Django

Documentation

Source code for django.core.files.uploadedfile

```
"""
Classes representing uploaded files.
"""

import os
from io import BytesIO

from django.conf import settings
from django.core.files import temp as tempfile
from django.core.files.base import File

__all__ = ('UploadedFile', 'TemporaryUploadedFile', 'InMemoryUploadedFile',
           'SimpleUploadedFile')


[docs]class UploadedFile(File):
    """
    An abstract uploaded file (``TemporaryUploadedFile`` and
    ``InMemoryUploadedFile`` are the built-in concrete subclasses).

    An ``UploadedFile`` object behaves somewhat like a file object and
    represents some file data that the user submitted with a form.
    """

    def __init__(self, file=None, name=None, content_type=None, size=None, charset=None, content_type_extra=None):
        super().__init__(file, name)
        self.size = size
        self.content_type = content_type
        self.charset = charset
        self.content_type_extra = content_type_extra

    def __repr__(self):
        return "<%s: %s (%s)>" % (self.__class__.__name__, self.name, self.content_type)

    def _get_name(self):
        return self._name

    def _set_name(self, name):
        # Sanitize the file name so that it can't be dangerous.
        if name is not None:
            # Just use the basename of the file -- anything else is dangerous.
            name = os.path.basename(name)

            # File names longer than 255 characters can cause problems on older OSes.
            if len(name) > 255:
                name, ext = os.path.splitext(name)
                ext = ext[:255]
                name = name[:255 - len(ext)] + ext

        self._name = name

    name = property(_get_name, _set_name)



[docs]class TemporaryUploadedFile(UploadedFile):
    """
    A file uploaded to a temporary location (i.e. stream-to-disk).
    """
    def __init__(self, name, content_type, size, charset, content_type_extra=None):
        _, ext = os.path.splitext(name)
        file = tempfile.NamedTemporaryFile(suffix='.upload' + ext, dir=settings.FILE_UPLOAD_TEMP_DIR)
        super().__init__(file, name, content_type, size, charset, content_type_extra)

[docs]    def temporary_file_path(self):
        """Return the full path of this file."""
        return self.file.name


    def close(self):
        try:
            return self.file.close()
        except FileNotFoundError:
            # The file was moved or deleted before the tempfile could unlink
            # it. Still sets self.file.close_called and calls
            # self.file.file.close() before the exception.
            pass
```

```
[docs]class InMemoryUploadedFile(UploadedFile):
    """
    A file uploaded into memory (i.e. stream-to-memory).
    """
    def __init__(self, file, field_name, name, content_type, size, charset, content_type_extra=None):
        super().__init__(file, name, content_type, size, charset, content_type_extra)
        self.field_name = field_name

    def open(self, mode=None):
        self.file.seek(0)
        return self

    def chunks(self, chunk_size=None):
        self.file.seek(0)
        yield self.read()

    def multiple_chunks(self, chunk_size=None):
        # Since it's in memory, we'll never have multiple chunks.
        return False



class SimpleUploadedFile(InMemoryUploadedFile):
    """
    A simple representation of a file, which just has content, size, and a name.
    """
    def __init__(self, name, content, content_type='text/plain'):
        content = content or b''
        super().__init__(BytesIO(content), None, name, content_type, len(content), None, None)

    @classmethod
    def from_dict(cls, file_dict):
        """
        Create a SimpleUploadedFile object from a dictionary with keys:
           - filename
           - content-type
           - content
        """
        return cls(file_dict['filename'],
                   file_dict['content'],
                   file_dict.get('content-type', 'text/plain'))
```