PyUploadcare Documentation

Release 2.4.0

Uploadcare LLC

April 25, 2018

Contents

The most important thing for us at Uploadcare is to make file uploading on the web really easy. Everyone is used to the routine work, related to allowing users upload their userpics or attach resumes: from installing image processing libraries to creating folder with right permissions to ensuring the server never goes down or out of space to enabling CDN. Feature like ability to simply use a picture from Facebook or manual cropping are much more burdensome, hence rare. Uploadcare's goal is to change the status quo.

This library consists of an API interface for Uploadcare and a couple of Django goodies.

A simple Uploadcare `FileField` can be added to an existing Django project in just a couple of *simple steps*. As a result, your users are going to be able to see the progress of the upload, choose files from Google Drive or Instagram, and edit form while files are uploading asynchornously.

Contents:

CHAPTER 1

Installation

This part of the documentation covers the installation of PyUploadcare.

Pip

Installing pyuploadcare is simple with pip:

```
$ pip install pyuploadcare
```

or, if you're into vintage:

```
$ easy_install pyuploadcare
```

Get the Code

PyUploadcare is developed on GitHub. You can clone the public repository:

```
$ git clone git://github.com/uploadcare/pyuploadcare.git
```

After that you can install it:

```
$ python setup.py install
```

Update to version 2.0

Some caveats about migration process from version 1.x to 2.x.

A version 2.0 contains the next breaking changes:

- Now, you should import Django models' fields (e.g `ImageField`) directly from the `pyuploadcare.dj.models` module.
- Changed initializing for the `FileList` and `GroupList` classes. The `since` and `until` parameters have been removed. Use combination of `starting_point` and `ordering` instead.
- The `ucare list` CLI command has been renamed to `ucare list_files`. And, according to the previous change, the `since` and `until` parameters have been removed. The `starting_point` and ordering parameters added.

These last two changes are necessary for working with version 0.5 of REST API. So that means you can't use these classes correctly with versions prior 0.5 (but that should not be an issue :)

Also, note that Django configuration option `UPLOADCARE['widget_variant']` now is deprecated and it will be removed in next major release. Use `UPLOADCARE['widget_build']` instead.

CHAPTER 2

Quickstart

This page gives a good introduction in how to get started with PyUploadcare. This assumes you have already installed PyUploadcare. If you do not, head over to the *Installation* section.

Warning: Keep in mind that Uploadcare signature authentication will fail if computer clock is not synchronized.

Get API Keys

First of all, you'll need API keys: public and private. You can get them at the Uploadcare website. If you don't have an account yet, you can use demo keys, as in example. However, the files on demo account are regularly deleted, so create an account as soon as Uploadcare catches your fancy.

How to use it with Django?

Assume you have a Django project with *gallery* app.

Application Setup

Add `pyuploadcare.dj` into `INSTALLED_APPS`:

```
INSTALLED_APPS = (
    # ...
    'pyuploadcare.dj',

    'gallery',
)
```

As soon as you *got your API keys*, add them to your Django settings file:

```
UPLOADCARE = {
    'pub_key': 'demopublickey',
    'secret': 'demoprivatekey',
}
```

Uploadcare image field adding to your `gallery/models.py` is really simple. Like that:

```
from django.db import models

from pyuploadcare.dj.models import ImageField


class Photo(models.Model):

    title = models.CharField(max_length=255)
    photo = ImageField()
```

`ImageField` doesn't require any arguments, file paths or whatever. **It just works.** That's the point of it all. It looks nice in the admin interface as well:

Obviously, you would want to use Uploadcare field outside an admin. It's going to work just as well, but, however, you have to remember to add `{{ form.media }}` in the `<head>` tag of your page:

```
{{ form.media }}

<form action="" method="post">
    {% csrf_token %}
    {{ form.as_p }}
    <input type="submit" value="Save"/>
</form>
```

This is a default Django form property which is going to render any scripts needed for the form to work, in our case – Uploadcare scripts.

Using it in templates

You can construct url with all effects manually:

```
{% for photo in photos %}
    {{ photo.title }}
    {{ photo.photo.cdn_url }}-/resize/400x300/-/effect/flip/-/effect/grayscale/
{% endfor %}
```

Refer to CDN docs for more information.

How to use it in command line?

```
$ ucare -h
```

CHAPTER 3

Django Widget

Settings

Besides required `pub_key`, `secret` settings there are optional settings, for example, `widget_version` or `widget_build`:

```
UPLOADCARE = {
    'pub_key': 'demopublickey',
    'secret': 'demoprivatekey',
    'widget_version': '3.x',  // ~= 3.0 (latest)
    'widget_build': 'min',  // without jQuery
    'cdn_base': 'https://cdn.mycompany.com',
}
```

PyUploadcare takes assets from Uploadcare CDN by default, e.g.:

```
<script src="https://ucarecdn.com/widget/x.y.z/uploadcare/uploadcare.full.min.js"></script>
```

If you don't want to use hosted assets you have to turn off this feature:

```
UPLOADCARE = {
    # ...
    'use_hosted_assets': False,
}
```

In this case local assets will be used.

If you want to provide custom url for assets then you have to specify widget url:

```
UPLOADCARE = {
    # ...
    'use_hosted_assets': False,
    'widget_url': 'http://path.to/your/widget.js',
}
```

Uploadcare widget will use default upload handler url, unless you specify:

```
UPLOADCARE = {
    # ...
    'upload_base_url' = 'http://path.to/your/upload/handler',
}
```

Model Fields

As you will see, with Uploadcare, adding and working with a file field is just as simple as with a TextField. To attach Uploadcare files to a model, you can use a *FileField* or *ImageField*. These fields play by common Django rules. South migrations are supported.

Note: When you call `your_model_form.is_valid()` or call `photo.full_clean()` directly it invokes `File.store()` method automatically. In other cases you should store objects manually, e.g:

```
photo.photo_2x3 = File('a771f854-c2cb-408a-8c36-71af77811f3b')
photo.save()

photo.photo_2x3.store()
```

FileField

`FileField` does not require an uploaded file to be any certain format.

```
from django.db import models

from pyuploadcare.dj.models import FileField


class Candidate(models.Model):

    resume = FileField()
```

ImageField

`ImageField` requires an uploaded file to be an image. An optional parameter `manual_crop` enables, if specified, a manual cropping tool: your user can select a part of an image she wants to use. If its value is an empty string, the user can select any part of an image; you can also use values like `"3:4"` or `"200x300"` to get exact proportions or dimensions of resulting image. Consult widget documentation regarding setting up the manual crop:

```
from django.db import models

from pyuploadcare.dj.models import ImageField


class Candidate(models.Model):

    photo = ImageField(blank=True, manual_crop="")
```

Advanced widget options

You can pass any widget options via `FileWidget`'s attrs argument:

```
from django import forms

from pyuploadcare.dj.forms import FileWidget, ImageField
```

```
# optional. provide advanced widget options: https://uploadcare.com/documentation/widget/#configurat
class CandidateForm(forms.Form):
    photo = ImageField(widget=FileWidget(attrs={
        'data-cdn-base': 'https://cdn.super-candidates.com',
        'data-image-shrink': '1024x1024',
    }))
```

FileGroupField

`FileGroupField` allows you to upload more than one file at a time. It stores uploaded files as a group:

```
from django.db import models

from pyuploadcare.dj.models import FileGroupField


class Book(models.Model):

    pages = FileGroupField()
```

ImageGroupField

`ImageGroupField` allows you to upload more than one **image** at a time. It stores uploaded images as a group:

```
from django.db import models

from pyuploadcare.dj.models import ImageGroupField


class Gallery(models.Model):

    photos = ImageGroupField()
```

CHAPTER 4

Command Line Tool

```
usage: ucare [-h] [--version] [--pub_key PUB_KEY] [--secret SECRET]
             [--api_base API_BASE] [--upload_base UPLOAD_BASE]
             [--no_check_upload_certificate] [--no_check_api_certificate]
             [--api_version API_VERSION]
             {list_files,list_groups,get,store,delete,upload_from_url,upload,create_group,s
             ...
```

Options:

--version show program's version number and exit

--pub_key API key, if not set is read from uploadcare.ini and ~/.uploadcare config files

--secret API secret, if not set is read from uploadcare.ini and ~/.uploadcare config files

--api_base API url, can be read from uploadcare.ini and ~/.uploadcare config files. Default value is https://api.uploadcare.com/

--upload_base Upload API url, can be read from uploadcare.ini and ~/.uploadcare config files. Default value is https://upload.uploadcare.com/

--no_check_upload_certificate=False Don't check the uploading API server certificate. Can be read from uploadcare.ini and ~/.uploadcare config files.

--no_check_api_certificate=False Don't check the REST API server certificate. Can be read from uploadcare.ini and ~/.uploadcare config files.

--api_version API version, can be read from uploadcare.ini and ~/.uploadcare config files. Default value is 0.5

Sub-commands:

list_files list all files

```
usage: ucare list_files [-h] [--starting_point STARTING_POINT]
                        [--ordering ORDERING] [--limit LIMIT]
                        [--request_limit REQUEST_LIMIT]
                        [--stored {True,False,None}]
                        [--removed {True,False,None}]
```

Options:

--starting_point a starting point for filtering files

--ordering specify the way the files should be sorted

--limit=100	files to show
--request_limit=100	files per request
--stored	filter stored files
	Possible choices: True, False, None
--removed=False	filter removed files
	Possible choices: True, False, None

list_groups list all groups

```
usage: ucare list_groups [-h] [--starting_point STARTING_POINT]
                         [--ordering ORDERING] [--limit LIMIT]
                         [--request_limit REQUEST_LIMIT]
```

Options:

--starting_point	a starting point for filtering groups
--ordering	specify the way the groups should be sorted
--limit=100	group to show
--request_limit=100	groups per request

get get file info

```
usage: ucare get [-h] path
```

Positional arguments:

path	file path

store store file

```
usage: ucare store [-h] [--timeout TIMEOUT] [--wait | --nowait]
                   paths [paths ...]
```

Positional arguments:

paths	file(s) path

Options:

--timeout=5	Set wait seconds until operation completed. Default value is 5 seconds
--wait=True	Wait for operation to be completed
--nowait=True	Do not wait for operation to be completed

delete request delete

```
usage: ucare delete [-h] [--timeout TIMEOUT] [--wait | --nowait]
                    paths [paths ...]
```

Positional arguments:

paths	file(s) path

Options:

--timeout=5	Set wait seconds until operation completed. Default value is 5 seconds
--wait=True	Wait for operation to be completed

--nowait=True Do not wait for operation to be completed

upload_from_url upload file from url

```
usage: ucare upload_from_url [-h] [--store | --nostore] [--info | --noinfo]
                             [--cdnurl] [--timeout TIMEOUT]
                             [--wait | --nowait]
                             url
```

Positional arguments:

url file url

Options:

--store=False Store uploaded file

--nostore=True Do not store uploaded file

--info=False Get uploaded file info

--noinfo=True Do not get uploaded file info

--cdnurl=False Store file and get CDN url.

--timeout=30 Set wait seconds file uploading from url. Default value is 30 seconds

--wait=True Wait for upload status

--nowait=True Do not wait for upload status

upload upload file

```
usage: ucare upload [-h] [--store | --nostore] [--info | --noinfo] [--cdnurl]
                    filename
```

Positional arguments:

filename filename

Options:

--store=False Store uploaded file

--nostore=True Do not store uploaded file

--info=False Get uploaded file info

--noinfo=True Do not get uploaded file info

--cdnurl=False Store file and get CDN url.

create_group create file group

```
usage: ucare create_group [-h] paths [paths ...]
```

Positional arguments:

paths file paths

sync sync files

```
usage: ucare sync [-h] [--starting_point STARTING_POINT] [--ordering ORDERING]
                  [--limit LIMIT] [--request_limit REQUEST_LIMIT]
                  [--stored {True,False,None}] [--removed {True,False,None}]
                  [--replace] [--uuids UUIDS [UUIDS ...]] [--effects EFFECTS]
                  [path]
```

Positional arguments:

path	Local path. It can contains special patterns like: ${uuid} ${effects} ${filename} ${ext} Default is ${uuid}${ext}

Options:

--starting_point	a starting point for filtering files
--ordering	specify the way the files should be sorted
--limit=100	files to show
--request_limit=100	files per request
--stored	filter stored files
	Possible choices: True, False, None
--removed=False	filter removed files
	Possible choices: True, False, None
--replace=False	replace exists files
--uuids	list of file's uuids for sync
--effects	apply effects for synced images.Note that effects will apply to images only.For more information look at: https://uploadcare.com/documentation/cdn/ Example: –effects=resize/200x/-/rotate/90/

CHAPTER 5

Deprecated Bits

This part of the documentation contains things that eventually will be deleted.

`UPLOADCARE['widget_variant']` Django setting. Use `UPLOADCARE['widget_build']` instead.

CHAPTER 6

API Reference

Core API

You can use pyuploadcare in any Python project. At first you need assign your project keys to conf object. After that you will be able to do direct api calls or use api resources:

```
>>> import pyuploadcare
>>> pyuploadcare.conf.pub_key = '<your public key>'
>>> pyuploadcare.conf.secret = '<your private key>'
>>> f = pyuploadcare.File('6c5e9526-b0fe-4739-8975-72e8d5ee6342')
>>> f.cdn_url
https://ucarecdn.com/6c5e9526-b0fe-4739-8975-72e8d5ee6342/
```

File API Resource

class pyuploadcare.api_resources.**File**(*cdn_url_or_file_id*)

File resource for working with user-uploaded files.

It can take file UUID or group CDN url:

```
>>> file_ = File('a771f854-c2cb-408a-8c36-71af77811f3b')
>>> file_.cdn_url
https://ucarecdn.com/a771f854-c2cb-408a-8c36-71af77811f3b/
>>> print File('https://ucarecdn.com/a771f854-c2cb-408a-8c36-71af77811f3b/-/effect/flip/')
https://ucarecdn.com/a771f854-c2cb-408a-8c36-71af77811f3b/-/effect/flip/
```

uuid

File UUID [1], e.g. a771f854-c2cb-408a-8c36-71af77811f3b.

default_effects

String of default effects that is used by File.cdn_url, e.g. effect/flip/-/effect/mirror/.

class FileFromUrl(*token*)

Contains the logic around an upload from url.

It expects uploading token, for instance:

```
>>> ffu = FileFromUrl(token='a6a2db73-2aaf-4124-b2e7-039aec022e18')
>>> ffu.info()
{
    "status": "progress",
```

[1] Universally unique identifier according to RFC 4122.

```
        "done": 226038,
        "total": 452076
    }
    >>> ffu.update_info()
    {
        "status": "success",
        "file_id": "63f652fd-3f40-4b54-996c-f17dc7db5bf1",
        "is_stored": false,
        "done": 452076,
        "uuid": "63f652fd-3f40-4b54-996c-f17dc7db5bf1",
        "original_filename": "olympia.jpg",
        "is_image": true,
        "total": 452076,
        "size": 452076
    }
    >>> ffu.get_file()
    <uploadcare.File 63f652fd-3f40-4b54-996c-f17dc7db5bf1>
```

But it could be failed:

```
    >>> ffu.update_info()
    {
        "status": "error",
        "error": "some error message"
    }
```

get_file()
Returns `File` instance if upload is completed.

info()
Returns actual information about uploading as `dict`.

First time it makes API request to get information and keeps it for further using.

update_info()
Updates and returns information by requesting Uploadcare API.

wait(*timeout=30*, *interval=0.3*, *until_ready=False*)

`File.`**cdn_path**(*effects=None*)

`File.`**cdn_url**
Returns file's CDN url.

Usage example:

```
    >>> file_ = File('a771f854-c2cb-408a-8c36-71af77811f3b')
    >>> file_.cdn_url
    https://ucarecdn.com/a771f854-c2cb-408a-8c36-71af77811f3b/
```

You can set default effects:

```
    >>> file_.default_effects = 'effect/flip/-/effect/mirror/'
    >>> file_.cdn_url
    https://ucarecdn.com/a771f854-c2cb-408a-8c36-71af77811f3b/-/effect/flip/-/effect/mirror/
```

classmethod `File.`**construct_from**(*file_info*)
Constructs `File` instance from file information.

For example you have result of `/files/1921953c-5d94-4e47-ba36-c2e1dd165e1a/` API request:

```
>>> file_info = {
        # ...
        'uuid': '1921953c-5d94-4e47-ba36-c2e1dd165e1a',
        # ...
    }
>>> File.construct_from(file_info)
<uploadcare.File 1921953c-5d94-4e47-ba36-c2e1dd165e1a>
```

`File.`**`copy`**(*effects=None*, *target=None*)

Creates a File Copy on Uploadcare or Custom Storage.

File.copy method is deprecated and will be removed in 4.0.0. Please use *create_local_copy* and *create_remote_copy* instead.

Args:

- **effects:** Adds CDN image effects. If `self.default_effects` property is set effects will be combined with default effects.
- **target:** Name of a custom storage connected to your project. Uploadcare storage is used if target is absent.

`File.`**`create_local_copy`**(*effects=None*, *store=None*)

Creates a Local File Copy on Uploadcare Storage.

Args:

- **effects:** Adds CDN image effects. If `self.default_effects` property is set effects will be combined with default effects.
- **store:** If `store` option is set to False the copy of your file will be deleted in 24 hour period after the upload. Works only if *autostore* is enabled in the project.

`File.`**`create_remote_copy`**(*target*, *effects=None*, *make_public=None*, *pattern=None*)

Creates file copy in remote storage.

Args:

- **target:** Name of a custom storage connected to the project.
- **effects:** Adds CDN image effects to `self.default_effects` if any.
- **make_public:** To forbid public from accessing your files on the storage set `make_public` option to be False. Default value is None. Files have public access by default.
- **pattern:** Specify `pattern` option to set S3 object key name. Takes precedence over pattern set in project settings. If neither is specified defaults to *${uuid}/${filename}${effects}${ext}*.

For more information on each of the options above please refer to REST API docs https://uploadcare.com/documentation/rest/#file.

Following example copies a file to custom storage named `samplefs`:

```
>>> file = File('e8ebfe20-8c11-4a94-9b40-52ecad7d8d1a')
>>> file.create_remote_copy(target='samplefs',
>>>                         make_public=True,
>>>                         pattern='${uuid}/${filename}${ext}')
```

Now custom storage `samplefs` contains publicly available file with original filename billmurray.jpg in in the directory named `e8ebfe20-8c11-4a94-9b40-52ecad7d8d1a`.

`File.`**`datetime_removed`**()

Returns file's remove aware *datetime* in UTC format.

It might do API request once because it depends on `info()`.

`File.`**`datetime_stored`**`()`
Returns file's store aware *datetime* in UTC format.

It might do API request once because it depends on `info()`.

`File.`**`datetime_uploaded`**`()`
Returns file's upload aware *datetime* in UTC format.

It might do API request once because it depends on `info()`.

`File.`**`delete`**`()`
Deletes file by requesting Uploadcare API.

`File.`**`filename`**`()`
Returns original file name, e.g. `"olympia.jpg"`.

It might do API request once because it depends on `info()`.

`File.`**`info`**`()`
Returns all available file information as `dict`.

First time it makes API request to get file information and keeps it for further using.

`File.`**`is_image`**`()`
Returns `True` if the file is an image.

It might do API request once because it depends on `info()`.

`File.`**`is_ready`**`()`
Returns `True` if the file is fully uploaded on S3.

It might do API request once because it depends on `info()`.

`File.`**`is_removed`**`()`
Returns `True` if file is removed.

It might do API request once because it depends on `info()`.

`File.`**`is_stored`**`()`
Returns `True` if file is stored.

It might do API request once because it depends on `info()`.

`File.`**`mime_type`**`()`
Returns the file MIME type, e.g. `"image/png"`.

It might do API request once because it depends on `info()`.

`File.`**`size`**`()`
Returns the file size in bytes.

It might do API request once because it depends on `info()`.

`File.`**`store`**`()`
Stores file by requesting Uploadcare API.

Uploaded files do not immediately appear on Uploadcare CDN. Let's consider steps until file appears on CDN:

- first file is uploaded into https://upload.uploadcare.com/;
- after that file is available by API and its `is_public`, `is_ready` are `False`. Now you can store it;
- `is_ready` will be `True` when file will be fully uploaded on S3.

File.**update_info**()
Updates and returns file information by requesting Uploadcare API.

classmethod File.**upload**(*file_obj*, *store=None*)
Uploads a file and returns File instance.

Args:

- file_obj: file object to upload to
- **store (Optional[bool]): Should the file be automatically stored** upon upload. Defaults to None. - False - do not store file - True - store file (can result in error if autostore is disabled for project)
  - None - use project settings

Returns: File instance

classmethod File.**upload_from_url**(*url*, *store=None*, *filename=None*)
Uploads file from given url and returns FileFromUrl instance.

Args:

- url (str): URL of file to upload to
- **store (Optional[bool]): Should the file be automatically stored** upon upload. Defaults to None. - False - do not store file - True - store file (can result in error if autostore is disabled for project)
  - None - use project settings
- **filename (Optional[str]): Name of the uploaded file. If this not** specified the filename will be obtained from response headers or source URL. Defaults to None.

Returns: FileFromUrl instance

classmethod File.**upload_from_url_sync**(*url*, *timeout=30*, *interval=0.3*, *until_ready=False*, *store=None*, *filename=None*)
Uploads file from given url and returns File instance.

Args:

- url (str): URL of file to upload to
- **store (Optional[bool]): Should the file be automatically stored** upon upload. Defaults to None. - False - do not store file - True - store file (can result in error if autostore is disabled for project)
  - None - use project settings
- **filename (Optional[str]): Name of the uploaded file. If this not** specified the filename will be obtained from response headers or source URL. Defaults to None.
- **timeout (Optional[int]): seconds to wait for successful upload.** Defaults to 30.
- **interval (Optional[float]): interval between upload status checks.** Defaults to 0.3.
- **until_ready (Optional[bool]): should we wait until file is** available via CDN. Defaults to False.

Returns: File instance

Raises: `TimeoutError` if file wasn't uploaded in time

`File.`**`uuid`**

File Group API Resource

class `pyuploadcare.api_resources.`**`FileGroup`**(*cdn_url_or_group_id*)
File Group resource for working with user-uploaded group of files.

It can take group id or group CDN url:

```
>>> file_group = FileGroup('0513dda0-582f-447d-846f-096e5df9e2bb~2')
```

You can iterate `file_group` or get `File` instance by key:

```
>>> [file_ for file_ in file_group]
[<uploadcare.File 6c5e9526-b0fe-4739-8975-72e8d5ee6342>, None]
>>> file_group[0]
<uploadcare.File 6c5e9526-b0fe-4739-8975-72e8d5ee6342>
>>> len(file_group)
2
```

But slicing is not supported because `FileGroup` is immutable:

```
>>> file_group[:]
TypeError: slicing is not supported
```

If file was deleted then you will get `None`:

```
>>> file_group[1]
None
```

`id`
Group id, e.g. `0513dda0-582f-447d-846f-096e5df9e2bb~2`.

`cdn_url`
Returns group's CDN url.

Usage example:

```
>>> file_group = FileGroup('0513dda0-582f-447d-846f-096e5df9e2bb~2')
>>> file_group.cdn_url
https://ucarecdn.com/0513dda0-582f-447d-846f-096e5df9e2bb~2/
```

classmethod `construct_from`(*group_info*)
Constructs `FileGroup` instance from group information.

classmethod `create`(*files*)
Creates file group and returns `FileGroup` instance.

It expects iterable object that contains `File` instances, e.g.:

```
>>> file_1 = File('6c5e9526-b0fe-4739-8975-72e8d5ee6342')
>>> file_2 = File('a771f854-c2cb-408a-8c36-71af77811f3b')
>>> FileGroup.create((file_1, file_2))
<uploadcare.FileGroup 0513dda0-6666-447d-846f-096e5df9e2bb~2>
```

`datetime_created`()
Returns file group's create aware *datetime* in UTC format.

`datetime_stored`()
Returns file group's store aware *datetime* in UTC format.

file_cdn_urls
Returns CDN urls of all files from group without API requesting.

Usage example:

```
>>> file_group = FileGroup('0513dda0-582f-447d-846f-096e5df9e2bb~2')
>>> file_group.file_cdn_urls[0]
'https://ucarecdn.com/0513dda0-582f-447d-846f-096e5df9e2bb~2/nth/0/'
```

info()
Returns all available group information as `dict`.

First time it makes API request to get group information and keeps it for further using.

is_stored()
Returns `True` if file is stored.

It might do API request once because it depends on `info()`.

store()
Stores all group's files by requesting Uploadcare API.

Uploaded files do not immediately appear on Uploadcare CDN.

update_info()
Updates and returns group information by requesting Uploadcare API.

File List API Resource

class pyuploadcare.api_resources.**FileList**(**args*, ***kwargs*)
List of File resources.

This class provides iteration over all uploaded files.

You can specify:

- `starting_point` – a starting point for filtering files. It is reflects a `from` parameter from REST API.
- `ordering` – a string with name of the field what must be used for sorting files. The actual list of supported fields you can find in documentation: http://uploadcare.com/documentation/rest/#file-files
- `limit` – a total number of objects to be iterated. If not specified, all available objects are iterated;
- `request_limit` – a number of objects retrieved per request (page). Usually, you don't need worry about this parameter.
- `stored` – `True` to include only stored files, `False` to exclude, `None` is default, will not exclude anything;
- `removed` – `True` to include only removed files, `False` to exclude, `None` will not exclude anything. The default is `False`.

Files can't be stored and removed at the same time, such query will always return an empty set.

But files can be not stored and not removed (just uploaded files).

Usage example:

```
>>> for f in FileList(removed=None):
>>>     print(f.datetime_uploaded())
```

Count objects:

```
>>> print('Number of stored files is', FileList(stored=True).count())
```

api_url(***qs*)

base_url = u'/files/'

constructor(*file_info*)
Constructs `File` instance from file information.

For example you have result of `/files/1921953c-5d94-4e47-ba36-c2e1dd165e1a/` API request:

```
>>> file_info = {
        # ...
        'uuid': '1921953c-5d94-4e47-ba36-c2e1dd165e1a',
        # ...
    }
>>> File.construct_from(file_info)
<uploadcare.File 1921953c-5d94-4e47-ba36-c2e1dd165e1a>
```

datetime_ordering_fields = (u'', u'datetime_uploaded')

Group List API Resource

class pyuploadcare.api_resources.**GroupList**(*starting_point=None*, *ordering=None*, *limit=None*, *request_limit=None*)
List of FileGroup resources.

This class provides iteration over all groups for project. You can specify:

- `starting_point` – a starting point for filtering groups. It is reflects a `from` parameter from the REST API.
- `ordering` – a string with name of the field what must be used for sorting files. The actual list of supported fields you can find in documentation: https://uploadcare.com/documentation/rest/#group-groups
- `limit` – a total number of objects to be iterated. If not specified, all available objects are iterated;
- `request_limit` – a number of objects retrieved per request (page). Usually, you don't need worry about this parameter.

Usage example:

```
>>> from datetime import datetime, timedelta
>>> last_week = datetime.now() - timedelta(weeks=1)
>>> for f in GroupList(starting_point=last_week):
>>>     print(f.datetime_created())
```

Count objects:

```
>>> print('Number of groups is', GroupList().count())
```

base_url = u'/groups/'

constructor(*group_info*)
Constructs `FileGroup` instance from group information.

datetime_ordering_fields = (u'', u'datetime_created')

API Clients

Uploadcare REST client.

It is JSON REST request abstraction layer that is used by the `pyuploadcare.api_resources`.

`pyuploadcare.api.`**`rest_request`**(*verb*, *path*, *data=None*, *timeout=<object object>*, *retry_throttled=<object object>*)

Makes REST API request and returns response as `dict`.

It provides auth headers as well and takes settings from `conf` module.

Make sure that given `path` does not contain leading slash.

Usage example:

```
>>> rest_request('GET', 'files/?limit=10')
{
    'next': 'https://api.uploadcare.com/files/?limit=10&page=2',
    'total': 1241,
    'page': 1,
    'pages': 125,
    'per_page': 10,
    'previous': None,
    'results': [
        # ...
        {
            # ...
            'uuid': 1921953c-5d94-4e47-ba36-c2e1dd165e1a,
            # ...
        },
        # ...
    ]
}
```

`pyuploadcare.api.`**`uploading_request`**(*verb*, *path*, *data=None*, *files=None*, *timeout=<object object>*)

Makes Uploading API request and returns response as `dict`.

It takes settings from `conf` module.

Make sure that given `path` does not contain leading slash.

Usage example:

```
>>> file_obj = open('photo.jpg', 'rb')
>>> uploading_request('POST', 'base/', files={'file': file_obj})
{
    'file': '9b9f4483-77b8-40ae-a198-272ba6280004'
}
>>> File('9b9f4483-77b8-40ae-a198-272ba6280004')
```

Exceptions

exception `pyuploadcare.exceptions.`**`APIConnectionError`**(*data=u''*, **args*, ***kwargs*)

Network communication with Uploadcare errors.

exception `pyuploadcare.exceptions.`**`APIError`**(*data=u''*, **args*, ***kwargs*)

API errors, e.g. bad json.

exception pyuploadcare.exceptions.**AuthenticationError**(*data=u''*, **args*, ***kwargs*)
Authentication with Uploadcare's API errors.

exception pyuploadcare.exceptions.**InvalidParamError**(*data=u''*, **args*, ***kwargs*)
Invalid parameters errors, e.g. invalid UUID

exception pyuploadcare.exceptions.**InvalidRequestError**(*data=u''*, **args*, ***kwargs*)
Invalid service parameters errors, e.g status 404

exception pyuploadcare.exceptions.**ThrottledRequestError**(*response*)
Raised when request was throttled.

exception pyuploadcare.exceptions.**TimeoutError**(*data=u''*, **args*, ***kwargs*)
Timed out errors.

It raises when user wants to wait the result of api requests, e.g.:

```
$ ucare store --wait 6c5e9526-b0fe-4739-8975-72e8d5ee6342
```

exception pyuploadcare.exceptions.**UploadError**(*data=u''*, **args*, ***kwargs*)
Upload errors.

It raises when user wants to wait the result of:

```
$ ucare upload_from_url --wait http://path.to/file.jpg
```

exception pyuploadcare.exceptions.**UploadcareException**(*data=u''*, **args*, ***kwargs*)
Base exception class of library.

Django Widget API

Model Fields

Form Fields

Command Line Tool API

pyuploadcare.ucare_cli.**create_group**(*arg_namespace*)

pyuploadcare.ucare_cli.**delete_files**(*arg_namespace*)

pyuploadcare.ucare_cli.**get_file**(*arg_namespace*)

pyuploadcare.ucare_cli.**list_files**(*arg_namespace*)

pyuploadcare.ucare_cli.**list_groups**(*arg_namespace*)

pyuploadcare.ucare_cli.**load_config_from_args**(*arg_namespace*)

pyuploadcare.ucare_cli.**load_config_from_file**(*filename*)

pyuploadcare.ucare_cli.**main**(*arg_namespace=None*, *config_file_names=(u'~/.uploadcare'*, *u'uploadcare.ini')*)

pyuploadcare.ucare_cli.**store_files**(*arg_namespace*)

pyuploadcare.ucare_cli.**ucare_argparser**()

pyuploadcare.ucare_cli.**upload**(*arg_namespace*)

pyuploadcare.ucare_cli.**upload_from_url**(*arg_namespace*)

CHAPTER 7

Indices and tables

- genindex
- modindex
- search

p

A

B

C

D

F

G

I

L

M

P

R

S

T

U

W