Django FileBrowser Documentation

Release 3.5.2

Patrick Kranzlmueller

December 15, 2013

Contents

Media-Management with Grappelli.

This documentation covers version 3.5.2 of the FileBrowser.

Note: FileBrowser 3.5.2 requires Django 1.4/1.5 and Grappelli 2.4. FileBrowser is always developed against the lastest stable Django release and is NOT tested with Djangos trunk.

Installation and Setup

CHAPTER 1

Quick start guide

For using the FileBrowser, Django needs to be installed and an Admin Site has to be activated.

1.1 Requirements

- Django 1.4/1.5, http://www.djangoproject.com
- Grappelli 2.4, https://github.com/sehmaschine/django-grappelli
- PIL, http://www.pythonware.com/products/pil/

1.2 Download

Using `pip`:

```
pip install django-filebrowser
```

Go to https://github.com/sehmaschine/django-filebrowser if you need to download a package or clone the repo.

1.3 Installation

Changed in version 3.4.0. Open `settings.py` and add `filebrowser` to your `INSTALLED_APPS` (before `django.contrib.admin`):

```
INSTALLED_APPS = (
    'grappelli',
    'filebrowser',
    'django.contrib.admin',
)
```

In your `url.py` import the default FileBrowser site:

```
from filebrowser.sites import site
```

and add the following URL-patterns (before any admin-urls):

```
urlpatterns = patterns('',
   url(r'^admin/filebrowser/', include(site.urls)),
)
```

Collect the media files:

```
python manage.py collectstatic
```

Note: Please refer to the Staticfiles Documentation for setting up and using `staticfiles`.

1.4 Settings

Check the *Settings*. You need to add a folder "uploads" within your `MEDIA_ROOT` when using the default settings.

1.5 Testing

Run the FileBrowser tests:

```
python manage.py test filebrowser
```

Start the devserver and login to your admin site:

```
python manage.py runserver <IP-address>:8000
```

Goto `/admin/filebrowser/browse/` and check if everything looks/works as expected. If you're having problems, see *Troubleshooting*.

CHAPTER 2

Settings

There are quite a few possibilities of customizing the FileBrowser to fit your needs. Nonetheless, you should be able to start with the default settings.

Note: All settings can be defined in your projects settings-file or the FileBrowsers settings-file (`settings.py`). When using the projects settings-file, you have to use the prefix `FILEBROWSER_` for every setting (e.g. `FILEBROWSER_MEDIA_URL` instead of `MEDIA_URL`).

2.1 Main URL/Paths Settings

2.1.1 MEDIA_ROOT

The absolute path to the directory that holds the media-files you want to browse:

```
MEDIA_ROOT = getattr(settings, "FILEBROWSER_MEDIA_ROOT", settings.MEDIA_ROOT)
```

2.1.2 MEDIA_URL

URL that handles the media served from `MEDIA_ROOT`:

```
MEDIA_URL = getattr(settings, "FILEBROWSER_MEDIA_URL", settings.MEDIA_URL)
```

2.1.3 DIRECTORY (relative to `MEDIA_ROOT`)

Main FileBrowser Directory. Leave empty in order to browse all files and folders within MEDIA_ROOT:

```
DIRECTORY = getattr(settings, "FILEBROWSER_DIRECTORY", 'uploads/')
```

2.2 FileBrowser Media, TinyMCE Media

2.2.1 URL_FILEBROWSER_MEDIA, PATH_FILEBROWSER_MEDIA

The URL and Path to your FileBrowser media-files:

```
URL_FILEBROWSER_MEDIA = getattr(settings, "FILEBROWSER_URL_FILEBROWSER_MEDIA", settings.STATIC_URL +
PATH_FILEBROWSER_MEDIA = getattr(settings, "FILEBROWSER_PATH_FILEBROWSER_MEDIA", os.path.join(setting
```

2.2.2 URL_TINYMCE, PATH_TINYMCE

The URL to your TinyMCE Installation:

```
URL_TINYMCE = getattr(settings, "FILEBROWSER_URL_TINYMCE", settings.ADMIN_MEDIA_PREFIX + "tinymce/js
PATH_TINYMCE = getattr(settings, "FILEBROWSER_PATH_TINYMCE", settings.ADMIN_MEDIA_PREFIX + "tinymce/
```

Note: Only change these settings if you're absolutely sure about what you're doing.

2.3 Extensions and Formats

2.3.1 EXTENSIONS

Allowed extensions for file upload:

```
EXTENSIONS = getattr(settings, "FILEBROWSER_EXTENSIONS", {
    'Folder': [''],
    'Image': ['.jpg','.jpeg','.gif','.png','.tif','.tiff'],
    'Document': ['.pdf','.doc','.rtf','.txt','.xls','.csv'],
    'Video': ['.mov','.wmv','.mpeg','.mpg','.avi','.rm'],
    'Audio': ['.mp3','.mp4','.wav','.aiff','.midi','.m4p']
})
```

2.3.2 SELECT_FORMATS

Set different Options for selecting elements from the FileBrowser:

```
SELECT_FORMATS = getattr(settings, "FILEBROWSER_SELECT_FORMATS", {
    'file': ['Folder','Image','Document','Video','Audio'],
    'image': ['Image'],
    'document': ['Document'],
    'media': ['Video','Audio'],
})
```

When using the browse-function for selecting Files/Folders, you can use an additional query-attribute `type` in order to restrict the choices.

2.4 Versions

2.4.1 VERSIONS_BASEDIR (relative to MEDIA_ROOT)

Changed in version 3.4.0. Directory to save image versions (and thumbnails). If no directory is given, versions are stored at the same location as the original image:

```
VERSIONS_BASEDIR = getattr(settings, 'FILEBROWSER_VERSIONS_BASEDIR', '')
```

Note: In versions previous to FileBrowser 3.4, it was possible to have VERSION_BASEDIR placed at a path which was not browsed by FileBrowser (by placing VERSION_BASEDIR anywhere else than under DIRECTORY).

However, this is not possible as of FileBrowser 3.4 because DIRECTORY variable is not used anymore and FileBrowser browses anything under MEDIA_ROOT. If you don't want FileBrowser to browse/display the contents of VERSION_BASEDIR, make this directory *hidden*.

2.4.2 VERSIONS

Define the versions according to your websites grid:

```
VERSIONS = getattr(settings, "FILEBROWSER_VERSIONS", {
    'admin_thumbnail': {'verbose_name': 'Admin Thumbnail', 'width': 60, 'height': 60, 'opts': 'crop'
    'thumbnail': {'verbose_name': 'Thumbnail (1 col)', 'width': 60, 'height': 60, 'opts': 'crop'},
    'small': {'verbose_name': 'Small (2 col)', 'width': 140, 'height': '', 'opts': ''},
    'medium': {'verbose_name': 'Medium (4col )', 'width': 300, 'height': '', 'opts': ''},
    'big': {'verbose_name': 'Big (6 col)', 'width': 460, 'height': '', 'opts': ''},
    'large': {'verbose_name': 'Large (8 col)', 'width': 680, 'height': '', 'opts': ''},
})
```

2.4.3 ADMIN_VERSIONS

The versions you want to show with the admin-interface:

```
ADMIN_VERSIONS = getattr(settings, 'FILEBROWSER_ADMIN_VERSIONS', ['thumbnail', 'small', 'medium', 'bi
```

2.4.4 ADMIN_THUMBNAIL

The version being used as the admin-thumbnail:

```
ADMIN_THUMBNAIL = getattr(settings, 'FILEBROWSER_ADMIN_THUMBNAIL', 'admin_thumbnail')
```

2.4.5 PLACEHOLDER

Path to placeholder image (relative to MEDIA_ROOT):

```
PLACEHOLDER = getattr(settings, "FILEBROWSER_PLACEHOLDER", "")
```

2.4.6 SHOW_PLACEHOLDER

Show Placeholder (instead of a Version) if the original image does not exist:

```
SHOW_PLACEHOLDER = getattr(settings, "FILEBROWSER_SHOW_PLACEHOLDER", False)
```

2.4.7 FORCE_PLACEHOLDER

Always show placeholder (even if the original image exists):

```
FORCE_PLACEHOLDER = getattr(settings, "FILEBROWSER_FORCE_PLACEHOLDER", False)
```

2.5 Extra Settings

2.5.1 SAVE_FULL_URL

Deprecated since version 3.4.0: With custom storage engines, saving the full URL (including MEDIA_ROOT) doesn't make sense anymore. Moreover, removing this settings allows for easily replacing a FileBrowseField with Djangos File- or ImageField.

2.5.2 STRICT_PIL

If set to `True`, the FileBrowser will not try to import a mis-installed PIL:

```
STRICT_PIL = getattr(settings, 'FILEBROWSER_STRICT_PIL', False)
```

2.5.3 IMAGE_MAXBLOCK

see http://mail.python.org/pipermail/image-sig/1999-August/000816.html:

```
IMAGE_MAXBLOCK = getattr(settings, 'FILEBROWSER_IMAGE_MAXBLOCK', 1024*1024)
```

2.5.4 EXCLUDE

Exclude-patterns for files you don't want to show:

```
EXTENSION_LIST = []
for exts in EXTENSIONS.values():
    EXTENSION_LIST += exts
EXCLUDE = getattr(settings, 'FILEBROWSER_EXCLUDE', (r'_(%(exts)s)_.*_q\d{1,3}\.(%(exts)s)' % {'exts'
```

2.5.5 MAX_UPLOAD_SIZE

Max. Upload Size in Bytes:

```
MAX_UPLOAD_SIZE = getattr(settings, "FILEBROWSER_MAX_UPLOAD_SIZE", 10485760)
```

2.5.6 CONVERT_FILENAME

`True` if you want to convert the filename on upload (replace spaces and convert to lowercase):

```
CONVERT_FILENAME = getattr(settings, "FILEBROWSER_CONVERT_FILENAME", True)
```

2.5.7 LIST_PER_PAGE

How many items appear on each paginated list:

```
LIST_PER_PAGE = getattr(settings, "FILEBROWSER_LIST_PER_PAGE", 50)
```

2.5.8 DEFAULT_SORTING_BY

Default sorting attribute:

```
DEFAULT_SORTING_BY = getattr(settings, "FILEBROWSER_DEFAULT_SORTING_BY", "date")
```

Options are: `date`, `filesize`, `filename_lower`, `filetype_checked`

2.5.9 DEFAULT_SORTING_ORDER

Default sorting order:

```
DEFAULT_SORTING_ORDER = getattr(settings, "FILEBROWSER_DEFAULT_SORTING_ORDER", "desc")
```

Options are: `asc` or `desc`

2.5.10 FOLDER_REGEX

regex to clean dir names before creation:

```
FOLDER_REGEX = getattr(settings, "FILEBROWSER_FOLDER_REGEX", r'^[\w._\ /-]+$')
```

2.5.11 SEARCH_TRAVERSE

New in version 3.3. `True`, if you want to traverse all subdirectories when searching. Please note that with thousands of files/directories, this might take a while:

```
SEARCH_TRAVERSE = getattr(settings, "FILEBROWSER_SEARCH_TRAVERSE", False)
```

2.5.12 DEFAULT_PERMISSIONS

New in version 3.3. Default Upload and Version Permissions:

```
DEFAULT_PERMISSIONS = getattr(settings, "FILEBROWSER_DEFAULT_PERMISSIONS", 0755)
```

2.5.13 OVERWRITE_EXISTING

New in version 3.5.1. `True` in order to overwrite existing files, `False` to use the behaviour of the storage engine:

```
OVERWRITE_EXISTING = getattr(settings, "FILEBROWSER_OVERWRITE_EXISTING", True)
```

FileBrowser Sites

CHAPTER 3

FileBrowser Sites

New in version 3.4.0. As of version 3.4, the FileBrowser application is respresented by an object of `filebrowser.sites.FileBrowserSite` (in analogy to Django's admin site). FileBrowser sites allow you to:

- associate *Custom Actions* (analogy to Django's admin actions) to a site,
- define a *file system storage* for a site (allows for browsing remote file servers) – see *File Storages*,
- subclass from FileBrowserSite and redefine the default FileBrowser's behavior,
- use multiple FileBrowser sites in your project.

The module variable `site` from filebrowser.sites is the default FileBrowser application.

3.1 Backward Incompatibilites

The only thing that you need to pay attention to when migrating to FileBrowser 3.4, is the specification of your URL-patterns. URL-patterns are now associated with a FileBrowser site, that is, each FileBrowser site can have different URL-patterns. See *Quick start guide* for how to setup your URL-patterns.

CHAPTER 4

Custom Actions

New in version 3.4.0. In analogy to Django's admin actions, you can define your FileBrowser actions and thus automate the typical tasks of your users. Registered custom actions are listed in the detail view of a file and a user can select a single action at a time. The selected action will then be applied to the file.

The default FileBrowser image actions, such as "Flip Vertical" or "Rotate 90° Clockwise" are in fact implemented as custom actions (in the module `filebrowser.actions`).

4.1 Writing Your Own Actions

Custom actions are simple functions of the form:

```
def foo(request, fileobjects):
    # Do something with the fileobjects
```

the first parameter is an `HttpRequest` object (representing the submitted form in which a user selected the action) and the second parameter is a list of `FileObjects` to which the action should be applied.

In the current FileBrowser version, the list contains exactly one instance of FileObject (representing the file from the detail view), but this may change in the future, as custom actions may become available also in browse views (similar to admin actions applied to a list of checked objects).

4.2 Registering an Action

In order to make your action visible, you need to register it at a FileBrowser site (see also *FileBrowser Sites*):

```
site.add_action(foo)
```

Once registered, the action will appear in the detail view of a file. You can also give your action a short description:

```
foo.short_description = 'Do foo with the File'
```

This short description will then appear in the list of available actions. If you do not provide any short description for your action, the function name will be used instead and FileBrowser will replace any underscores in the function name with spaces.

4.3 Associating Actions with Specific Files

Each custom action can be associated with a specific file type (e.g., images, audio file, etc) to which it applies. In order to do that, you need to define a predicate/filter function, which takes a single argument – a FileObject – and returns `True` if your action is applicable to that FileObject. Finally, you need to register this filter function with your action:

```
foo.applies_to(lambda fileobject: fileobject.filetype == 'Image')
```

In the above example, foo will appear in the action list only for image files. If you do not specify any filter function for your action, FileBrowser considers the action as applicable to all files.

4.4 Messages & Intermediate Pages

You can provide a feedback to a user about or successful or failed execution of an action by registering a message at the request object. For example:

```
from django.contrib import messages

def desaturate_image(request, fileobjects):
    for f in fileobjects:
        # Desaturate the image
        messages.add_message(request, messages.SUCCESS, _("Image '%s' was desaturated.") % f.filename
```

Some actions may require user confirmation (e.g., in order to prevent accidental and irreversible modification to files). In order to that, follow the same pattern as with Django's admin action and return an `HttpResponse` object from your action. Good practice for intermediate pages is to implement a confirm view and have your action return an `HttpResponseRedirect` object redirecting a user to that view:

```
def crop_image(request, fileobjects):
    files = '&f='.join([f.path_relative for f in fileobjects])
    return HttpResponseRedirect('/confirm/?action=crop_image&f=%s' % files)
```

CHAPTER 5

File Storages

New in version 3.4.0. Starting with FileBrowser 3.4, you have the option to specify which file storage engine a FileBrowser should use to browse/upload/modify your media files. This enables you to use a FileBrowser even if your media files are located at some remote system. See also the Django's documentation on storages https://docs.djangoproject.com/en/dev/topics/files/.

To associate a FileBrowser site with a particular storage, set the `storage` property of a site object:

```
from django.core.files.storage import FileSystemStorage
site.storage = FileSystemStorage(location='/path/to/media/directory', base_url='/media/')
```

For storage classes other than FileSystemStorage (or those that inherit from that class), there's a little bit more effort involved in providing a storage object that can be used with FileBrowser. See *StorageMixin Class*

Note: Prior FileBrowser 3.4, the way to specify FileBrowser's MEDIA_ROOT and MEDIA_URL was via `settings.py`. Starting from version 3.4, those variables are associated with the storage instance and you can set them as illustrated in the above example.

Warning: For the reason of backward compatibility, FileBrowser settings `FILEBROWSER_MEDIA_ROOT` and `FILEBROWSER_MEDIA_URL` can still be used to customize FileBrowser as long as you're using the default FileBrowser's site without having changed its storage engine. In the next major release of FileBrowser these settings will be removed.

5.1 StorageMixin Class

A FileBrowser uses the Django's Storage class to access media files. However, the API of the Storage class does not provide all methods necessary for FileBrowser's functionality. A `StorageMixin` class from `filebrowser.storage` module therefore defines all the additional methods that a FileBrowser requires:

`isdir(self, name):`
Returns true if name exists and is a directory.

`isfile(self, name):`
Returns true if name exists and is a regular file.

`move(self, old_file_name, new_file_name, allow_overwrite=False):`

Moves safely a file from one location to another. If `allow_ovewrite==False` and *new_file_name* exists, raises an exception.

`makedirs(self, name):`
Creates all missing directories specified by name. Analogue to os.mkdirs().

`rmtree(self, name):`
Deletes a directory and everything it contains. Analogue to shutil.rmtree().

Note: FileBrowser provides these methods only for FileSystemStorage (by mixing-in the `filebrowser.storage.FileSystemStorageMixin` class). If you're using a custom storage engine, which does not inherit from Django's FileSystemStorage, you will need to provide those method yourself.

FileBrowser API

CHAPTER 6

FileListing class

The FileListing is a group of FileObjects for a given directory:

```
filelisting = FileListing(path, filter_func=None, sorting_by=None, sorting_order=None)
```

For example, if you want to list all files for MEDIA_ROOT you can type:

```
from filebrowser.settings import MEDIA_ROOT
filelisting = FileListing(MEDIA_ROOT, sorting_by='date', sorting_order='desc')
```

6.1 Options

6.1.1 filter_func

Filter function, based on a FileObject:

```
def filter_filelisting(item):
    return item.filetype != "Folder"
```

6.1.2 sorting_by

Sort the files by any attribute of FileObject (e.g. filetype, date, ...).

6.1.3 sorting_order

Sorting order, either asc or desc.

6.2 Attributes

For the below examples, we're using this folder-structure and filter_browse as filter_func (see views.py):

```
/media/uploads/testfolder/testimage.jpg
/media/uploads/blog/1/images/blogimage.jpg
```

6.2.1 listing

Returns all items for the given path with `os.listdir(path)`:

```
>>> for item in filelisting.listing():
...     print item
blog
testfolder
```

6.2.2 walk

Returns all items for the given path with `os.walk(path)`:

```
>>> for item in filelisting.walk():
...     print item
blog
blog/1
blog/1/images
blog/1/images/blogimage.jpg
blog/1/images/blogimage_admin_thumbnail.jpg
blog/1/images/blogimage_medium.jpg
blog/1/images/blogimage_small.jpg
blog/1/images/blogimage_thumbnail.jpg
testfolder
testfolder/testimage.jpg
```

6.2.3 files_listing_total

Returns a sorted list of `FileObjects` for `self.listing()`:

```
>>> for item in filelisting.walk():
...     print item
uploads/blog/
uploads/testfolder/
```

6.2.4 files_walk_total

Returns a sorted list of `FileObjects` for `self.walk()`:

```
>>> for item in filelisting.files_walk_total():
...     print item
uploads/blog/
uploads/blog/1/
uploads/blog/1/images/
uploads/blog/1/images/blogimage.jpg
uploads/blog/1/images/blogimage_admin_thumbnail.jpg
uploads/blog/1/images/blogimage_medium.jpg
uploads/blog/1/images/blogimage_small.jpg
uploads/blog/1/images/blogimage_thumbnail.jpg
```

```
uploads/testfolder/
uploads/testfolder/testimage.jpg
```

6.2.5 files_listing_filtered

Returns a sorted and filtered list of `FileObjects` for `self.listing()`:

```
>>> for item in filelisting.files_listing_filtered():
...     print item
uploads/blog/
uploads/testfolder/
```

6.2.6 files_walk_filtered

Returns a sorted and filtered list of `FileObjects` for `self.walk()`:

```
>>> for item in filelisting.files_walk_filtered():
...     print item
uploads/blog/
uploads/blog/1/
uploads/blog/1/images/
uploads/blog/1/images/blogimage.jpg
uploads/testfolder/
uploads/testfolder/testimage.jpg
```

Note: The versions are not listed (compared with files_walk_total) because of `filter_func`.

6.2.7 results_listing_total

Number of total files, based on `files_listing_total`:

```
>>> filelisting.results_listing_total()
2
```

6.2.8 results_walk_total

Number of total files, based on `files_walk_total`:

```
>>> filelisting.results_listing_total()
10
```

6.2.9 results_listing_filtered

Number of filtered files, based on `files_listing_filtered`:

```
>>> filelisting.results_listing_filtered()
2
```

6.2.10 `results_walk_filtered`

Number of filtered files, based on `files_walk_filtered`:

```
>>> filelisting.results_walk_filtered()
6
```

CHAPTER 7

FileObject class

When browsing a directory on the server, you get a `FileObject` (see `base.py`) for every file within that directory. The `FileObject` is also returned when using the `FileBrowseField`:

```
fileobject = FileObject("relative/server/path/to/storage/location/file.ext")
```

For the examples below we use:

```
fileobject = FileObject(os.path.join(MEDIA_ROOT,DIRECTORY,"testfolder","testimage.jpg"))
```

7.1 General attributes

7.1.1 filename

Name of the file (including the extension) or name of the folder:

```
>>> print fileobject.filename
'testimage.jpg'
```

7.1.2 filetype

Type of the file, as defined with `EXTENSIONS`. With a folder, the filetype is "Folder":

```
>>> print fileobject.filetype
'Image'
```

7.1.3 mimetype

New in version 3.3. Mimetype, based on http://docs.python.org/library/mimetypes.html:

```
>>> print fileobject.mimetype
('image/jpeg', None)
```

7.1.4 filesize

Filesize in Bytes. Display with `filesizeformat`:

```
>>> print fileobject.filesize
870037L
```

7.1.5 extension

File extension, including the dot. With a folder, the extensions is `None`:

```
>>> print fileobject.extension
'.jpg'
```

7.1.6 date

Date, based on `getmtime`:

```
>>> print fileobject.date
1299760347.0
```

7.1.7 datetime

Datetime object:

```
>>> print fileobject.datetime
datetime.datetime(2011, 3, 10, 13, 32, 27)
```

7.2 Path and URL attributes

7.2.1 path

Absolute server path to the file/folder, including `MEDIA_ROOT`:

```
>>> print fileobject.path
'/var/www/testsite/media/uploads/testfolder/testimage.jpg'
```

7.2.2 path_relative

Server path to the file/folder, relative to `MEDIA_ROOT`:

```
>>> print fileobject.path_relative
u'uploads/testfolder/testimage.jpg'
```

7.2.3 url_full

Deprecated since version 3.3: Use `url` instead.

7.2.4 url

New in version 3.3. URL for the file/folder, including MEDIA_URL:

```
>>> print fileobject.url
u'/media/uploads/testfolder/testimage.jpg'
```

7.2.5 url_relative

URL for the file/folder, relative to MEDIA_URL:

```
>>> print fileobject.url_relative
u'uploads/testfolder/testimage.jpg'
```

7.2.6 url_save

URL for the file/folder, based on the setting SAVE_FULL_URL used for the FileBrowseField (either url or url_relative):

```
>>> print fileobject.url_save
u'uploads/testfolder/testimage.jpg'
```

7.3 Image attributes

7.3.1 dimensions

Image dimensions as a tuple:

```
>>> print fileobject.dimensions
(1000, 750)
```

7.3.2 width

Image width in px:

```
>>> print fileobject.width
1000
```

7.3.3 height

Image height in px:

```
>>> print fileobject.height
750
```

7.3.4 aspectratio

Aspect ratio (float format):

```
>>> print fileobject.aspectratio
1.33534908
```

7.3.5 orientation

Image orientation, either "landscape" or "portrait":

```
>>> print fileobject.orientation
'Landscape'
```

7.4 Folder attributes

7.4.1 folder

```
>>> print fileobject.folder
u'testfolder'
```

7.4.2 is_folder

`true`, if path is a folder:

```
>>> print fileobject.is_folder
False
```

7.4.3 is_empty

`true`, if the folder is empty:

```
>>> print fileobject.is_empty
```

7.5 Version attributes

7.5.1 is_version

`true` if the File is a `version` of another File:

```
>>> print fileobject.is_version
False
```

7.5.2 version_basedir

The absolute path to the versions-folder:

```
>>> print fileobject.version_basedir
'/var/www/testsite/media/uploads/testfolder'
```

7.5.3 version_name(version_suffix)

Get the filename for a version:

```
>>> print fileobject.version_name("medium")
'testimage_medium.jpg'
```

Note: The version is not being generated.

7.5.4 versions()

List all filenames based on `VERSIONS`:

```
>>> print fileobject.versions()
['/var/www/testsite/media/uploads/testfolder/testimage_admin_thumbnail.jpg',
'/var/www/testsite/media/uploads/testfolder/testimage_thumbnail.jpg',
'/var/www/testsite/media/uploads/testfolder/testimage_small.jpg',
'/var/www/testsite/media/uploads/testfolder/testimage_medium.jpg',
'/var/www/testsite/media/uploads/testfolder/testimage_big.jpg',
'/var/www/testsite/media/uploads/testfolder/testimage_large.jpg']
```

Note: The versions are not being generated.

7.5.5 admin_versions()

List all filenames based on `ADMIN_VERSIONS`:

```
>>> print fileobject.admin_versions()
['/var/www/testsite/media/uploads/testfolder/testimage_thumbnail.jpg',
'/var/www/testsite/media/uploads/testfolder/testimage_small.jpg',
'/var/www/testsite/media/uploads/testfolder/testimage_medium.jpg',
'/var/www/testsite/media/uploads/testfolder/testimage_big.jpg',
'/var/www/testsite/media/uploads/testfolder/testimage_large.jpg']
```

Note: The versions are not being generated.

7.5.6 version_generate(version_suffix)

Generate a version:

```
>>> print fileobject.version("medium")
uploads/testfolder/testimage_medium.jpg
```

7.6 Functions

7.6.1 delete()

Delete the File or Folder from the server.

Warning: If you delete a Folder, all items within the folder are being deleted. Be very careful with using this!

7.6.2 delete_versions()

Delete all VERSIONS.

7.6.3 delete_admin_versions()

Delete all ADMIN_VERSIONS.

Admin Interface

CHAPTER 8

Views

All views use the `staff_member_requird` and `path_exists` decorator in order to check if the server path actually exists. Some views also use the `file_exists` decorator.

8.1 Browse

Browse a directory on your server. Returns a *FileListing class*:

`http://mysite.com/adminurl/filebrowser/browse/`

- URL: `fb_browse`
- Optional query string args: `dir`, `o`, `ot`, `q`, `p`, `filter_date`, `filter_type`, `type`

8.2 Create directory

Create a new folder on your server:

`http://mysite.com/adminurl/filebrowser/createdir/`

- URL: `fb_createdir`
- Optional query string args: `dir`
- Signals: *filebrowser_pre_createdir*, *filebrowser_post_createdir*

8.3 Upload

Multiple upload:

`http://mysite.com/adminurl/filebrowser/upload/`

- URL: `fb_upload`
- Optional query string args: `dir`
- Signals: *filebrowser_pre_upload*, *filebrowser_post_upload*

8.4 Edit

Edit a file or folder:

`http://mysite.com/adminurl/filebrowser/edit/?filename=testimage.jpg`

- URL: `fb_edit`
- Required query string args: `filename`
- Optional query string args: `dir`
- Signals: *filebrowser_pre_rename*, *filebrowser_post_rename*

You are able to apply custom actions (see *Custom Actions*) to the edit-view.

Note: This won't check if you use the file or folder anywhere with your models.

8.5 Confirm delete

Confirm the deletion of a file or folder:

`http://mysite.com/adminurl/filebrowser/confirm_delete/?filename=testimage.jpg`

- URL: `fb_confirm_delete`
- Required query string args: `filename`
- Optional query string args: `dir`

Note: If you try to delete a folder, all files/folders within this folder are listed on this page.

8.6 Delete

Delete a file or folder:

`http://mysite.com/adminurl/filebrowser/delete/?filename=testimage.jpg`

- URL: `fb_delete`
- Required query string args: `filename`
- Optional query string args: `dir`
- Signals: *filebrowser_pre_delete*, *filebrowser_post_delete*

Note: This won't check if you use the file or folder anywhere with your models.

Warning: If you delete a Folder, all items within this Folder are being deleted.

8.7 Version

Generate a version of an Image as defined with `ADMIN_VERSIONS`:

`http://mysite.com/adminurl/filebrowser/version/?filename=testimage.jpg`

- URL: `fb_version`
- Required query string args: `filename`
- Query string args: `dir`

Note: This is a helper used by the `FileBrowseField` and TinyMCE for selecting an Image-Version.

CHAPTER 9

Signals

The FileBrowser sends a couple of different signals:

9.1 filebrowser_pre_upload

Sent before a an Upload starts. Arguments:

- `path`: Absolute server path to the file/folder
- `name`: Name of the file/folder
- `site`: Current `FileBrowserSite` instance

9.2 filebrowser_post_upload

Sent after an Upload has finished. Arguments:

- `path`: Absolute server path to the file/folder
- `name`: Name of the file/folder
- `site`: Current `FileBrowserSite` instance

9.3 filebrowser_pre_delete

Sent before an Item (File, Folder) is deleted. Arguments:

- `path`: Absolute server path to the file/folder
- `name`: Name of the file/folder
- `site`: Current `FileBrowserSite` instance

9.4 filebrowser_post_delete

Sent after an Item (File, Folder) has been deleted. Arguments:

- `path`: Absolute server path to the file/folder
- `name`: Name of the file/folder
- `site`: Current `FileBrowserSite` instance

9.5 filebrowser_pre_createdir

Sent before a new Folder is created. Arguments:

- `path`: Absolute server path to the folder
- `name`: Name of the new folder
- `site`: Current `FileBrowserSite` instance

9.6 filebrowser_post_createdir

Sent after a new Folder has been created. Arguments:

- `path`: Absolute server path to the folder
- `name`: Name of the new folder
- `site`: Current `FileBrowserSite` instance

9.7 filebrowser_pre_rename

Sent before an Item (File, Folder) is renamed. Arguments:

- `path`: Absolute server path to the file/folder
- `name`: Name of the file/folder
- `new_name`: New name of the file/folder
- `site`: Current `FileBrowserSite` instance

9.8 filebrowser_post_rename

Sent after an Item (File, Folder) has been renamed.

- `path`: Absolute server path to the file/folder
- `name`: Name of the file/folder
- `new_name`: New name of the file/folder
- `site`: Current `FileBrowserSite` instance

9.9 filebrowser_actions_pre_apply

Sent before a custom action is applied. Arguments:

- `action_name`: Name of the custom action
- `fileobjects`: A list of fileobjects the action will be applied to
- `site`: Current `FileBrowserSite` instance

9.10 filebrowser_actions_post_apply

Sent after a custom action has been applied.

- `action_name`: Name of the custom action
- `fileobjects`: A list of fileobjects the action has been be applied to
- `results`: The response you defined with your custom action
- `site`: Current `FileBrowserSite` instance

9.11 Example for using these Signals

Here's a small example for using the above Signals:

```
from filebrowser import signals

def pre_upload_callback(sender, **kwargs):
    """
    Receiver function called before an upload starts.
    """
    print "Pre Upload Callback"
    print "kwargs:", kwargs
signals.filebrowser_pre_upload.connect(pre_upload_callback)

def post_upload_callback(sender, **kwargs):
    """
    Receiver function called each time an upload has finished.
    """
    print "Post Upload Callback"
    print "kwargs:", kwargs
    # You can use all attributes available with the FileObject
    # This is just an example ...
    print "Filesize:", kwargs['file'].filesize
    print "Orientation:", kwargs['file'].orientation
    print "Extension:", kwargs['file'].extension
signals.filebrowser_post_upload.connect(post_upload_callback)
```

Fields and Widgets

CHAPTER 10

FileBrowseField

The `FileBrowseField` is a `Model field` for selecting a file from your Media Server:

```
from filebrowser.fields import FileBrowseField

class BlogEntry(models.Model):

    image = FileBrowseField("Image", max_length=200, directory="images/", extensions=[".jpg"], blank=
    document = FileBrowseField("PDF", max_length=200, directory="documents/", extensions=[".pdf",".do
    ...
```

10.1 Attributes

`max_length` Since the `FileBrowseField` is a `CharField`, you have to define `max_length`.

`site` (optional) The FileBrowser site you want to use with this field. Defaults to the main site, if not given.

`directory` (optional) Subdirectory of `DIRECTORY`. If `DIRECTORY` is not defined, subdirectory of `MEDIA_ROOT`. Do not prepend a slash.

`extensions` (optional) List of allowed extensions.

`format` (optional) Use this attribute to restrict selection to specific filetypes. E.g., if you use format='image', only Images can be selected from the FileBrowser. Note: The `format` has to be defined within `SELECT_FORMATS`.

10.2 FileBrowseField in Templates

When using a `FileBrowseField`, you'll get a *FileObject class* back.

With the above Model, you can use:

```
{{ blogentry.image }}
```

to output the contents of your image-field. For example, this could result in something like "myimage.jpg".

Now, if you want to actually display the Image, you write:

```
<img src="{{ publication.image.url }}" />
```

More complicated, if you want to display "Landscape" Images only:

```
{% ifequal blogentry.image.image_orientation "landscape" %}
    <img src="{{ blogentry.image.url }}" class="landscape" />
{% endifequal %}
```

10.3 Showing Thumbnail in the Changelist

If you want to show a Thumbnail in the Changelist, you can define a Model-/Admin-Method:

```
from filebrowser.settings import ADMIN_THUMBNAIL

def image_thumbnail(self, obj):
    if obj.image and obj.image.filetype == "Image":
        return '<img src="%s" />' % obj.image.version_generate(ADMIN_THUMBNAIL).url
    else:
        return ""
image_thumbnail.allow_tags = True
image_thumbnail.short_description = "Thumbnail"
```

10.4 Using the FileBrowseField with TinyMCE

You can also replace the TinyMCE Image/File Manager with the FileBrowser. What you have to do is a FileBrowser Callback. There's an example in /media/js/ called TinyMCEAdmin.js (a TinyMCE Configuration File). You can copy the FileBrowserCallback to your own file, take a look at http://wiki.moxiecode.com/index.php/TinyMCE:Custom_filebrowser or just use tinymce_setup.js (which comes with django-grappelli).

Just add these lines to your AdminModel:

```
class Media:
    js = ['/path/to/tinymce/jscripts/tiny_mce/tiny_mce.js', '/path/to/your/tinymce_setup.js',]
```

CHAPTER 11

FileInput

Subclass of `FileInput` with an additional Image-Thumbnail:

```
from filebrowser.widgets import FileInput

class BlogEntryOptions(admin.ModelAdmin):
    formfield_overrides = {
        models.ImageField: {'widget': FileInput},
    }
```

CHAPTER 12

ClearableFileInput

Subclass of `ClearableFileInput` with an additional Image-Thumbnail:

```
from filebrowser.widgets import ClearableFileInput

class BlogEntryOptions(admin.ModelAdmin):
    formfield_overrides = {
        models.ImageField: {'widget': ClearableFileInput},
    }
```

Django `FileField` and the FileBrowser

Generate a `FileObject` from a `FileField` or `ImageField` with:

```
from filebrowser.base import FileObject

image_upload = models.ImageField(u"Image (Upload)", max_length=250, upload_to=image_upload_path, blan

def image(self):
    if self.image_upload:
        return FileObject(self.image_upload.path)
    return None
```

To show a Thumbnail on your changelist, you could use a ModelAdmin-Method:

```
from filebrowser.base import FileObject

def image_thumbnail(self, obj):
    if obj.image_upload:
        image = FileObject(obj.image_upload.path)
        if image.filetype == "Image":
            return '<img src="%s" />' % image.version_generate(ADMIN_THUMBNAIL).url
    else:
        return ""
image_thumbnail.allow_tags = True
image_thumbnail.short_description = "Thumbnail"
```

Note: There are different ways to achieve this. The above examples show one of several options.

Image Versions

CHAPTER 14

Image Versions

With the FileBrowser, you are able to define different versions/sizes for Images. This enables you to save an Original Image on your Server while having different versions of that Image to automatically fit your websites Grid. Versions are also useful for responsive/adaptive layouts.

14.1 Versions and the grid

First you need to know which versions/sizes of an image you'd like to use on your website. Let's say you're using a 12 column grid with 60px for each column and 20px margin (which is a total of 940px). With this grid, you could (for example) generate these image versions:

```
VERSIONS = getattr(settings, "FILEBROWSER_VERSIONS", {
  'admin_thumbnail': {'verbose_name': 'Admin Thumbnail', 'width': 60, 'height': 60, 'opts': 'crop'},
  'thumbnail': {'verbose_name': 'Thumbnail (1 col)', 'width': 60, 'height': 60, 'opts': 'crop'},
  'small': {'verbose_name': 'Small (2 col)', 'width': 140, 'height': '', 'opts': ''},
  'medium': {'verbose_name': 'Medium (4col )', 'width': 300, 'height': '', 'opts': ''},
  'big': {'verbose_name': 'Big (6 col)', 'width': 460, 'height': '', 'opts': ''},
  'large': {'verbose_name': 'Large (8 col)', 'width': 680, 'height': '', 'opts': ''},
})
```

New in version 3.4.0: `methods` If you need to add some filter for the version (like grayscale, sepia, ...), you can also use the `methods` argument:

```
def grayscale(im):
    '''Convert the PIL image to grayscale'''
    if im.mode != "L":
        im = im.convert("L")
    return im

FILEBROWSER_VERSIONS = {
    'big': {'verbose_name': 'Big (6 col)', 'width': 460, 'height': '', 'opts': '', 'methods': [grays
})
```

14.2 Versions with the admin-interface

With the admin-interface, you need to define `ADMIN_VERSIONS`:

```
ADMIN_VERSIONS = getattr(settings, 'FILEBROWSER_ADMIN_VERSIONS', ['thumbnail', 'small', 'medium', 'bi
```

Don't forget to select one version for your admin-thumbnail:

```
ADMIN_THUMBNAIL = getattr(settings, 'FILEBROWSER_ADMIN_THUMBNAIL', 'admin_thumbnail')
```

14.3 Versions on your website

In order to generate versions, you have two different tags to choose from: `version` and `version_object`. With either using the version-tag or the version_object-tag, the Image-version will be generated "on the fly" if the version doesn't already exist or if the original Image is newer than the version. This means that if you need to update an Image, you just overwrite the original Image - the versions will be re-generated automatically (as you request them within your template).

A Model example:

```
from filebrowser.fields import FileBrowseField

class BlogEntry(models.Model):
    image = FileBrowseField("Image", max_length=200, blank=True, null=True)
```

First you need to load the templatetags with:

```
{% load fb_versions %}
```

You have two different tags to choose from: `version` and `version_object`.

Note: With both templatetags, `version_prefix` can either be a string or a variable. If `version_prefix` is a string, use quotes.

14.3.1 Templatetag `version`

Generate a version and retrieve the URL:

```
{% version model.field_name version_prefix %}
```

With the above Model, in order to generate a version you type:

```
{% version blogentry.image 'medium' %}
```

Since you retrieve the URL, you can display the image with:

```
<img src="{% version blogentry.image 'medium' %}" />
```

14.3.2 Templatetag `version_object`

Generate a version and retrieve the FileObject:

```
{% version_object model.field_name version_prefix as variable %}
```

With the above Model, in order to generate a version you type:

```
{% version_object blogentry.image 'medium' as version_medium %}
```

Since you retrieve a `FileObject`, you can use all attributes:

```
{{ version_medium.width }}
```

or just:

```
<img src="{{ version_medium }}" />
```

14.4 Versions in views

If you have a `FileObject` you can easily generate/retrieve a version with:

```
obj.image.version(version_prefix)
```

So, if you need to generate/retrieve the admin thumbnail for an Image, you can type:

```
from filebrowser.settings import ADMIN_THUMBNAIL
obj.image.version_generate(ADMIN_THUMBNAIL).url
```

14.5 Placeholder

When developing on a locale machine or a development-server, you might not have all the images (resp. media-files) available that are on your production instance and downloading these files on a regular basis might not be an option.

In that case, you might wanna use a placeholder instead of an image-version. You just need to define the `PLACEHOLDER` and overwrite the settings `SHOW_PLACEHOLDER` and/or `FORCE_PLACEHOLDER` (see *PLACEHOLDER*).

CHAPTER 15

Management Commands

15.1 Command `fb_version_generate`

If you need to generate certain (or all) versions, type:

```
python manage.py fb_version_generate
```

15.2 Command `fb_version_remove`

If you need to generate certain (or all) versions, type:

```
python manage.py fb_version_remove
```

> **Warning:** Please be very careful with this command.

Help

CHAPTER 16

FAQ

Some questions, some answers.

16.1 Why should I use the FileBrowser?

If you need your editors or customer to manage files, the FileBrowser is an alternative to an FTP-client. Moreover, with the FileBrowser you are able to define different image-versions according to your websites grid. Alternatives to the FileBrowser can be found at http://djangopackages.com/grids/g/file-managers/.

16.2 I want to use FileBrowser, but I don't want to use Grappelli?

Grappelli is a requirement for using the FileBrowser. There are several filebrowser-no-grappelli versions (most of them on GitHub), but we don't follow the development.

16.3 I need help!

see *Troubleshooting*.

16.4 Why are there no fancy effects?

The FileBrowser is about managing files. We think that you should prepare your files *before* uploading them to the server.

16.5 How do I upload to another server?

Use a custom storage engine, see https://docs.djangoproject.com/en/1.3/howto/custom-file-storage/.

16.6 Why should I need image-versions?

You need image-versions if your website is based on a *grid*.

16.7 Is the FileBrowser stable?

We've developed the FileBrowser for a couple of years and use it with almost all of our clients. That said, Grappelli is the more stable and mature application.

16.8 How can I contribute?

Help is very much needed and appreciated. Test the FileBrowser and submit feedback/patches.

16.9 Who develops the FileBrowser?

The FileBrowser is developed and maintained by Patrick Kranzlmüller & Axel Swoboda of vonautomatisch.

CHAPTER 17

Troubleshooting

Sometimes you might have a problem installing/using the FileBrowser.

17.1 Check your setup

First, please check if the problem is caused by your setup.

- Read *Quick start guide*.
- Check if the static/media-files are served correctly.
- Make sure you have removed all customized filebrowser templates from all locations in TEMPLATE_DIRS paths or check that these templates are compatible with the FileBrowser.

17.2 Run the FileBrowser tests

Start the shell and type:

```
python manage.py test filebrowser
```

17.3 Check issues

If your setup is fine, please check if your problem is a known issue.

- Take a look at all FileBrowser Issues (incuding closed) and search the FileBrowser Google-Group.

17.4 Add a ticket

If you think you've found a bug, please add a ticket.

- Try to describe your problem as precisely as possible.
- Tell us what you did in order to solve the problem.

- Tell us what version of the FileBrowser you are using.
- Tell us what version of Django you are using.
- Please do NOT add tickets if you're having problems with serving static/media-files (because this is not related to the FileBrowser).
- Please do NOT add tickets referring to Djangos trunk version.
- At best: add a patch.

Note: Be aware that we may close issues not following these guidlines without further notifications.

CHAPTER 18

Translation

New in version 3.3. Translation is done via Transifex.

CHAPTER 19

Supported Languages

see https://www.transifex.net/projects/p/django-filebrowser/resource/djangopo/

CHAPTER 20

FileBrowser 3.5 Release Notes

FileBrowser 3.5 is compatible with Django 1.4/1.5 and Grappelli 2.4.

20.1 Update from FileBrowser 3.4.x

- Update Django to 1.4 or 1.5 and check https://docs.djangoproject.com/en/dev/releases/1.4/ or https://docs.djangoproject.com/en/dev/releases/1.5/
- Update Grappelli to 2.4.x
- Update FileBrowser to 3.5.x

CHAPTER 21

Changelog

21.1 3.5.3 (not yet released)

21.2 3.5.2 (February 22, 2013)

- Fixed: Use placeholder with version_generate (not only templatetags).
- Fixed: translate extension group name in upload form.
- Fixed: updated filter dropdown HTML.
- Fixed: Make setup.py work with Python 3.
- Fixed: File submit with search traversal.
- Fixed: Fixed fileobject path with Windows.
- Improved: Throwing an exception when in DEBUG and version is not generated (with using the templatetag).
- Compatibility with Django 1.5.

21.3 3.5.1 (November 09, 2012)

- Fixed: Documentation with Signals.
- Fixed: File Upload using basic submission.
- Fixed: Added site instance to Signals.
- Improved: Don't hide errors during generate-command.
- Improved: Follow symlinks with generate-command.
- Improved: Added some translations (e.g. for "Upload File").
- New: Setting OVERWRITE_EXISTING.
- New: Added file ‘‘signals.py‘.
- New: Support for Django 1.5.

21.4 3.5.0 (July 20, 2012)

- Compatibility with Django 1.4 and Grappelli 2.4.

21.5 3.4.3 (20.6.2012)

- Fixed a bug with versions not being generated (in case of capitalized extensions).

21.6 3.4.2 (26.3.2012)

- Fixed security bug: added staff_member_required decorator to the upload-function.
- Fixed a XSS vulnerability with fb_tags.

21.7 3.4.1 (7.3.2012)

- Fixed an error with quotes (french translation) in upload.html.
- Updated translations.
- FileObject now returns path (with __unicode__ and __str__), instead of filename. This is needed because otherwise form.has_changed will always be triggerde when using a FileBrowseField.
- Fixed a bug with versions and "f referenced before assignment" (e.g. when an image is being deleted)
- Updated docs (warning that FILEBROWSER_MEDIA_ROOT and FILEBROWSER_MEDIA_URL will be removed with the next major release – use custom storage engine instead).
- Fixed issue with MEDIA_URL hardcoded in tests.
- Fixed issue when MEDIA_URL starts with https://.
- Fixed issue with default-site (if no site is given).
- Fixed bug with using L10N and MAX_UPLOAD_SIZE in upload.html.
- Fixed small bug with importing Http404 in sites.py.
- Fixed bug with Fileobject.exists.
- Added NORMALIZE_FILENAME.

21.8 3.4.0 (15/11/2011)

- Final release of 3.4, see *FileBrowser 3.5 Release Notes*

CHAPTER 22

Main Features

- Browse your media files with the admin-interface.
- Multiple Upload, including a progress bar.
- Automatic Thumbnails.
- Image-Versions to fit your websites grid (esp. useful with adaptive/responsive layouts).
- Integration with TinyMCE (AdvImage & AdvLink).
- `FileBrowseField` to select Images/Documents.
- `FileInput` and `ClearableFileInput` with Image-Preview.
- Signals for Upload, Rename and Delete.
- Custom Actions.
- Custom File Storage Engines.

CHAPTER 23

Code

https://github.com/sehmaschine/django-filebrowser

CHAPTER 24

Discussion

Use the FileBrowser Google Group to ask questions or discuss features.

CHAPTER 25

Versions and Compatibility

- FileBrowser 3.5.3 (Development Version, not yet released, see Branch Stable/3.5.x)
- FileBrowser 3.5.2 (February 22 2013): Compatible with Django 1.4/1.5
- FileBrowser 3.4.3 (April 2012): Compatible with Django 1.3

Older versions are availabe at GitHub, but are not supported anymore.