Django File Picker Documentation

Release 0.5

Caktus Consulting Group LLC

May 20, 2017

Contents

django-file-picker is a pluggable Django application used for uploading, browsing, and inserting various forms of media into HTML form fields.

Using jQuery Tools, file_picker integrates seamlessly into pre-existing pages by installing an overlay that lists file details and, when applicable, image thumbnails. New files can also be uploaded from within the overlay (via AJAX Upload).

`file_picker` provides a few optional extensions to help get started, including `file_picker.uploads`, an app with pre-built Image and File models, and `file_picker.wymeditor`, an app that integrates with WYMeditor, a web-based WYSIWYM (What You See Is What You Mean) XHTML editor. These extensions are provided for convenience and can easily be replaced by custom modules.

For complete documentation checkout, http://django-file-picker.readthedocs.org

CHAPTER 1

Dependencies

Required

- Python 2.6 or 2.7 (**note**: Python 3 is not yet supported)
- Django 1.6 to 1.9 (inclusive)
- sorl-thumbnail==12.3
- jQuery 1.4.x
- jQuery Tools 1.2.x
- AJAX Upload (included)

Optional

- django-staticfiles
- WYMeditor 0.5

  If you are using *django-staticfiles* (or `django.contrib.staticfiles` in Django 1.3) then add `file_picker` to your INSTALLED_APPS to include the related css/js.

  Otherwise make sure to include the contents of the static folder in your projects media folder.

CHAPTER 2

Basic Installation

1. Add `file_picker` to INSTALLED_APPS in settings.py:

```
INSTALLED_APPS = (
    'file_picker',
    'file_picker.uploads', # file and image Django app
    'file_picker.wymeditor', # optional WYMeditor plugin
)
```

   `file_picker.uploads` will automatically create two pickers name 'images' and 'files'.

2. Add the `file_picker` URLs to urls.py, e.g.:

```
import file_picker
file_picker.autodiscover()

urlpatterns = patterns('',
    # ...
    (r'^file-picker/', include(file_picker.site.urls)),
    # ...
)
```

Development sponsored by Caktus Consulting Group, LLC..

CHAPTER 3

Contents

Basic Setup

1. Use or create a model for storing images and/or files. For simplicity here we will use the models in `file_picker.uploads`, Image and File.
2. Use or create another model to contian the text field(s) to be inserted in by the picker. Here we will use the Post model from the `sample_project.article`. Which has two text fields, Body and Teaser.
3. To use the pickers on both the teaser and body fields use a *formfield_override* to override the widget with the `file_picker.widgets.SimpleFilePickerWidget`:

```
import file_picker
from django.contrib import admin
from sample_project.article import models as article_models

class PostAdmin(admin.ModelAdmin):
    formfield_overrides = {
        models.TextField: {
            'widget': file_picker.widgets.SimpleFilePickerWidget(pickers={
                'image': "images", # a picker named "images" from file_picker.
↪uploads
                'file': "files", # a picker named "files" from file_picker.uploads
            }),
        },
    }

    class Media:
        js = ("http://cdn.jquerytools.org/1.2.5/full/jquery.tools.min.js",)

admin.site.register(article_models.Post, PostAdmin)
```

Simple File Picker Widget

class `file_picker.widgets.`**`SimpleFilePickerWidget`**

To use the simple file picker widget override the desired form field's widget. It takes in a dictionary with expected keys *"image"* and/or *"file"* these define which link to use "Add Image" and/or "Add File".

For an example of usage look at the.

The Uploads App

Designed to make it easy to get File Picker up and running without having to add models or register them with `file_picker`. The uploads app includes two simple pickers which can be attached to your own projects text fields. For install instructions check out *Basic Setup*

FilePicker

class `file_picker.uploads.file_pickers.`**`FilePicker`**

Is a simple class based off of the `file_picker.FilePickerBase` which is connected to the *File* model and can be found in the Uploads admin section.

ImagePicker

class `file_picker.uploads.file_pickers.`**`ImagePicker`**

Is a simple class based off of the `file_picker.ImagePickerBase` which is connected to the *Image* model and can be found in the Uploads admin section.

Simple File Picker Widget

These pickers can be used like any other. Below is an example of how to add them on a single text field:

```
body = forms.TextField(
            widget=file_picker.widgets.SimpleFilePickerWidget(pickers={
                'file': "files",
                'image': "images",
            }))
```

Where the *"file"* and *"image"* keywords add classes to the inputs, so that the links for the overlay can be added. They can also be added to all fields in a form by using the *formfield_overrides* as in:ref:*setup*.

The WYMeditor App

Included to make the intergration of File Picker with a popular WYSIWYG easy. WYMeditor is a javascript based editor, its documentation can be found here. This application offers an extra form widget for applying WYMeditor to a text field with buttons for files and images if desired.

WYMeditorWidget

class `file_picker.wymeditor.widgets.`**`WYMeditorWidget`**

To use the WYMeditorWidget override the desired form field's widget. It takes in a dictionary with expected keys *"image"* and/or *"file"* these define which button is used to call the overlay, either an image or a paper clip icon respectively.

Example TextField Override

An example using a *formfield_override* in an admin class use WYMeditor and File Picker for each *TextField* in the form.

```
class PostAdmin(admin.ModelAdmin):
        formfield_overrides = {
            models.TextField: {
                'widget': file_picker.wymeditor.widgets.WYMeditorWidget(
                    pickers={
                        'image': "images",
                        'file': "files",
                    }
                ),
            },
        }

        class Media:
            js = ("http://cdn.jquerytools.org/1.2.5/full/jquery.tools.min.js",)
```

Creating Custom Pickers

File Picker offers many ways to create custom pickers and assign them to your own models.

The FilePickerBase Class

The base file picker class has a mixture of class based views and helper functions for building the colorbox on the page. File pickers should be included in the *file_pickers.py* file in the root directory of any app so that the auto-discovery process can find it.

Attributes

Each picker can take a set of attributes for easy customization.:

```
from myapp.models import CustomModel
from myapp.forms import CustomForm
import file_pickers

class CustomPicker(file_picker.FilePickerBase):
    form = CustomForm
    page_size = 4
    link_headers = ['Insert File',]
    columns = ['name', 'body', 'description',]
    extra_headers = ['Name', 'Body', 'Description',]
```

```
    ordering = '-date'

file_picker.site.register(CustomModel, CustomPicker, name='custom')
```

None of these attributes are required and they all have sane defaults.

- *form*- If left blank is created by building a *ModelForm* from the model defined in the register function. It is used to build the form on the Upload tab.
- *link_headers*- Defines the headers for the first set of columns which are used to insert content into the textbox or WYSIWYG of your choice. By default it converts _ to ' ' and capitalizes first letter of the fields name.
- *columns*- Defines the fields you want to be included on the listing page and their ordering.
- *extra_headers*- The list is used to define the headers for the columns and needs to be in the same order as columns.
- *ordering*- Defines the order of items on the listing page in to be used as `query_set.order_by('-date')`.

Methods

The three main methods consist of *append*, *list*, and *upload_file*. List and upload_file take in the request object and act as views while append takes in an model item and helps build the JSON output for list. Other methods are available but typically do not need to be modified.

append(obj)

This method takes in *obj* which is a item from the model and outputs a dictionary to be used by list. This dictoinary is of the form.:

```
{
    'name': 'Name for the object.',
    'url': 'The url to the file.',
    'extra': {
        'column_name_1': 'value',
        'column_name_2': 'value',
    },
    'insert': ['string/html to insert if first link is clicked', 'for second link',],
    'link_content': ['string to show on first insert link', 'for second link',],
}
```

As a note the *link_content* list and insert list must be the same length, as well as the extra dictionary and the *link_headers* attribute.

list(request)

This takes in a *request* object and outputs.:

```
{
    'page': Integer of current,
    'pages': List of pages,
    'search': The current search term,
    'result': List returned from ,
    'has_next': Lets the paginator know whether to hide/show the next button.,
    'has_previous': Same as above for previous button.,
```

```
    'link_headers': List of link headers either generated or defined by the attribute.
↪,
    'extra_headers': List of the extra headers made in the same we as above.,
    'columns': List of column names to be included(same as the columns attribute.),
}
```

upload_file(request)

Builds the upload file form and is used to upload files in two steps, file first then the other form parameters.

If called without a POST it returns a JSON dictionary with the key form with an html block for the form.

If called with a file and then with the post, its a two step process. If the form passed on the second step it returns the result of append for the object which was just created.

Screenshots

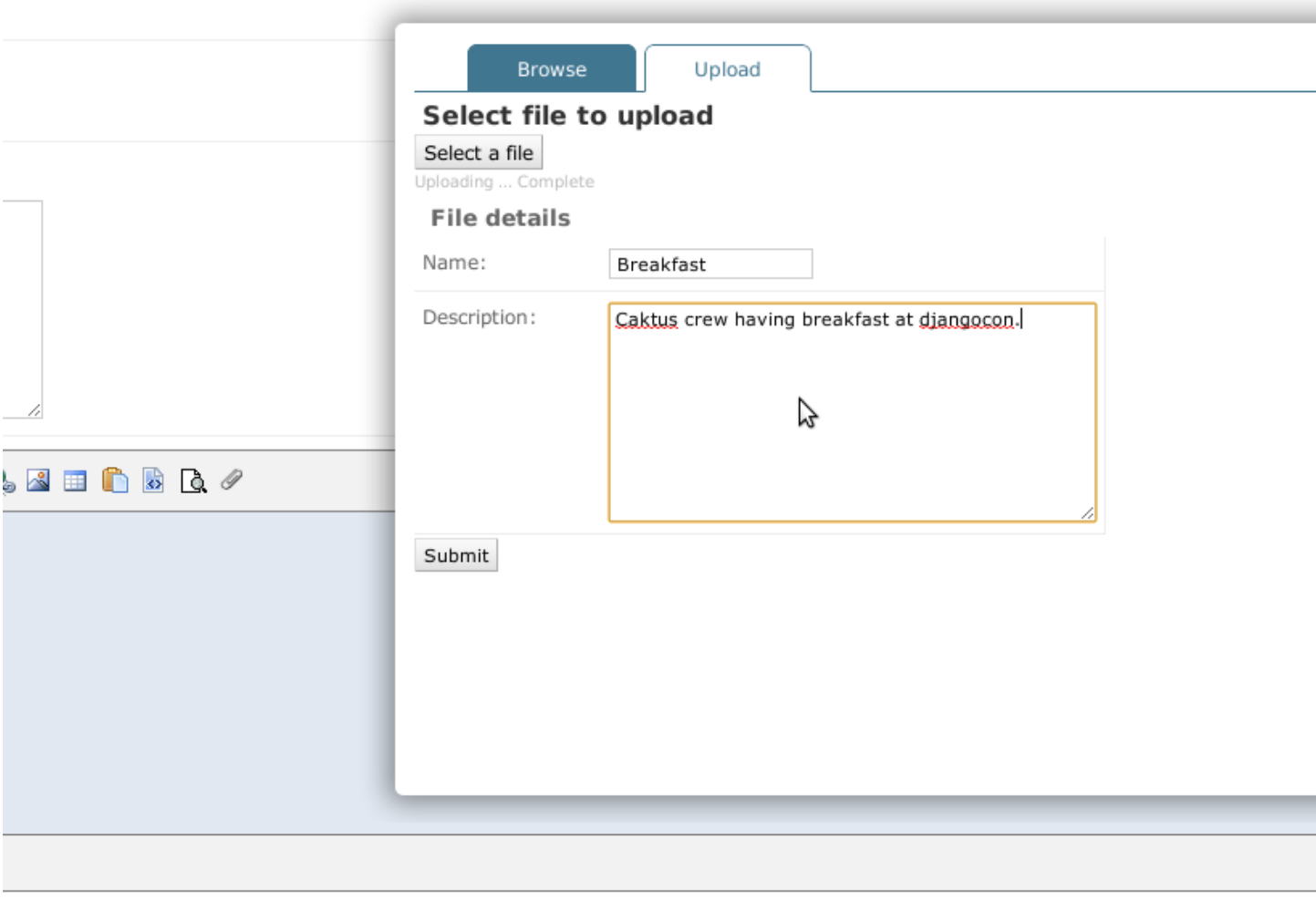

Fig. 3.1: The upload pane from the sample project.

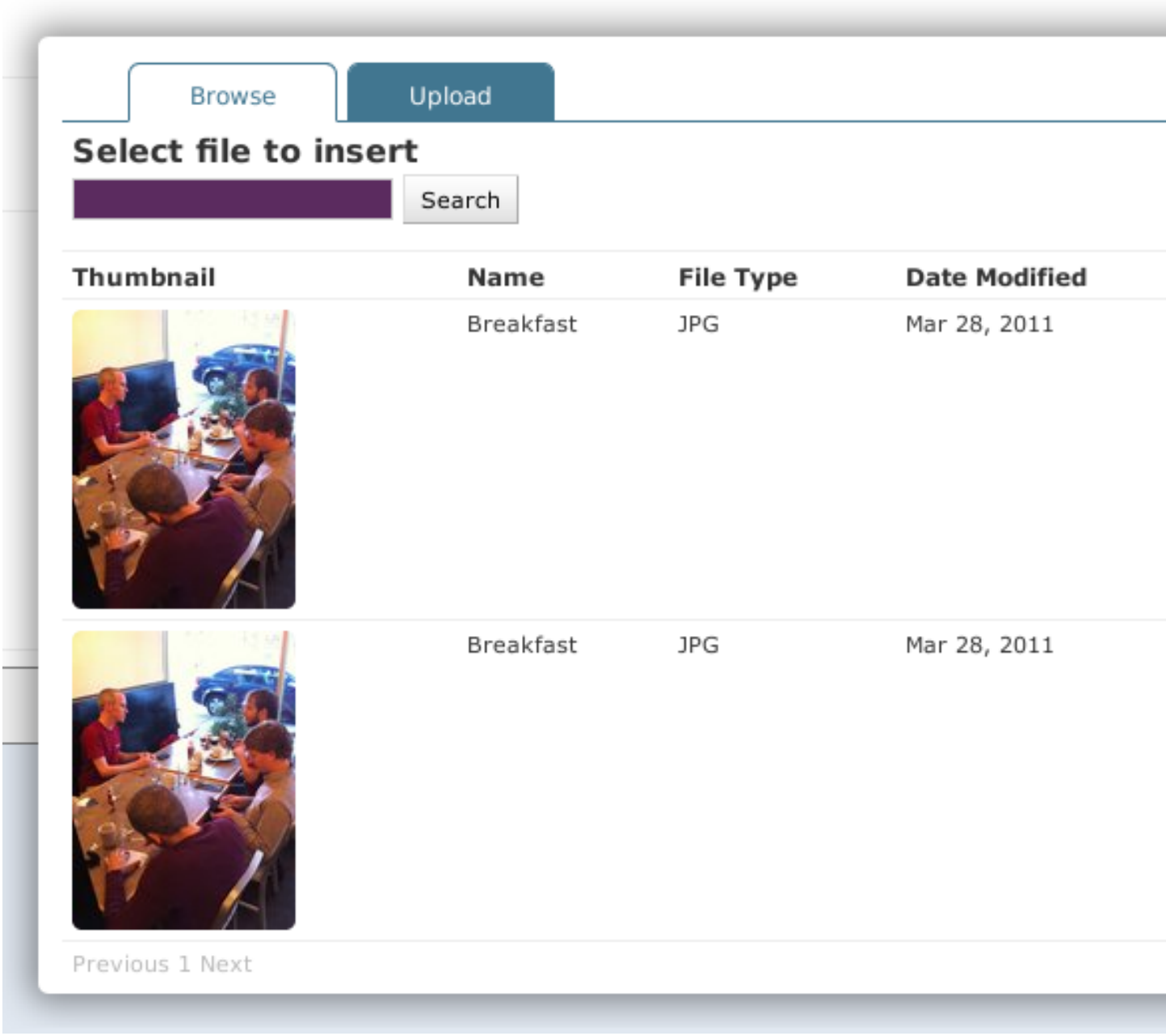

Fig. 3.2: The browser pane from the sample project.

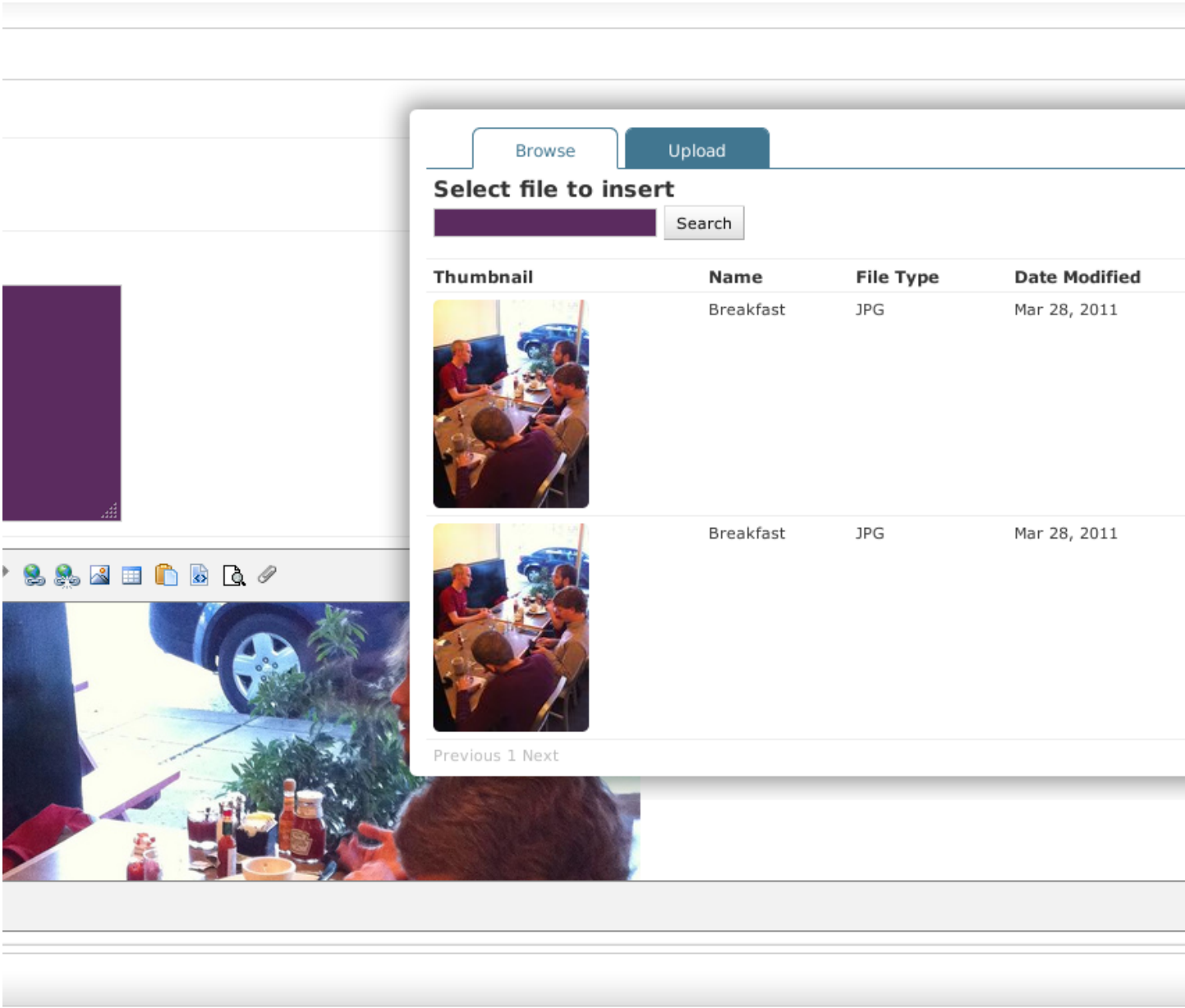

Fig. 3.3: An example of inseting an image with File Picker.

Motivation

The deep concern while building file picker has been flexibility. Too many projects focus on wrapping everything together so that they can make deep connections.

Our main goal has been to build a application that facilitates connections, that can be attached accross multiple applications and models.

CHAPTER 4

Indices and tables

- genindex
- modindex
- search

Index

F