
django-downloadview Documentation

Release 1.1

Benoît Bryon

October 28, 2013

Contents

Django-DownloadView provides generic views to make Django serve files.

It can serve files from models, storages, local filesystem, arbitrary URL... and even generated files.

For increased performances, it can delegate the actual streaming to a reverse proxy, via mechanisms such as Nginx's X-Accel.

Example

In some `urls.py`, serve files managed in a model:

```
from django.conf.urls import url, url_patterns
from django_downloadview import ObjectDownloadView
from demoproject.download.models import Document # A model with a FileField

# ObjectDownloadView inherits from django.views.generic.BaseDetailView.
download = ObjectDownloadView.as_view(model=Document, file_field='file')

url_patterns = (
    url('^download/(?P<slug>[A-Za-z0-9_-]+)/$', download, name='download'),
)
```

More examples in the *demo project*!

Views

Several views are provided to cover frequent use cases:

- `ObjectDownloadView` to serve some file field of a model.
- `StorageDownloadView` to serve files in a storage, without models.
- `PathDownloadView` to serve files located on local filesystem, without storage.
- `HTTPDownloadView` to proxy some arbitrary URL.
- `VirtualDownloadView` to serve in-memory or generated files.

See *Download views* for details.

Then get increased performances with *Optimizations*.

Ressources

- Documentation: <http://django-downloadview.readthedocs.org>
- PyPI page: <http://pypi.python.org/pypi/django-downloadview>
- Code repository: <https://github.com/benoitbryon/django-downloadview>
- Bugtracker: <https://github.com/benoitbryon/django-downloadview/issues>
- Continuous integration: <https://travis-ci.org/benoitbryon/django-downloadview>

Contents

4.1 Demo project

The `demo/` folder holds a demo project to illustrate `django-downloadview` usage.

4.1.1 Browse demo code online

See `demo` folder in project's repository ¹.

4.1.2 Deploy the demo

System requirements:

- `Python` ² version 2.6 or 2.7, available as `python` command.

Note: You may use `Virtualenv` ³ to make sure the active `python` is the right one.

- `make` and `wget` to use the provided `Makefile`.

Execute:

```
git clone git@github.com:benoitbryon/django-downloadview.git
cd django-downloadview/
make demo
```

It installs and runs the demo server on localhost, port 8000. So have a look at <http://localhost:8000/>

Note: If you cannot execute the `Makefile`, read it and adapt the few commands it contains to your needs.

Browse and use `demo/demoproject/` as a sandbox.

¹ <https://github.com/benoitbryon/django-downloadview/tree/master/demo/demoproject/>

² <http://python.org>

³ <http://virtualenv.org>

4.1.3 Base example provided in the demo

In the “demoproject” project, there is an application called “download”.

demo/demoproject/settings.py:

```
STATIC_URL = '/static/'

# Applications.
INSTALLED_APPS = (
    # Standard Django applications.
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.sites',
    'django.contrib.messages',
    'django.contrib.staticfiles',
    # The actual django-downloadview demo.
    'demoproject',
    'demoproject.download', # Sample standard download views.
    'demoproject.nginx',    # Sample optimizations for Nginx.
    # For test purposes. The demo project is part of django-downloadview
```

This application holds a Document model.

demo/demoproject/download/models.py:

```
from django.db import models

class Document(models.Model):
    """A sample model with a FileField."""
    slug = models.SlugField(verbose_name='slug')
    file = models.FileField(verbose_name='file', upload_to='document')
```

Note: The storage is the default one, i.e. it uses settings.MEDIA_ROOT. Combined to this upload_to configuration, files for Document model live in var/media/document/ folder, relative to your django-downloadview clone root.

There is a download view named “download_document” for this model:

demo/demoproject/download/urls.py:

```
# coding=utf8
"""URL mapping."""
from django.conf.urls import patterns, url

urlpatterns = patterns(
    'demoproject.download.views',
    # Model-based downloads.
    url(r'^document/(?P<slug>[a-zA-Z0-9_-]+)/$',
        'download_document',
        name='document'),
    # Storage-based downloads.
    url(r'^storage/(?P<path>[a-zA-Z0-9_-]+\.[a-zA-Z0-9]{1,4})$',
        'download_fixture_from_storage',
        name='fixture_from_storage'),
```

```
# Path-based downloads.
url(r'^hello-world\.txt$',
    'download_hello_world',
    name='hello_world'),
url(r'^path/(?P<path>[a-zA-Z0-9_-]+\.[a-zA-Z0-9]{1,4})$',
    'download_fixture_from_path',
    name='fixture_from_path'),
# URL-based downloads.
url(r'^http/readme\.txt$',
    'download_http_hello_world',
    name='http_hello_world'),
# Generated downloads.
url(r'^generated/hello-world\.txt$',
    'download_generated_hello_world',
    name='generated_hello_world'),
)
```

As is, Django is to serve the files, i.e. load chunks into memory and stream them.

4.1.4 References

4.2 Installation

This project is open-source, published under BSD license. See *License* for details.

If you want to install a development environment, you should go to *Contributing to the project* documentation.

Install the package with your favorite Python installer. As an example, with pip:

```
pip install django-downloadview
```

Note: Since version 1.1, django-downloadview requires Django>=1.5, which provides StreamingHttpResponse.

There is no need to register this application in your Django's `INSTALLED_APPS` setting.

Next, you'll have to setup some download view(s). See *demo project* for examples, and *API documentation*.

Optionally, you may setup additional *server optimizations*.

4.3 Download views

This section contains narrative overview about class-based views provided by django-downloadview.

By default, all of those views would stream the file to the client. But keep in mind that you can setup *Optimizations* to delegate actual streaming to a reverse proxy.

4.3.1 ObjectDownloadView

The `django_downloadview.views.ObjectDownloadView` class-based view allows you to **serve files given a model with some file fields** such as `FileField` or `ImageField`.

Use this view anywhere you could use Django's builtin `ObjectDetailView`.

Some options allow you to store file metadata (size, content-type, ...) in the model, as deserialized fields.

4.3.2 StorageDownloadView

The `django_downloadview.views.StorageDownloadView` class-based view allows you to **serve files given a storage and a path**.

Use this view when you manage files in a storage (which is a good practice), unrelated to a model.

4.3.3 PathDownloadView

The `django_downloadview.views.PathDownloadView` class-based view allows you to **serve files given an absolute path on local filesystem**.

Two main use cases:

- as a shortcut. This dead-simple view is straight to call, so you can use it to simplify code in more complex views, provided you have an absolute path to a local file.
- override. Extend `django_downloadview.views.PathDownloadView` and override `django_downloadview.views.PathDownloadView:get_path()`.

4.3.4 HTTPDownloadView

The `django_downloadview.views.HTTPDownloadView` class-based view allows you to **serve files given an URL**. That URL is supposed to be downloadable from the Django server.

Use it when you want to setup a proxy to remote files:

- the Django view filters input and computes target URL.
- if you setup optimizations, Django itself doesn't proxy the file,
- but, as a fallback, Django uses [requests](https://pypi.python.org/pypi/requests)⁴ to proxy the file.

Extend `django_downloadview.views.HTTPDownloadView` then override `django_downloadview.views.HTTPDownloadView:get_url()`.

4.3.5 VirtualDownloadView

The `django_downloadview.views.VirtualDownloadView` class-based view allows you to **serve files that don't live on disk**.

Use it when you want to stream a file which content is dynamically generated or which lives in memory.

References

4.4 Optimizations

Some reverse proxies allow applications to delegate actual download to the proxy:

- with Django, manage permissions, generate files...
- let the reverse proxy serve the file.

As a result, you get increased performance: reverse proxies are more efficient than Django at serving static files.

⁴ <https://pypi.python.org/pypi/requests>

4.4.1 Nginx

If you serve Django behind Nginx, then you can delegate the file download service to Nginx and get increased performance:

- lower resources used by Python/Django workers ;
- faster download.

See [Nginx X-accel documentation](#)⁵ for details.

Configure some download view

Let's start in the situation described in the *demo application*:

- a project “demoproject”
- an application “demoproject.download”
- a `django_downloadview.views.ObjectDownloadView` view serves files of a “Document” model.

We are to make it more efficient with Nginx.

Note: Examples below are taken from the *demo project*.

Write tests

Use `django_downloadview.nginx.assert_x_accel_redirect()` function as a shortcut in your tests.

demo/demoproject/nginx/tests.py:

```
"""Test suite for demoproject.nginx."""
from django.core.files import File
from django.core.urlresolvers import reverse_lazy as reverse

from django_downloadview.nginx import assert_x_accel_redirect
from django_downloadview.test import temporary_media_root

from demoproject.download.models import Document
from demoproject.download.tests import DownloadTestCase

class XAccelRedirectDecoratorTestCase(DownloadTestCase):
    @temporary_media_root()
    def test_response(self):
        """'download_document_nginx' view returns a valid X-Accel response."""
        document = Document.objects.create(
            slug='hello-world',
            file=File(open(self.files['hello-world.txt'])),
        )
        download_url = reverse('download_document_nginx',
                               kwargs={'slug': 'hello-world'})
        response = self.client.get(download_url)
        self.assertEqual(response.status_code, 200)
        # Validation shortcut: assert_x_accel_redirect.
        assert_x_accel_redirect(
```

⁵ <http://wiki.nginx.org/X-accel>

```

        self,
        response,
        content_type="text/plain; charset=utf-8",
        charset="utf-8",
        basename="hello-world.txt",
        redirect_url="/download-optimized/document/hello-world.txt",
        expires=None,
        with_buffering=None,
        limit_rate=None)
    # Check some more items, because this test is part of
    # django-downloadview tests.
    self.assertFalse('ContentEncoding' in response)
    self.assertEqual(response['Content-Disposition'],
                      'attachment; filename=hello-world.txt')

```

Right now, this test should fail, since you haven't implemented the view yet.

Setup Django

At the end of this setup, the test should pass, but you still have to setup Nginx!

You have two options: global setup with a middleware, or per-view setup with decorators.

Global delegation, with XAccelRedirectMiddleware

If you want to delegate all file downloads to Nginx, then use `django_downloadview.nginx.XAccelRedirectMiddleware`.

Register it in your settings:

```

MIDDLEWARE_CLASSES = (
    # ...
    'django_downloadview.nginx.XAccelRedirectMiddleware',
    # ...
)

```

Setup the middleware:

```

NGINX_DOWNLOAD_MIDDLEWARE_MEDIA_ROOT = MEDIA_ROOT # Could be elsewhere.
NGINX_DOWNLOAD_MIDDLEWARE_MEDIA_URL = '/proxied-download'

```

Optionally fine-tune the middleware. Default values are None, which means “use Nginx’s defaults”.

```

NGINX_DOWNLOAD_MIDDLEWARE_EXPIRES = False # Force no expiration.
NGINX_DOWNLOAD_MIDDLEWARE_WITH_BUFFERING = False # Force buffering off.
NGINX_DOWNLOAD_MIDDLEWARE_LIMIT_RATE = False # Force limit rate off.

```

Local delegation, with x_accel_redirect decorator

If you want to delegate file downloads to Nginx on a per-view basis, then use `django_downloadview.nginx.x_accel_redirect()` decorator.

demo/demoproject/nginx/views.py:

```

"""Views."""
from django_downloadview.nginx import x_accel_redirect

```

```
from demoproject.download import views
```

```
download_document_nginx = x_accel_redirect(
    views.download_document,
    source_dir='/var/www/files',
    destination_url='/download-optimized')
```

And use it in som URL conf, as an example in demo/demoproject/nginx/urls.py:

```
"""URL mapping."""
from django.conf.urls import patterns, url

urlpatterns = patterns('demoproject.nginx.views',
    url(r'^document-nginx/(?P<slug>[a-zA-Z0-9_-]+)/$',
        'download_document_nginx', name='download_document_nginx'),
)
```

Note: In real life, you'd certainly want to replace the “download_document” view instead of registering a new view.

Setup Nginx

See [Nginx X-accel documentation](#)¹ for details.

Here is what you could have in /etc/nginx/sites-available/default:

```
charset utf-8;

# Django-powered service.
upstream frontend {
    server 127.0.0.1:8000 fail_timeout=0;
}

server {
    listen 80 default;

    # File-download proxy.
    #
    # Will serve /var/www/files/myfile.tar.gz when passed URI
    # like /optimized-download/myfile.tar.gz
    #
    # See http://wiki.nginx.org/X-accel
    # and https://github.com/benoitbryon/django-downloadview
    location /proxied-download {
        internal;
        # Location to files on disk.
        # See Django's settings.NGINX_DOWNLOAD_MIDDLEWARE_MEDIA_ROOT
        alias /var/www/files/;
    }

    # Proxy to Django-powered frontend.
    location / {
        proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
        proxy_set_header Host $http_host;
        proxy_redirect off;
    }
}
```

```
        proxy_pass http://frontend;
    }
}
```

... where specific configuration is the `location /optimized-download` section.

Note: `/proxied-download` is not available for the client, i.e. users won't be able to download files via `/optimized-download/<filename>`.

Warning: Make sure Nginx can read the files to download! Check permissions.

Common issues

Unknown charset "utf-8" to override

Add `charset utf-8;` in your nginx configuration file.

`open() "path/to/something" failed (2: No such file or directory)`

Check your `settings.NGINX_DOWNLOAD_MIDDLEWARE_MEDIA_ROOT` in Django configuration VS alias in nginx configuration: in a standard configuration, they should be equal.

References

Currently, only nginx's X-Accel⁶ is supported, but contributions are welcome⁷!

4.4.2 How does it work?

The feature is inspired by Django's `TemplateResponse`⁸: the download views return some `django_downloadview.response.DownloadResponse` instance. Such a response doesn't contain file data.

By default, at the end of Django's request/response handling, Django is to iterate over the `content` attribute of the response. In a `DownloadResponse`, this `content` attribute is a file wrapper.

It means that decorators and middlewares are given an opportunity to capture the `DownloadResponse` before the content of the file is loaded into memory. As an example, `django_downloadview.nginx.XAccelRedirectMiddleware` replaces `DownloadResponse` instance by some `django_downloadview.nginx.XAccelRedirectResponse`.

References

4.5 Testing download views

This project includes shortcuts to simplify testing.

⁶ <http://wiki.nginx.org/X-accel>

⁷ <https://github.com/benoitbryon/django-downloadview/issues?labels=optimizations>

⁸ <https://docs.djangoproject.com/en/1.5/ref/template-response/>

See `django_downloadview.test` for details.

4.6 API

Here is API documentation, generated from code.

4.6.1 django_downloadview

django_downloadview Package

django_downloadview Package

django-downloadview provides generic download views for Django.

decorators Module

View decorators.

See also decorators provided by server-specific modules, such as `django_downloadview.nginx.x_accel_redirect()`.

class `django_downloadview.decorators.DownloadDecorator` (*middleware_factory*)

Bases: `object`

View decorator factory to apply middleware to `view_func` response.

Middleware instance is built from `middleware_factory` with `*args` and `**kwargs`. Middleware factory is typically a class, such as some `django_downloadview.middlewares.XAccelMiddleware` subclass.

Response is built from view, then the middleware's `process_response` method is applied on response.

files Module

File wrappers for use as exchange data between views and responses.

class `django_downloadview.files.HTTPFile` (*request_factory=<function get at 0x28bc7d0>*, *url=''*, *name=u''*, ***kwargs*)

Bases: `django.core.files.base.File`

Wrapper for files that live on remote HTTP servers.

Acts as a proxy.

Uses <https://pypi.python.org/pypi/requests>.

Always sets “stream=True” in requests kwargs.

file

request

size

Return the total size, in bytes, of the file.

Reads response's “content-length” header.

class `django_downloadview.files.StorageFile` (*storage, name, file=None*)

Bases: `django.core.files.base.File`

A file in a Django storage.

This class looks like `django.db.models.fields.files.FieldFile`, but unrelated to model instance.

accessed_time

Return the last accessed time (as datetime object) of the file.

Proxy to `self.storage.accessed_time(self.name)`.

created_time

Return the creation time (as datetime object) of the file.

Proxy to `self.storage.created_time(self.name)`.

delete()

Delete the specified file from the storage system.

Proxy to `self.storage.delete(self.name)`.

exists()

Return True if file already exists in the storage system.

If False, then the name is available for a new file.

file

Required by `django.core.files.utils.FileProxy`.

modified_time

Return the last modification time (as datetime object) of the file.

Proxy to `self.storage.modified_time(self.name)`.

open (*mode='rb'*)

Retrieves the specified file from storage and return `open()` result.

Proxy to `self.storage.open(self.name, mode)`.

path

Return a local filesystem path which is suitable for `open()`.

Proxy to `self.storage.path(self.name)`.

May raise `NotImplementedError` if storage doesn't support file access with Python's built-in `open()` function

save (*content*)

Saves new content to the file.

Proxy to `self.storage.save(self.name)`.

The content should be a proper File object, ready to be read from the beginning.

size

Return the total size, in bytes, of the file.

Proxy to `self.storage.size(self.name)`.

url

Return an absolute URL where the file's contents can be accessed.

Proxy to `self.storage.url(self.name)`.

```
class django_downloadview.files.VirtualFile (file=None, name=u'', url='', size=None)
    Bases: django.core.files.base.File

    Wrapper for files that live in memory.

    size
```

middlewares Module

Base material for download middlewares.

```
class django_downloadview.middlewares.BaseDownloadMiddleware
    Bases: object

    Base (abstract) Django middleware that handles download responses.

    Subclasses must implement process_download_response() method.

    is_download_response (response)
        Return True if response can be considered as a file download.

        By default, this method uses django_downloadview.response.is_download_response().
        Override this method if you want a different behaviour.

    process_download_response (request, response)
        Handle file download response.

    process_response (request, response)
        Call process_download_response() if response is download.
```

nginx Module

Optimizations for Nginx.

See also [Nginx X-accel documentation](#) and *narrative documentation about Nginx optimizations*.

```
class django_downloadview.nginx.BaseXAccelRedirectMiddleware (source_dir=None,
                                                            source_url=None,
                                                            destination_url=None,
                                                            expires=None,
                                                            with_buffering=None,
                                                            limit_rate=None,
                                                            media_root=None,
                                                            media_url=None)

    Bases: django_downloadview.middlewares.BaseDownloadMiddleware

    Configurable middleware, for use in decorators or in global middlewares.

    Standard Django middlewares are configured globally via settings. Instances of this class
    are to be configured individually. It makes it possible to use this class as the factory in
    django_downloadview.decorators.DownloadDecorator.

    get_redirect_url (response)
        Return redirect URL for file wrapped into response.

    is_download_response (response)
        Return True for DownloadResponse, except for “virtual” files.

        This implementation can’t handle files that live in memory or which are to be dynamically iterated over.
        So, we capture only responses whose file attribute have either an URL or a file name.
```

process_download_response (*request, response*)

Replace DownloadResponse instances by NginxDownloadResponse ones.

`django_downloadview.nginx.DEFAULT_DESTINATION_URL = None`

Default value for settings.NGINX_DOWNLOAD_MIDDLEWARE_DESTINATION_URL.

`django_downloadview.nginx.DEFAULT_EXPIRES = None`

Default value for X-Accel-Limit-Expires header. Also default value for settings.NGINX_DOWNLOAD_MIDDLEWARE_EXPIRES.

See <http://wiki.nginx.org/X-accел#X-Accel-Limit-Expires>

Default value is None, which means “let Nginx choose”, i.e. use Nginx defaults or specific configuration.

If set to False, Nginx buffering is disabled. Else, it indicates the expiration delay, in seconds.

`django_downloadview.nginx.DEFAULT_LIMIT_RATE = None`

Default value for X-Accel-Limit-Rate header. Also default value for settings.NGINX_DOWNLOAD_MIDDLEWARE_LIMIT_RATE.

See <http://wiki.nginx.org/X-accел#X-Accel-Limit-Rate>

Default value is None, which means “let Nginx choose”, i.e. use Nginx defaults or specific configuration.

If set to False, Nginx limit rate is disabled. Else, it indicates the limit rate in bytes.

`django_downloadview.nginx.DEFAULT_SOURCE_DIR = “`

Default value for settings.NGINX_DOWNLOAD_MIDDLEWARE_SOURCE_DIR.

`django_downloadview.nginx.DEFAULT_SOURCE_URL = “`

Default value for settings.NGINX_DOWNLOAD_MIDDLEWARE_SOURCE_URL.

`django_downloadview.nginx.DEFAULT_WITH_BUFFERING = None`

Default value for X-Accel-Buffering header. Also default value for settings.NGINX_DOWNLOAD_MIDDLEWARE_WITH_BUFFERING.

See <http://wiki.nginx.org/X-accел#X-Accel-Limit-Buffering>

Default value is None, which means “let Nginx choose”, i.e. use Nginx defaults or specific configuration.

If set to False, Nginx buffering is disabled. If set to True, Nginx buffering is enabled.

class `django_downloadview.nginx.XAccelRedirectMiddleware`

Bases: `django_downloadview.nginx.BaseXAccelRedirectMiddleware`

Apply X-Accel-Redirect globally, via Django settings.

Available settings are:

NGINX_DOWNLOAD_MIDDLEWARE_SOURCE_URL: The string at the beginning of URLs to replace with `NGINX_DOWNLOAD_MIDDLEWARE_DESTINATION_URL`. If None, then URLs aren’t captured. Defaults to `settings.MEDIA_URL`.

NGINX_DOWNLOAD_MIDDLEWARE_SOURCE_DIR: The string at the beginning of filenames (path) to replace with `NGINX_DOWNLOAD_MIDDLEWARE_DESTINATION_URL`. If None, then filenames aren’t captured. Defaults to `settings.MEDIA_ROOT`.

NGINX_DOWNLOAD_MIDDLEWARE_DESTINATION_URL: The base URL where requests are proxied to. If None an `ImproperlyConfigured` exception is raised.

Note: The following settings are deprecated since version 1.1. URLs can be used as redirection source since 1.1, and then “`MEDIA_ROOT`” and “`MEDIA_URL`” became too confuse.

NGINX_DOWNLOAD_MIDDLEWARE_MEDIA_ROOT: Replaced by `NGINX_DOWNLOAD_MIDDLEWARE_SOURCE_D`

NGINX_DOWNLOAD_MIDDLEWARE_MEDIA_URL: Replaced by `NGINX_DOWNLOAD_MIDDLEWARE_DESTINATION`

```
class django_downloadview.nginx.XAccelRedirectResponse (redirect_url,          con-
                                                         tent_type,          base-
                                                         name=None,    expires=None,
                                                         with_buffering=None,
                                                         limit_rate=None)
```

Bases: `django.http.response.HttpResponse`

Http response that delegates serving file to Nginx.

```
class django_downloadview.nginx.XAccelRedirectValidator
```

Bases: `object`

Utility class to validate `XAccelRedirectResponse` instances.

See also `assert_x_accel_redirect()` shortcut function.

```
assert_basename (test_case, response, value)
```

```
assert_charset (test_case, response, value)
```

```
assert_content_type (test_case, response, value)
```

```
assert_expires (test_case, response, value)
```

```
assert_limit_rate (test_case, response, value)
```

```
assert_redirect_url (test_case, response, value)
```

```
assert_with_buffering (test_case, response, value)
```

```
assert_x_accel_redirect_response (test_case, response)
```

```
django_downloadview.nginx.assert_x_accel_redirect (test_case, response, **assertions)
```

Make `test_case` assert that response is a `XAccelRedirectResponse`.

Optional assertions dictionary can be used to check additional items:

- `basename`: the basename of the file in the response.
- `content_type`: the value of “Content-Type” header.
- `redirect_url`: the value of “X-Accel-Redirect” header.
- `charset`: the value of X-Accel-Charset header.
- `with_buffering`: the value of X-Accel-Buffering header. If `False`, then makes sure that the header disables buffering. If `None`, then makes sure that the header is not set.
- `expires`: the value of X-Accel-Expires header. If `False`, then makes sure that the header disables expiration. If `None`, then makes sure that the header is not set.
- `limit_rate`: the value of X-Accel-Limit-Rate header. If `False`, then makes sure that the header disables limit rate. If `None`, then makes sure that the header is not set.

```
django_downloadview.nginx.x_accel_redirect = <django_downloadview.decorators.DownloadDecorator object at
```

Apply `BaseXAccelRedirectMiddleware` to `view_func` response.

Proxies additional arguments (`*args`, `**kwargs`) to `BaseXAccelRedirectMiddleware` constructor (`expires`, `with_buffering`, and `limit_rate`).

response Module

HttpResponse subclasses.

```
class django_downloadview.response.DownloadResponse (file_instance, attachment=True,  
                                                    basename=None, status=200,  
                                                    content_type=None)
```

Bases: `django.http.response.StreamingHttpResponse`

File download response.

`content` attribute is supposed to be a file object wrapper, which makes this response “lazy”.

default_headers

Return dictionary of automatically-computed headers.

Uses an internal `_default_headers` cache. Default values are computed if only cache hasn’t been set.

get_basename()

Return basename.

get_charset()

Return the charset of the file to serve.

get_content_type()

Return a suitable “Content-Type” header for `self.file`.

get_encoding()

Return encoding of the file to serve.

get_mime_type()

Return mime-type of the file.

items()

Return iterable of (header, value).

This method is called by http handlers just before WSGI’s `start_response()` is called... but it is not called by `django.test.ClientHandler!` :(

```
django_downloadview.response.is_download_response (response)
```

Return True if response is a download response.

Current implementation returns True if response is an instance of `django_downloadview.response.DownloadResponse`.

test Module

Testing utilities.

```
class django_downloadview.test.DownloadResponseValidator
```

Bases: `object`

Utility class to validate `DownloadResponse` instances.

```
assert_attachment (test_case, response, value)
```

```
assert_basename (test_case, response, value)
```

```
assert_content (test_case, response, value)
```

```
assert_content_type (test_case, response, value)
```

```
assert_download_response (test_case, response)
```

assert_mime_type (*test_case, response, value*)

`django_downloadview.test.assert_download_response` (*test_case, response, **assertions*)
 Make *test_case* assert that *response* is a `DownloadResponse`.

Optional assertions dictionary can be used to check additional items:

- basename**: the basename of the file in the response.
- content_type**: the value of “Content-Type” header.
- charset**: the value of X-Accel-Charset header.
- content**: the content of the file to be downloaded.

class `django_downloadview.test.temporary_media_root` (***kwargs*)

Bases: `django.test.utils.override_settings`

Context manager or decorator to override settings.MEDIA_ROOT.

```
>>> from django_downloadview.test import temporary_media_root
>>> from django.conf import settings
>>> global_media_root = settings.MEDIA_ROOT
>>> with temporary_media_root():
...     global_media_root == settings.MEDIA_ROOT
False
>>> global_media_root == settings.MEDIA_ROOT
True

>>> @temporary_media_root()
... def use_temporary_media_root():
...     return settings.MEDIA_ROOT
>>> tmp_media_root = use_temporary_media_root()
>>> global_media_root == tmp_media_root
False
>>> global_media_root == settings.MEDIA_ROOT
True
```

disable ()

Remove directory settings.MEDIA_ROOT then restore original setting.

enable ()

Create a temporary directory and use it to override settings.MEDIA_ROOT.

utils Module

Utility functions.

`django_downloadview.utils.content_type_to_charset` (*content_type*)

Return charset part of content-type header.

```
>>> from django_downloadview.utils import content_type_to_charset
>>> content_type_to_charset('text/html; charset=utf-8')
'utf-8'
```

views Module

Views.

```
class django_downloadview.views.BaseDownloadView (**kwargs)
    Bases: django_downloadview.views.DownloadMixin, django.views.generic.base.View
```

```
    get (request, *args, **kwargs)
        Handle GET requests: stream a file.
```

```
class django_downloadview.views.DownloadMixin
    Bases: object
```

Placeholders and base implementation to create file download views.

The `get_file()` method is a placeholder, which raises `NotImplementedError` in base implementation.

The other methods provide an implementation that use the file object returned by `get_file()`, supposing the file is hosted on the local filesystem.

You may override one or several methods to adapt the implementation to your use case.

```
attachment = True
    Whether to return the response as attachment or not.
```

```
basename = None
    Client-side filename, if only file is returned as attachment.
```

```
get_basename ()
```

```
get_file ()
    Return a file wrapper instance.
```

```
render_to_response (*args, **kwargs)
    Returns a response with a file as attachment.
```

```
response_class
    Response class to be used in render_to_response().

    alias of DownloadResponse
```

```
class django_downloadview.views.HTTPDownloadView (**kwargs)
    Bases: django_downloadview.views.BaseDownloadView
```

Proxy files that live on remote servers.

```
get_file ()
    Return wrapper which has an url attribute.
```

```
get_request_factory ()
    Return request factory to perform actual HTTP request.
```

```
get_request_kwargs ()
    Return keyword arguments for use with request factory.
```

```
get_url ()
    Return remote file URL (the one we are proxying).
```

```
request_kwargs = {}
    Additional keyword arguments for request handler.
```

```
url = u''
    URL to download (the one we are proxying).
```

```
class django_downloadview.views.ObjectDownloadView (**kwargs)
    Bases: django_downloadview.views.DownloadMixin, django.views.generic.detail.BaseDetailView
```

Download view for models which contain a `FileField`.

This class extends `BaseDetailView`, so you can use its arguments to target the instance to operate on: `slug`, `slug_kwarg`, `model`, `queryset`... See Django's `DetailView` reference for details.

In addition to `BaseDetailView` arguments, you can set arguments related to the file to be downloaded.

The main one is `file_field`.

The other arguments are provided for convenience, in case your model holds some (deserialized) metadata about the file, such as its `basename`, its `modification time`, its `MIME type`... These fields may be particularly handy if your file storage is not the local filesystem.

`basename_field = None`

Optional name of the model's attribute which contains the `basename`.

`charset_field = None`

Optional name of the model's attribute which contains the `charset`.

`encoding_field = None`

Optional name of the model's attribute which contains the `encoding`.

`file_field = 'file'`

Name of the model's attribute which contains the file to be streamed. Typically the name of a `FileField`.

`get_basename()`

Return client-side filename.

`get_file()`

Return `FieldFile` instance.

`mime_type_field = None`

Optional name of the model's attribute which contains the `MIME type`.

`modification_time_field = None`

Optional name of the model's attribute which contains the `modification`

`size_field = None`

Optional name of the model's attribute which contains the `size`.

`class django_downloadview.views.PathDownloadView(kwargs)`**

Bases: `django_downloadview.views.BaseDownloadView`

Serve a file using filename.

`get_file()`

Use path to return wrapper around file to serve.

`get_path()`

Return actual path of the file to serve.

Default implementation simply returns view's `path`.

Override this method if you want custom implementation. As an example, `path` could be relative and your custom `get_path()` implementation makes it absolute.

`path = None`

Server-side name (including path) of the file to serve.

Filename is supposed to be an absolute filename of a file located on the local filesystem.

`path_url_kwarg = 'path'`

Name of the URL argument that contains `path`.

`class django_downloadview.views.StorageDownloadView(kwargs)`**

Bases: `django_downloadview.views.PathDownloadView`

Serve a file using storage and filename.

get_file()
Use path and storage to return wrapper around file to serve.

get_path()
Return path of the file to serve, relative to storage.

Default implementation simply returns view's path.

Override this method if you want custom implementation.

path = None
Path to the file to serve relative to storage.

storage = <django.core.files.storage.DefaultStorage object at 0x28ba6d0>
Storage the file to serve belongs to.

class django_downloadview.views.**VirtualDownloadView** (**kwargs)
Bases: django_downloadview.views.BaseDownloadView

Serve not-on-disk or generated-on-the-fly file.

Use this class to serve StringIO files.

Override the `get_file()` method to customize file wrapper.

get_file()
Return wrapper.

4.7 About django-downloadview

4.7.1 Alternatives and related projects

This document presents other projects that provide similar or complementary functionalities. It focuses on differences with django-downloadview.

Django's static file view

Django has a builtin static file view ⁹. It can stream files. As explained in Django documentation, it is designed for development purposes. For production, static files'd better be served by some optimized server.

Django-downloadview can replace Django's builtin static file view:

- perform actions with Django when receiving download requests: check permissions, generate files, gzip, logging, signals...
- delegate actual download to a reverse proxy for increased performance.
- disable optimization middlewares or decorators in development, if you want to serve files with Django.

django-sendfile

django-sendfile ¹⁰ is a wrapper around web-server specific methods for sending files to web clients.

⁹ <https://docs.djangoproject.com/en/1.4/ref/contrib/staticfiles/#static-file-development-view>

¹⁰ <http://pypi.python.org/pypi/django-sendfile>

API is made of a single `sendfile()` function, which returns a download response. The download response type depends on the chosen backend, which could be Django, Lighttpd's X-Sendfile, Nginx's X-Accel...

It seems that `django-senfile` main focus is simplicity: you call the `sendfile()` method inside your views.

Django-downloadview main focus is reusability: you configure (or override) class-based views depending on the use case.

As of 2012-04-11, `django-sendfile` (version 0.3.2) seems quite popular and may be a good alternative **provided you serve files that live in local filesystem**, because the `sendfile()` method only accepts filenames relative to local filesystem (i.e. using `os.path.exists`).

Django-downloadview (since version 1.1) handles file wrappers, and thus allows you to serve files from more locations:

- models,
- storages,
- local filesystem,
- remote URL (using `requests` ¹¹),
- in-memory (or generated) files (such as `StringIO`),
- ... and your custom ones with little efforts.

django-private-files

`django-private-files` ¹² provides utilities for controlling access to static files based on conditions you can specify within your Django application.

django-protected-files

`django-protected-files` ¹³ is a Django application that lets you serve protected static files via your frontend server after authorizing the user against `django.contrib.auth`.

As of 2012-12-10, this project seems inactive.

References

4.7.2 License

Copyright (c) 2012, Benoît Bryon. All rights reserved.

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

- Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.
- Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.
- Neither the name of django-downloadview nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission.

¹¹ <https://pypi.python.org/pypi/requests>

¹² <http://pypi.python.org/pypi/django-private-files>

¹³ <https://github.com/lincolnloop/django-protected-files>

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS “AS IS” AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT HOLDER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

4.7.3 Authors & contributors

Maintainer: Benoît Bryon <benoit@marmelune.net>

Original code by Novapost team:

- Nicolas Tobo <<https://github.com/nicolastobo>>
- Lauréline Guérin <<https://github.com/zebuline>>
- Gregory Tappero <<https://github.com/coulx>>
- Benoît Bryon <benoit@marmelune.net>
- Rémy Hubscher <remy.hubscher@novapost.fr>

4.7.4 Changelog

1.1 (2013-04-11)

Various improvements. Contains **backward incompatible changes**.

- Added HTTPDownloadView to proxy to arbitrary URL.
- Added VirtualDownloadView to support files living in memory.
- Using StreamingHttpResponse introduced with Django 1.5. Makes Django 1.5 a requirement!
- Added `django_downloadview.test.assert_download_response` utility.
- Download views and response now use file wrappers. Most logic around file attributes, formerly in views, moved to wrappers.
- Replaced DownloadView by PathDownloadView and StorageDownloadView. Use the right one depending on the use case.

1.0 (2012-12-04)

- Introduced optimizations for Nginx X-Accel: a middleware and a decorator
- Introduced generic views: DownloadView and ObjectDownloadView
- Initialized project

4.8 Contributing to the project

This document provides guidelines for people who want to contribute to the project.

4.8.1 Create tickets

Please use the [bugtracker](#) ¹⁴ **before** starting some work:

- check if the bug or feature request has already been filed. It may have been answered too!
- else create a new ticket.
- if you plan to contribute, tell us, so that we are given an opportunity to give feedback as soon as possible.
- Then, in your commit messages, reference the ticket with some `refs #TICKET-ID` syntax.

4.8.2 Fork and branch

- Work in forks and branches.
- Prefix your branch with the ticket ID corresponding to the issue. As an example, if you are working on ticket #23 which is about contribute documentation, name your branch like `23-contribute-doc`.
- If you work in a development branch and want to refresh it with changes from master, please [rebase](#) ¹⁵ or [merge-based rebase](#) ¹⁶, i.e. don't merge master.

4.8.3 Setup a development environment

System requirements:

- [Python](#) ¹⁷ version 2.6 or 2.7, available as `python` command.

Note: You may use [Virtualenv](#) ¹⁸ to make sure the active `python` is the right one.

- `make` and `wget` to use the provided `Makefile`.

Execute:

```
git clone git@github.com:benoitbryon/django-downloadview.git
cd django-downloadview/
make develop
```

If you cannot execute the `Makefile`, read it and adapt the few commands it contains to your needs.

4.8.4 The Makefile

A `Makefile` is provided to ease development. Use it to:

- setup the development environment: `make develop`

¹⁴ <https://github.com/benoitbryon/django-downloadview/issues>

¹⁵ <http://git-scm.com/book/en/Git-Branching-Rebasing>

¹⁶ <http://tech.novapost.fr/psycho-rebasing-en.html>

¹⁷ <http://python.org>

¹⁸ <http://virtualenv.org>

- update it, as an example, after a pull: `make update`
- run tests: `make test`
- build documentation: `make documentation`

The `Makefile` is intended to be a live reference for the development environment.

4.8.5 Documentation

Follow [style guide for Sphinx-based documentations](#) ¹⁹ when editing the documentation.

4.8.6 Test and build

Use the `Makefile`.

4.8.7 Demo project included

The *Demo project* is part of the tests. Maintain it along with code and documentation.

4.8.8 References

¹⁹ <http://documentation-style-guide-sphinx.readthedocs.org/>

Python Module Index

d

- `django_downloadview.__init__, ??`
- `django_downloadview.decorators, ??`
- `django_downloadview.files, ??`
- `django_downloadview.middlewares, ??`
- `django_downloadview.nginx, ??`
- `django_downloadview.response, ??`
- `django_downloadview.test, ??`
- `django_downloadview.utils, ??`
- `django_downloadview.views, ??`