from pyramid.config import Configurator
from pyramid.response import Response
from pyramid.view import view_config
import pyramid_excel as excel


upload_form = """
<!doctype html>
<title>Upload an excel file</title>
<h1>Excel file upload</h1>
<form action="" method=post enctype=multipart/form-data><p>
<input type=file name=file><input type=submit value=Upload>
</form>
"""


@view_config(route_name='upload')
def upload_view(request):
    if request.method == 'POST':
        data = request.get_array(field_name='file')
        return excel.make_response_from_array(data, 'xls', file_name="response")
    return Response(upload_form)


if __name__ == '__main__':
    config = Configurator()
    config.include('pyramid_excel')
    config.add_route('upload', '/upload')
    config.scan()
    app = config.make_wsgi_app()
    server = make_server('0.0.0.0', 5000, app)
```

(continues on next page)

(continued from previous page)

```
    print("Listening on 0.0.0.0:5000")
    server.serve_forever()
```

Before you start the server, let's install a plugin to support xls file format:

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

And you can start the tiny server by this command, assuming you have save it as tiny_server.py:

```
$ python tiny_server.py
Listening on 0.0.0.0:5000
```

Note: Alternatively, you can check out the code from github

```
git clone https://github.com/pyexcel/pyramid-excel.git
```

The test application for pyramid-excel is a fully fledged site according to the tutorial here.

Once you have the code, please change to pyramid-excel directory and then install all dependencies:

```
$ cd pyramid-excel
$ pip install -r requirements.txt
$ pip install -r test_requirements.txt
```

Then run the test application:

```
$ pserve development.ini
Starting server in PID 9852.
serving on http://127.0.0.1:5000
```

4.1 Support the project

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

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

And your issues will get prioritized if you would like to become my patreon as *pyexcel pro user*.

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

4.2 More excel file formats

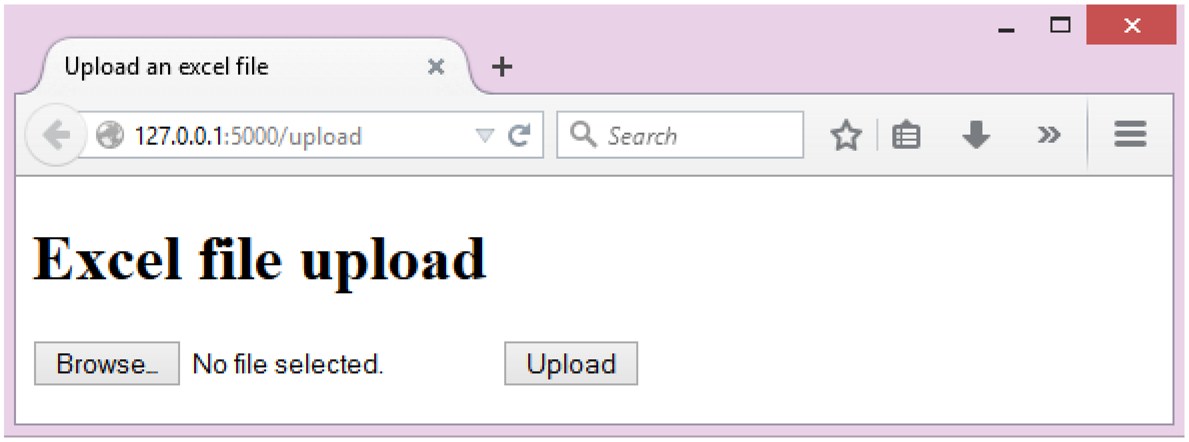
The example application understands csv, tsv and its zipped variants: csvz and tsvz. If you would like to expand the list of supported excel file formats (see *A list of file formats supported by external plugins*) for your own application, you could install one or all of the following:

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

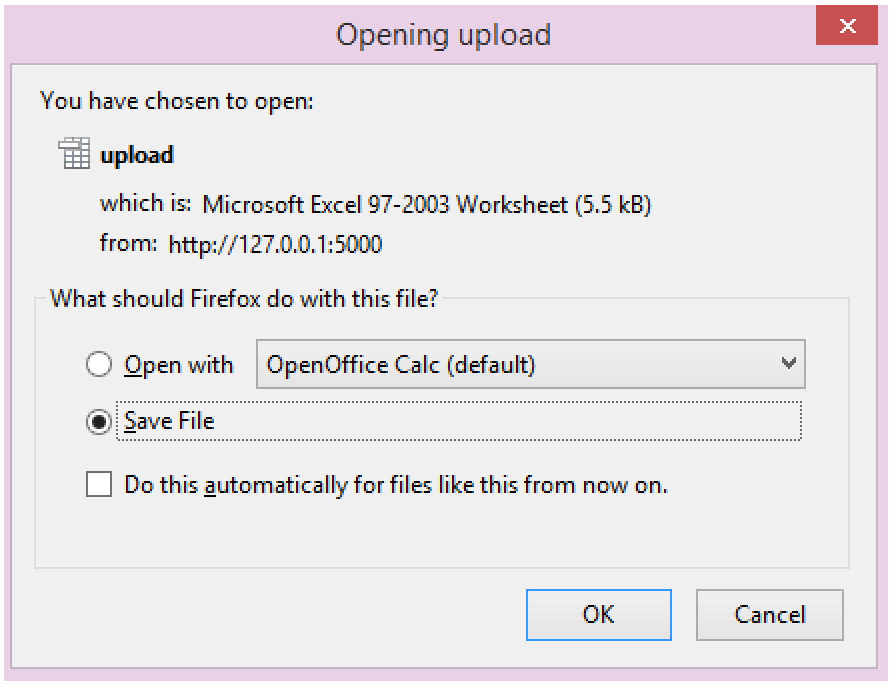
Warning: If you are using pyexcel <=0.2.1, you still need to import each plugin manually, e.g. *import pyexcel.ext.xls* and Your IDE or pyflakes may highlight it as un-used but it is used. The registration of the extra file format support happens when the import action is performed

4.2.1 Handle excel file upload and download

This example shows how to process uploaded excel file and how to make data download as an excel file. Open your browser and visit http://localhost:5000/upload, you shall see this upload form:



please upload an xls file and you would get this dialog:


Opening upload

You have chosen to open:

upload

which is: Microsoft Excel 97-2003 Worksheet (5.5 kB)

from: http://127.0.0.1:5000

What should Firefox do with this file?

[ ] Open with: OpenOffice Calc (default)

[✓ Save File]

[ ] Do this automatically for files like this from now on.

Please focus on the following code section:

```
@view_config(route_name='upload')
def upload_view(request):
    if request.method == 'POST':
        data = request.get_array(field_name='file')
        return excel.make_response_from_array(data, 'xls')
    return Response(upload_form)
```

By default, the GET request will be served with upload_form. Once an excel file is uploaded, this library kicks in and help you get the data as an array. Then you can make an excel file as download by using make_response_from_array.

4.3 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 sqlalchemy import (
    Column,
    Index,
    Integer,
    Text,
    String,
    ForeignKey,
    DateTime,
    create_engine
    )

from sqlalchemy.ext.declarative import declarative_base
from sqlalchemy.orm import relationship, backref
from sqlalchemy.orm import (
    scoped_session,
    sessionmaker,
    )

from zope.sqlalchemy import ZopeTransactionExtension

DBSession = scoped_session(sessionmaker(extension=ZopeTransactionExtension()))
Base = declarative_base()
```

And paste some models:

```
class Post(Base):
    __tablename__ = 'post'
    id = Column(Integer, primary_key=True)
    title = Column(String(80))
    body = Column(Text)
    pub_date = Column(DateTime)

    category_id = Column(Integer, ForeignKey('category.id'))
    category = relationship('Category',
```

(continues on next page)

(continued from previous page)

```
        backref=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(Base):
    __tablename__ = 'category'
    id = Column(Integer, primary_key=True)
    name = Column(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:

```
def init_db():
    engine = create_engine('sqlite:///tmp.db')
    DBSession.configure(bind=engine)
    Base.metadata.drop_all(engine)
    Base.metadata.create_all(engine)
```

And make sure we call init_db in main:

```
if __name__ == '__main__':
    config = Configurator()
    config.include('pyramid_excel')
    config.add_route('upload', '/upload')
    config.add_route('import', '/import')
    config.add_route('export', '/export')
    config.scan()
    init_db()  # <-------
    app = config.make_wsgi_app()
    server = make_server('0.0.0.0', 5000, app)
    print("Listening on 0.0.0.0:5000")
    server.serve_forever()
```

Write up the view functions for data import:

```
@view_config(route_name="import")
def doimport(request):
    if request.method == 'POST':
        def category_init_func(row):
            c = Category(row['name'])
            c.id = row['id']
            return c
```

(continues on next page)

(continued from previous page)

```
        def post_init_func(row):
            c = DBSession.query(Category).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=DBSession,
                                      tables=[Category, Post],
                                      initializers=[category_init_func, post_init_
↪func])
        return Response("Saved")
    return Response(upload_form)
```

Write up the view function for data export:

```
@view_config(route_name="export")
def doexport(request):
    return excel.make_response_from_tables(DBSession, [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.

4.4 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:

```
@view_config(route_name="custom_export")
def docustomexport(request):
    query_sets = DBSession.query(Category).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:

4.5 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

4.6 API Reference

pyramid-excel attaches **pyexcel** functions to pyramid's **Request** class.

`pyramid_excel.ExcelRequestFactory.`**`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

`pyramid_excel.ExcelRequestFactory.`**`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

`pyramid_excel.ExcelRequestFactory.`**`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

`pyramid_excel.ExcelRequestFactory.`**`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

`pyramid_excel.ExcelRequestFactory.`**`get_book`**(*field_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

`pyramid_excel.ExcelRequestFactory.`**`get_book_dict`** (*field_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

`pyramid_excel.ExcelRequestFactory.`**`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()`

`pyramid_excel.ExcelRequestFactory.`**`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()`

`pyramid_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

`pyramid_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()*

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

Parameters

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

`pyramid_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()*

`pyramid_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()*

`pyramid_excel.`**`make_response_from_a_table`**(*model*, *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()`*

`pyramid_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()`*

`pyramid_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()`*

Python Module Index

p

pyramid_excel, 16

pyramid_excel.ExcelRequestFactory, 15

Index

G

get_array() (*in module pyramid_excel.ExcelRequestFactory*), 15

get_book() (*in module pyramid_excel.ExcelRequestFactory*), 15

get_book_dict() (*in module pyramid_excel.ExcelRequestFactory*), 16

get_dict() (*in module pyramid_excel.ExcelRequestFactory*), 15

get_records() (*in module pyramid_excel.ExcelRequestFactory*), 15

get_sheet() (*in module pyramid_excel.ExcelRequestFactory*), 15

M

make_response() (*in module pyramid_excel*), 16

make_response_from_a_table() (*in module pyramid_excel*), 17

make_response_from_array() (*in module pyramid_excel*), 17

make_response_from_book_dict() (*in module pyramid_excel*), 17

make_response_from_dict() (*in module pyramid_excel*), 17

make_response_from_query_sets() (*in module pyramid_excel*), 18

make_response_from_records() (*in module pyramid_excel*), 17

make_response_from_tables() (*in module pyramid_excel*), 18

P

pyramid_excel (*module*), 16

pyramid_excel.ExcelRequestFactory (*module*), 15

S

save_book_to_database() (*in module pyramid_excel.ExcelRequestFactory*), 16

save_to_database() (*in module pyramid_excel.ExcelRequestFactory*), 16