faker-file Documentation

Release 0.8

Artur Barseghyan <artur.barseghyan@gmail.com>

Feb 10, 2023

CONTENTS

Generate fake files

CHAPTER

ONE

PREREQUISITES

All of core dependencies of this package are *MIT* licensed. Most of optional dependencies of this package are *MIT* licensed, while a few are *BSD*- or *Apache 2* licensed. All licenses are mentioned below between the brackets.

- Core package requires Python 3.7, 3.8, 3.9, 3.10 and 3.11.
- `Django` (*BSD*) integration with `factory_boy` (*MIT*) has been tested with `Django` 2.2, 3.0, 3.1, 3.2, 4.0 and 4.1.
- `DOCX` file support requires `python-docx` (*MIT*).
- `ICO`, `JPEG`, `PNG`, `SVG` and `WEBP` files support requires `imgkit` (*MIT*).
- `PDF` file support requires `pdfkit` (*MIT*).
- `PPTX` file support requires `python-pptx` (*MIT*).
- `ODS` file support requires `tablib` (*MIT*) and `odfpy` (*Apache 2*).
- `XLSX` file support requires `tablib` (*MIT*) and `openpyxl` (*MIT*).
- `PathyFileSystemStorage` storage support requires `pathy` (*Apache 2*).
- `AWSS3Storage` storage support requires `pathy` (*Apache 2*) and `boto3` (*Apache 2*).
- `AzureCloudStorage` storage support requires `pathy` (*Apache 2*) and `azure-storage-blob` (*MIT*).
- `GoogleCloudStorage` storage support requires `pathy` (*Apache 2*) and `google-cloud-storage` (*Apache 2*).

CHAPTER

TWO

DOCUMENTATION

Documentation is available on Read the Docs.

CHAPTER

THREE

INSTALLATION

Latest stable version on PyPI:

```
pip install faker-file[all]
```

Or development version from GitHub:

```
pip install https://github.com/barseghyanartur/faker-file/archive/main.tar.gz
```

CHAPTER

FOUR

SUPPORTED FILE TYPES

- BIN
- CSV
- DOCX
- ICO
- JPEG
- ODS
- PDF
- PNG
- PPTX
- SVG
- TXT
- WEBP
- XLSX
- ZIP

CHAPTER

FIVE

USAGE EXAMPLES

5.1 With Faker

One way

```
from faker import Faker
from faker_file.providers.txt_file import TxtFileProvider

FAKER = Faker()

file = TxtFileProvider(FAKER).txt_file()
```

Or another

```
from faker import Faker
from faker_file.providers.txt_file import TxtFileProvider

FAKER = Faker()
FAKER.add_provider(TxtFileProvider)

file = FAKER.txt_file()
```

5.2 With factory_boy

5.2.1 upload/models.py

```
from django.db import models

class Upload(models.Model):

    # ...
    file = models.FileField()
```

5.2.2 upload/factory.py

Note, that when using `faker-file` with `Django`, you need to pass your `MEDIA_ROOT` setting as `root_path` value (which is by default set to `tempfile.gettempdir()`).

```
import factory
from django.conf import settings
from factory import Faker
from factory.django import DjangoModelFactory
from faker_file.providers.docx_file import DocxFileProvider

from upload.models import Upload

factory.Faker.add_provider(DocxFileProvider)

class UploadFactory(DjangoModelFactory):

    # ...
    file = Faker("docx_file", root_path=settings.MEDIA_ROOT)

    class Meta:
        model = Upload
```

CHAPTER

SIX

SUPPORTED STORAGES

All file operations are delegated to a separate abstraction layer of storages.

The following storages are implemented:

- `FileSystemStorage`: Does not have additional requirements.
- `PathyFileSystemStorage`: Requires *pathy*.
- `AzureCloudStorage`: Requires *pathy* and *Azure* related dependencies.
- `GoogleCloudStorage`: Requires *pathy* and *Google Cloud* related dependencies.
- `AWSS3Storage`: Requires *pathy* and *AWS S3* related dependencies.

6.1 Usage example with storages

6.1.1 *FileSystemStorage* example

Native file system storage. Does not have dependencies.

```
import tempfile
from faker import Faker
from faker_file.providers.txt_file import TxtFileProvider
from faker_file.storages.filesystem import FileSystemStorage

FS_STORAGE = FileSystemStorage("root_path", "rel_path")

FAKER = Faker()

file = TxtFileProvider(FAKER).txt_file(storage=FS_STORAGE)

FS_STORAGE.exists(file)
```

6.1.2 *PathyFileSystemStorage* example

Native file system storage. Requires *pathy*.

```
import tempfile
from pathy import use_fs
from faker import Faker
from faker_file.providers.txt_file import TxtFileProvider
from faker_file.storages.cloud import PathyFileSystemStorage

use_fs(tempfile.gettempdir())
PATHY_FS_STORAGE = PathyFileSystemStorage("bucket_name", "rel_path")

FAKER = Faker()

file = TxtFileProvider(FAKER).txt_file(storage=PATHY_FS_STORAGE)

PATHY_FS_STORAGE.exists(file)
```

6.1.3 *AWSS3Storage* example

AWS S3 storage. Requires *pathy*.

```
import tempfile
from pathy import use_fs
from faker import Faker
from faker_file.providers.txt_file import TxtFileProvider
from faker_file.storages.aws_s3 import AWSS3Storage

S3_STORAGE = AWSS3Storage(
    bucket_name="bucket_name",
    rel_path="rel_path",
    credentials={"key_id": "YOUR KEY ID", "key_secret": "YOUR KEY SECRET"},
)

FAKER = Faker()

file = TxtFileProvider(FAKER).txt_file(storage=S3_STORAGE)

S3_STORAGE.exists(file)
```

CHAPTER

SEVEN

TESTING

Simply type:

```
pytest -vvv
```

Or use tox:

```
tox
```

Or use tox to check specific env:

```
tox -e py310-django41
```

CHAPTER

EIGHT

WRITING DOCUMENTATION

Keep the following hierarchy.

```
=====
title
=====

header
======

sub-header
----------

sub-sub-header
~~~~~~~~~~~~~~

sub-sub-sub-header
^^^^^^^^^^^^^^^^^^

sub-sub-sub-sub-header
++++++++++++++++++++++

sub-sub-sub-sub-sub-header
**************************
```

CHAPTER

NINE

LICENSE

MIT

CHAPTER

TEN

SUPPORT

For any security issues contact me at the e-mail given in the *Author* section.

For overall issues, go to GitHub.

CHAPTER

ELEVEN

AUTHOR

Artur Barseghyan <artur.barseghyan@gmail.com>

CHAPTER

TWELVE

PROJECT DOCUMENTATION

Contents:

Table of Contents

12.1 Quick start

12.1.1 Installation

```
pip install faker-file[all]
```

12.1.2 Usage

12.1.2.1 With Faker

```
from faker import Faker
from faker_file.providers.bin_file import BinFileProvider
from faker_file.providers.csv_file import CsvFileProvider
from faker_file.providers.docx_file import DocxFileProvider
from faker_file.providers.ico_file import IcoFileProvider
from faker_file.providers.jpeg_file import JpegFileProvider
from faker_file.providers.ods_file import OdsFileProvider
from faker_file.providers.pdf_file import PdfFileProvider
from faker_file.providers.png_file import PngFileProvider
from faker_file.providers.pptx_file import PptxFileProvider
from faker_file.providers.svg_file import SvgFileProvider
from faker_file.providers.txt_file import TxtFileProvider
from faker_file.providers.webp_file import WebpFileProvider
from faker_file.providers.xlsx_file import XlsxFileProvider
from faker_file.providers.zip_file import ZipFileProvider

FAKER = Faker()
FAKER.add_provider(BinFileProvider)
FAKER.add_provider(CsvFileProvider)
FAKER.add_provider(DocxFileProvider)
FAKER.add_provider(IcoFileProvider)
FAKER.add_provider(JpegFileProvider)
FAKER.add_provider(OdsFileProvider)
FAKER.add_provider(PdfFileProvider)
FAKER.add_provider(PngFileProvider)
FAKER.add_provider(PptxFileProvider)
FAKER.add_provider(SvgFileProvider)
FAKER.add_provider(TxtFileProvider)
FAKER.add_provider(WebpFileProvider)
FAKER.add_provider(XlsxFileProvider)
FAKER.add_provider(ZipFileProvider)

bin_file = FAKER.bin_file()
csv_file = FAKER.csv_file()
docx_file = FAKER.docx_file()
ico_file = FAKER.ico_file()
jpeg_file = FAKER.jpeg_file()
ods_file = FAKER.ods_file()
pdf_file = FAKER.pdf_file()
png_file = FAKER.png_file()
```

(continues on next page)

(continued from previous page)

```
pptx_file = FAKER.pptx_file()
svg_file = FAKER.svg_file()
txt_file = FAKER.txt_file()
webp_file = FAKER.webp_file()
xlsx_file = FAKER.xlsx_file()
zip_file = FAKER.zip_file()
```

12.1.2.2 With `factory_boy`

12.1.2.2.1 upload/models.py

```
from django.db import models

class Upload(models.Model):
    """Upload model."""

    name = models.CharField(max_length=255, unique=True)
    description = models.TextField(null=True, blank=True)

    # Files
    docx_file = models.FileField(null=True)
    pdf_file = models.FileField(null=True)
    pptx_file = models.FileField(null=True)
    txt_file = models.FileField(null=True)
    zip_file = models.FileField(null=True)

    class Meta:
        verbose_name = "Upload"
        verbose_name_plural = "Upload"

    def __str__(self):
        return self.name
```

12.1.2.2.2 upload/factories.py

```
from django.conf import settings

from factory import Faker
from factory.django import DjangoModelFactory

# Import all providers we want to use
from faker_file.providers.docx_file import DocxFileProvider
from faker_file.providers.pdf_file import PdfFileProvider
from faker_file.providers.pptx_file import PptxFileProvider
from faker_file.providers.txt_file import TxtFileProvider
from faker_file.providers.zip_file import ZipFileProvider

from upload.models import Upload
```

(continues on next page)

(continued from previous page)

```
# Add all providers we want to use
Faker.add_provider(DocxFileProvider)
Faker.add_provider(PdfFileProvider)
Faker.add_provider(PptxFileProvider)
Faker.add_provider(TxtFileProvider)
Faker.add_provider(ZipFileProvider)


class UploadFactory(DjangoModelFactory):
    """Upload factory."""

    name = Faker("text", max_nb_chars=100)
    description = Faker("text", max_nb_chars=1000)

    # Files
    docx_file = Faker("docx_file", root_path=settings.MEDIA_ROOT)
    pdf_file = Faker("pdf_file", root_path=settings.MEDIA_ROOT)
    pptx_file = Faker("pptx_file", root_path=settings.MEDIA_ROOT)
    txt_file = Faker("txt_file", root_path=settings.MEDIA_ROOT)
    zip_file = Faker("zip_file", root_path=settings.MEDIA_ROOT)

    class Meta:
        model = Upload
```

12.2 Recipes

12.2.1 When using with `Faker`

When using with `Faker`, there are two ways of using the providers.

12.2.1.1 Imports and initializations

12.2.1.1.1 One way

```
from faker import Faker
from faker_file.providers.bin_file import BinFileProvider
from faker_file.providers.docx_file import DocxFileProvider
from faker_file.providers.pdf_file import PdfFileProvider
from faker_file.providers.pptx_file import PptxFileProvider
from faker_file.providers.txt_file import TxtFileProvider
from faker_file.providers.zip_file import ZipFileProvider

FAKER = Faker()

# Usage example
file = TxtFileProvider(FAKER).txt_file(content="Lorem ipsum")
```

12.2.1.1.2 Or another

```
from faker import Faker
from faker_file.providers.docx_file import DocxFileProvider
from faker_file.providers.pdf_file import PdfFileProvider
from faker_file.providers.pptx_file import PptxFileProvider
from faker_file.providers.txt_file import TxtFileProvider
from faker_file.providers.zip_file import ZipFileProvider

FAKER = Faker()
FAKER.add_provider(DocxFileProvider)
FAKER.add_provider(PdfFileProvider)
FAKER.add_provider(PptxFileProvider)
FAKER.add_provider(TxtFileProvider)
FAKER.add_provider(ZipFileProvider)

# Usage example
file = FAKER.txt_file(content="Lorem ipsum")
```

Throughout documentation we will be mixing these approaches.

12.2.1.2 Create a TXT file with static content

- Content of the file is `Lorem ipsum`.

```
file = TxtFileProvider(FAKER).txt_file(content="Lorem ipsum")
```

12.2.1.3 Create a DOCX file with dynamically generated content

- Content is generated dynamically.
- Content is limited to 1024 chars.
- Wrap lines after 80 chars.
- Prefix the filename with `zzz`.

```
file = DocxFileProvider(FAKER).docx_file(
    prefix="zzz",
    max_nb_chars=1_024,
    wrap_chars_after=80,
)
```

12.2.1.4 Create a ZIP file consisting of TXT files with static content

- 5 TXT files in the ZIP archive (default value is 5).
- Content of all files is `Lorem ipsum`.

```
file = ZipFileProvider(FAKER).zip_file(options={"content": "Lorem ipsum"})
```

12.2.1.5 Create a ZIP file consisting of 3 DOCX files with dynamically generated content

- 3 DOCX files in the ZIP archive.
- Content is generated dynamically.
- Content is limited to 1024 chars.
- Prefix the filenames in archive with `xxx_`.
- Prefix the filename of the archive itself with `zzz`.
- Inside the ZIP, put all files in directory `yyy`.

```
from faker_file.providers.zip_file import create_inner_docx_file
file = ZipFileProvider(FAKER).zip_file(
    prefix="zzz",
    options={
        "count": 3,
        "create_inner_file_func": create_inner_docx_file,
        "create_inner_file_args": {
            "prefix": "xxx_",
            "max_nb_chars": 1_024,
        }
        "directory": "yyy",
    }
)
```

12.2.1.6 Create a nested ZIP file

Create a ZIP file which contains 5 ZIP files which contain 5 ZIP files which contain 5 DOCX files.

- 5 ZIP files in the ZIP archive.
- Content is generated dynamically.
- Prefix the filenames in archive with `nested_level_1_`.
- Prefix the filename of the archive itself with `nested_level_0_`.
- Each of the ZIP files inside the ZIP file in their turn contains 5 other ZIP files, prefixed with `nested_level_2_`, which in their turn contain 5 DOCX files.

```
from faker_file.providers.zip_file import create_inner_docx_file, create_inner_zip_file
file = ZipFileProvider(FAKER).zip_file(
    prefix="nested_level_0_",
    options={
        "create_inner_file_func": create_inner_zip_file,
        "create_inner_file_args": {
```

(continues on next page)

(continued from previous page)

```
            "prefix": "nested_level_1_",
            "options": {
                "create_inner_file_func": create_inner_zip_file,
                "create_inner_file_args": {
                    "prefix": "nested_level_2_",
                    "options": {
                        "create_inner_file_func": create_inner_docx_file,
                    }
                },
            }
        },
    }
)
```

12.2.1.7 Create a TXT file with static content

```
file = FAKER.txt_file(content="Lorem ipsum dolor sit amet")
```

12.2.1.8 Create a DOCX file with dynamically generated content

- Content is generated dynamically.
- Content is limited to 1024 chars.
- Wrap lines after 80 chars.
- Prefix the filename with `zzz`.

```
file = FAKER.docx_file(
    prefix="zzz",
    max_nb_chars=1_024,
    wrap_chars_after=80,
)
```

12.2.1.9 Create a PDF file with predefined template containing dynamic fixtures

- Content template is predefined and contains dynamic fixtures.
- Wrap lines after 80 chars.

```
template = """
{{date}} {{city}}, {{country}}

Hello {{name}},

{{text}} {{text}} {{text}}

{{text}} {{text}} {{text}}

{{text}} {{text}} {{text}}
```

(continues on next page)

(continued from previous page)

```
Address: {{address}}

Best regards,

{{name}}
{{address}}
{{phone_number}}
"""

file = FAKER.pdf_file(content=template, wrap_chars_after=80)
```

12.2.1.10 Pick a random file from a directory given

- Create an exact copy of the randomly picked file under a different name.
- Prefix of the destination file would be `zzz`.
- `source_dir_path` is the absolute path to the directory to pick files from.

```
from faker_file.providers.random_file_from_dir import (
    RandomFileFromDirProvider,
)

file = RandomFileFromDirProvider(FAKER).random_file_from_dir(
    source_dir_path="/tmp/tmp/",
    prefix="zzz",
)
```

12.2.2 When using with `Django` (and `factory_boy`)

When used with Django (to generate fake data with `factory_boy` factories), the `root_path` argument shall be provided. Otherwise (although no errors will be triggered) the generated files will reside outside the `MEDIA_ROOT` directory (by default in `/tmp/tmp/` on Linux) and further operations with those files through Django will cause `SuspiciousOperation` exception.

12.2.2.1 Basic example

12.2.2.1.1 Imaginary `Django` model

```
from django.db import models

class Upload(models.Model):
    """Upload model."""

    name = models.CharField(max_length=255, unique=True)
    description = models.TextField(null=True, blank=True)

    # Files
```

(continues on next page)

(continued from previous page)

```
    docx_file = models.FileField(null=True)
    pdf_file = models.FileField(null=True)
    pptx_file = models.FileField(null=True)
    txt_file = models.FileField(null=True)
    zip_file = models.FileField(null=True)
    file = models.FileField(null=True)

    class Meta:
        verbose_name = "Upload"
        verbose_name_plural = "Upload"

    def __str__(self):
        return self.name
```

12.2.2.1.2 Correspondent `factory_boy` factory

```
from django.conf import settings

from factory import Faker
from factory.django import DjangoModelFactory

# Import all providers we want to use
from faker_file.providers.docx_file import DocxFileProvider
from faker_file.providers.pdf_file import PdfFileProvider
from faker_file.providers.pptx_file import PptxFileProvider
from faker_file.providers.txt_file import TxtFileProvider
from faker_file.providers.zip_file import ZipFileProvider

from upload.models import Upload

# Add all providers we want to use
Faker.add_provider(DocxFileProvider)
Faker.add_provider(PdfFileProvider)
Faker.add_provider(PptxFileProvider)
Faker.add_provider(TxtFileProvider)
Faker.add_provider(ZipFileProvider)

class UploadFactory(DjangoModelFactory):
    """Upload factory."""

    name = Faker("text", max_nb_chars=100)
    description = Faker("text", max_nb_chars=1000)

    # Files
    docx_file = Faker("docx_file", root_path=settings.MEDIA_ROOT)
    pdf_file = Faker("pdf_file", root_path=settings.MEDIA_ROOT)
    pptx_file = Faker("pptx_file", root_path=settings.MEDIA_ROOT)
    txt_file = Faker("txt_file", root_path=settings.MEDIA_ROOT)
    zip_file = Faker("zip_file", root_path=settings.MEDIA_ROOT)
    file = Faker("txt_file", root_path=settings.MEDIA_ROOT)
```

(continues on next page)

(continued from previous page)

```
    class Meta:
        model = Upload
```

12.2.2.2 Randomize provider choice

```
from random import choice

from factory import LazyAttribute
from faker import Faker as FakerFaker

FAKER = FakerFaker()

PROVIDER_CHOICES = [
    lambda: DocxFileProvider(FAKER).docx_file(root_path=settings.MEDIA_ROOT),
    lambda: PdfFileProvider(FAKER).pdf_file(root_path=settings.MEDIA_ROOT),
    lambda: PptxFileProvider(FAKER).pptx_file(root_path=settings.MEDIA_ROOT),
    lambda: TxtFileProvider(FAKER).txt_file(root_path=settings.MEDIA_ROOT),
    lambda: ZipFileProvider(FAKER).zip_file(root_path=settings.MEDIA_ROOT),
]

def pick_random_provider(*args, **kwargs):
    return choice(PROVIDER_CHOICES)()

class UploadFactory(DjangoModelFactory):
    """Upload factory that randomly picks a file provider."""

    # ...
    file = LazyAttribute(pick_random_provider)
    # ...
```

12.2.2.3 Generate a file of a certain size

The only two file types for which it is easy to foresee the file size are BIN and TXT. Note, that size of BIN files is always exact, while for TXT it is approximate.

12.2.2.3.1 BIN

```
file = BinFileProvider(FAKER).bin_file(length=1024**2)  # 1 Mb
file = BinFileProvider(FAKER).bin_file(length=3*1024**2)  # 3 Mb
file = BinFileProvider(FAKER).bin_file(length=10*1024**2)  # 10 Mb

file = BinFileProvider(FAKER).bin_file(length=1024)  # 1 Kb
file = BinFileProvider(FAKER).bin_file(length=3*1024)  # 3 Kb
file = BinFileProvider(FAKER).bin_file(length=10*1024)  # 10 Kb
```

12.2.2.3.2 TXT

```
file = TxtFileProvider(FAKER).txt_file(max_nb_chars=1024**2)  # 1 Mb
file = TxtFileProvider(FAKER).txt_file(max_nb_chars=3*1024**2)  # 3 Mb
file = TxtFileProvider(FAKER).txt_file(max_nb_chars=10*1024**2)  # 10 Mb

file = TxtFileProvider(FAKER).txt_file(max_nb_chars=1024)  # 1 Kb
file = TxtFileProvider(FAKER).txt_file(max_nb_chars=3*1024)  # 3 Kb
file = TxtFileProvider(FAKER).txt_file(max_nb_chars=10*1024)  # 10 Kb
```

12.3 Release history and notes

Sequence based identifiers are used for versioning (schema follows below):

```
major.minor[.revision]
```

- It's always safe to upgrade within the same minor version (for example, from 0.3 to 0.3.4).
- Minor version changes might be backwards incompatible. Read the release notes carefully before upgrading (for example, when upgrading from 0.3.4 to 0.4).
- All backwards incompatible changes are mentioned in this document.

12.3.1 0.8

2022-12-16

Note, that this release introduces breaking changes!

- All file system based operations are moved to a separate abstraction layer of file storages. The following storages have been implemented: *FileSystemStorage*, *PathyFileSystemStorage*, *AWSS3Storage*, *GoogleCloudStorage* and *AzureStorage*. The *root_path* and *rel_path* params of the providers are deprecated in favour of storages. See the docs more usage examples.

12.3.2 0.7

2022-12-12

- Added `RandomFileFromDirProvider` which picks a random file from directory given.
- Improved docs.

12.3.3 0.6

2022-12-11

- Pass optional `generator` argument to inner functions of the `ZipFileProvider`.
- Added `create_inner_zip_file` inner function which allows to create nested ZIPs.
- Reached test coverage of 100%.

12.3.4 0.5

2022-12-10

Note, that this release introduces breaking changes!

- Added *ODS* file support.
- Switched to `tablib` for easy, non-variant support of various formats (*XLSX*, *ODS*).
- Silence `imgkit` logging output.
- *ZipFileProvider* allows to pass arbitrary arguments to inner functions. Put all your inner function arguments into a dictionary and pass it in *create_inner_file_args* key inside *options* argument. See the example below.

```
zip_file = ZipFileProvider(None).file(
    prefix="zzz_archive_",
    options={
        "count": 5,
        "create_inner_file_func": create_inner_docx_file,
        "create_inner_file_args": {
            "prefix": "zzz_file_",
            "max_nb_chars": 1_024,
            "content": "{{date}}\r\n{{text}}\r\n{{name}}",
        },
        "directory": "zzz",
    }
)
```

12.3.5 0.4

2022-12-09

Note, that this release introduces breaking changes!

- Remove the concept of content generators (and the correspondent `content_generator` arguments in implemented providers). Instead, allow usage of dynamic fixtures in the provided `content` argument.
- Remove temporary files when creating ZIP archives.
- Various improvements and fixes in docs.

12.3.6 0.3

2022-12-08

- Add support for *BIN*, *CSV* and *XLSX* files.
- Better visual representation of generated images and PDFs.

12.3.7 0.2

2022-12-07

- Added support for *ICO*, *JPEG*, *PNG*, *SVG* and *WEBP* files.
- Documentation improvements.

12.3.8 0.1

2022-12-06

- Initial beta release.

12.4 Package

Contents:

Table of Contents

12.4.1 faker_file package

12.4.1.1 Subpackages

12.4.1.1.1 faker_file.providers package

####### 12.4.1.1.1.1 Submodules

####### 12.4.1.1.1.2 faker_file.providers.bin_file module

class faker_file.providers.bin_file.**BinFileProvider**(*generator: Any*)

Bases: `BaseProvider`, *`FileMixin`*

BIN file provider.

Usage example:

from faker import Faker from faker_file.providers.bin_file import BinFileProvider

file = BinFileProvider(Faker()).bin_file()

Usage example with options:

file = BinFileProvider(Faker()).bin_file(
prefix="zzz", length=1024**2,

)

Usage example with *FileSystemStorage* storage (for *Django*):

from django.conf import settings from faker_file.storages.filesystem import FileSystemStorage

file = BinFileProvider(Faker()).bin_file(

storage=FileSystemStorage(
root_path=settings.MEDIA_ROOT, rel_path="tmp",

), prefix="zzz", length=1024**2,

)

Usage example with AWS S3 storage:

from faker_file.storages.aws_s3 import AWSS3Storage

file = BinFileProvider(Faker()).bin_file(
storage=AWSS3Storage(bucket_name="My-test-bucket"), prefix="zzz", length=1024**2,

)

bin_file(*storage: Optional[BaseStorage] = None*, *prefix: Optional[str] = None*, *length: int = 1048576*, *content: Optional[bytes] = None*, ***kwargs*) → *StringValue*

Generate a CSV file with random text.

Parameters

- **storage** – Storage class. Defaults to *FileSystemStorage*.
- **prefix** – File name prefix.
- **length** –
- **content** – File content. If given, used as is.

Returns
Relative path (from root directory) of the generated file.

extension: str = 'bin'

12.4.1.1.1.3 faker_file.providers.csv_file module

class faker_file.providers.csv_file.**CsvFileProvider**(*generator: Any*)

Bases: BaseProvider, *FileMixin*

CSV file provider.

Usage example:

from faker import Faker from faker_file.providers.csv_file import CsvFileProvider

file = CsvFileProvider(Faker()).csv_file()

Usage example with options:

from faker_file.providers.csv_file import CsvFileProvider

file = CsvFileProvider(Faker()).csv_file(
prefix="zzz", num_rows=100, data_columns=('{{name}}', '{{sentence}}', '{{address}}'), include_row_ids=True,

)

Usage example with *FileSystemStorage* storage (for *Django*):

from django.conf import settings from faker_file.storages.filesystem import FileSystemStorage

file = CsvFileProvider(Faker()).csv_file(

storage=FileSystemStorage(
root_path=settings.MEDIA_ROOT, rel_path="tmp",

), prefix="zzz", num_rows=100,

)

csv_file(*storage: Optional[BaseStorage] = None*, *prefix: Optional[str] = None*, *header: Optional[Sequence[str]] = None*, *data_columns: Tuple[str, str] = ('{{name}}', '{{address}}')*, *num_rows: int = 10*, *include_row_ids: bool = False*, *content: Optional[str] = None*, ***kwargs*) → *StringValue*

Generate a CSV file with random text.

Parameters

- **storage** – Storage. Defaults to *FileSystemStorage*.
- **prefix** – File name prefix.
- **header** – The `header` argument expects a list or a tuple of strings that will serve as the header row if supplied.
- **data_columns** – The `data_columns` argument expects a list or a tuple of string tokens, and these string tokens will be passed to `pystr_format()` for data generation. Argument Groups are used to pass arguments to the provider methods. Both `header` and `data_columns` must be of the same length.
- **num_rows** – The `num_rows` argument controls how many rows of data to generate, and the `include_row_ids` argument may be set to `True` to include a sequential row ID column.
- **include_row_ids** –
- **content** – File content. If given, used as is.

Returns
Relative path (from root directory) of the generated file.

extension: str = 'csv'

12.4.1.1.1.4 faker_file.providers.docx_file module

class faker_file.providers.docx_file.**DocxFileProvider**(*generator: Any*)

Bases: `BaseProvider`, *FileMixin*

DOCX file provider.

Usage example:

from faker import Faker from faker_file.providers.docx_file import DocxFileProvider

file = DocxFileProvider(Faker()).docx_file()

Usage example with options:

file = DocxFileProvider(Faker()).docx_file(
prefix="zzz", max_nb_chars=100_000, wrap_chars_after=80,

)

Usage example with *FileSystemStorage* storage (for *Django*):

from django.conf import settings from faker_file.storages.filesystem import FileSystemStorage

file = DocxFileProvider(Faker()).docx_file(

storage=FileSystemStorage(
root_path=settings.MEDIA_ROOT, rel_path="tmp",

), prefix="zzz", max_nb_chars=100_000, wrap_chars_after=80,

)

`docx_file`(*storage: Optional[BaseStorage] = None*, *prefix: Optional[str] = None*, *max_nb_chars: int = 10000*, *wrap_chars_after: Optional[int] = None*, *content: Optional[str] = None*, ***kwargs*) → *StringValue*

Generate a DOCX file with random text.

Parameters

- **storage** – Storage. Defaults to *FileSystemStorage*.
- **prefix** – File name prefix.
- **max_nb_chars** – Max number of chars for the content.
- **wrap_chars_after** – If given, the output string would be separated by line breaks after the given position.
- **content** – File content. Might contain dynamic elements, which are then replaced by correspondent fixtures.

Returns
Relative path (from root directory) of the generated file.

`extension: str = 'docx'`

12.4.1.1.1.5 faker_file.providers.ico_file module

`class faker_file.providers.ico_file.IcoFileProvider`(*generator: Any*)

Bases: `BaseProvider`, *ImageMixin*

ICO file provider.

Usage example:

from faker import Faker from faker_file.providers.png_file import IcoFileProvider

file = IcoFileProvider(Faker()).ico_file()

Usage example with options:

file = IcoFileProvider(Faker()).ico_file(
prefix="zzz", max_nb_chars=100_000, wrap_chars_after=80,

)

Usage example with *FileSystemStorage* storage (for *Django*):

from django.conf import settings from faker_file.storages.filesystem import FileSystemStorage

file = IcoFileProvider(Faker()).ico_file(

storage=FileSystemStorage(
root_path=settings.MEDIA_ROOT, rel_path="tmp",

), prefix="zzz", max_nb_chars=100_000, wrap_chars_after=80,

)

extension: str = 'ico'

ico_file(*storage: Optional[BaseStorage] = None*, *prefix: Optional[str] = None*, *max_nb_chars: int = 5000*, *wrap_chars_after: Optional[int] = None*, *content: Optional[str] = None*, ***kwargs*) → *StringValue*

Generate an ICO file with random text.

Parameters

- **storage** – Storage. Defaults to *FileSystemStorage*.
- **prefix** – File name prefix.
- **max_nb_chars** – Max number of chars for the content.
- **wrap_chars_after** – If given, the output string would be separated by line breaks after the given position.
- **content** – File content. Might contain dynamic elements, which are then replaced by correspondent fixtures.

Returns

Relative path (from root directory) of the generated file.

12.4.1.1.1.6 faker_file.providers.jpeg_file module

class faker_file.providers.jpeg_file.**JpegFileProvider**(*generator: Any*)

Bases: BaseProvider, *ImageMixin*

JPEG file provider.

Usage example:

from faker import Faker from faker_file.providers.jpeg_file import JpegFileProvider

file = JpegFileProvider(None).jpeg_file()

Usage example with options:

file = JpegFileProvider(None).jpeg_file(
prefix="zzz", max_nb_chars=100_000, wrap_chars_after=80,

)

Usage example with *FileSystemStorage* storage (for *Django*):

from django.conf import settings from faker_file.storages.filesystem import FileSystemStorage

file = JpegFileProvider(Faker()).jpeg_file(

storage=FileSystemStorage(
root_path=settings.MEDIA_ROOT, rel_path="tmp",

), prefix="zzz", max_nb_chars=100_000, wrap_chars_after=80,

)

extension: str = 'jpg'

jpeg_file(*storage: Optional[BaseStorage] = None*, *prefix: Optional[str] = None*, *max_nb_chars: int = 5000*, *wrap_chars_after: Optional[int] = None*, *content: Optional[str] = None*, ***kwargs*) → *StringValue*

Generate a JPEG file with random text.

Parameters

- **storage** – Storage. Defaults to *FileSystemStorage*.
- **prefix** – File name prefix.
- **max_nb_chars** – Max number of chars for the content.
- **wrap_chars_after** – If given, the output string would be separated by line breaks after the given position.
- **content** – File content. Might contain dynamic elements, which are then replaced by correspondent fixtures.

Returns

Relative path (from root directory) of the generated file.

12.4.1.1.1.7 faker_file.providers.mixins module

class faker_file.providers.mixins.**ImageMixin**

Bases: *FileMixin*

Image mixin.

extension: str

formats: List[str]

generator: Union[Provider, Faker]

numerify: Callable

random_element: Callable

12.4.1.1.1.8 faker_file.providers.ods_file module

class faker_file.providers.ods_file.**OdsFileProvider**(*generator: Any*)

Bases: BaseProvider, *FileMixin*

ODS file provider.

Usage example:

from faker import Faker from faker_file.providers.ods_file import OdsFileProvider

file = OdsFileProvider(Faker()).ods_file()

Usage example with options:

from faker import Faker from faker_file.providers.ods_file import OdsFileProvider

file = OdsFileProvider(Faker()).ods_file(
prefix="zzz", num_rows=100, data_columns={

"name": "{{name}}", "residency": "{{address}}",

}, include_row_ids=True,

)

Usage example with *FileSystemStorage* storage (for *Django*):

from django.conf import settings from faker_file.storages.filesystem import FileSystemStorage

file = OdsFileProvider(Faker()).ods_file(

storage=FileSystemStorage(
root_path=settings.MEDIA_ROOT, rel_path="tmp",

), prefix="zzz", num_rows=100, data_columns={

"name": "{{name}}", "residency": "{{address}}",

}, include_row_ids=True,

)

`extension: str = 'ods'`

`ods_file`(*storage: Optional[BaseStorage] = None*, *prefix: Optional[str] = None*, *data_columns: Optional[Dict[str, str]] = None*, *num_rows: int = 10*, *content: Optional[str] = None*, ***kwargs*) → *StringValue*

Generate a ODS file with random text.

Parameters

- **storage** – Storage. Defaults to *FileSystemStorage*.
- **data_columns** – The `data_columns` argument expects a list or a tuple of string tokens, and these string tokens will be passed to `pystr_format()` for data generation. Argument Groups are used to pass arguments to the provider methods. Both `header` and `data_columns` must be of the same length.
- **num_rows** – The `num_rows` argument controls how many rows of data to generate, and the `include_row_ids` argument may be set to `True` to include a sequential row ID column.
- **prefix** – File name prefix.
- **content** – List of dicts with content (JSON-like format). If given, used as is.

Returns
Relative path (from root directory) of the generated file.

12.4.1.1.1.9 faker_file.providers.pdf_file module

`class faker_file.providers.pdf_file.PdfFileProvider`(*generator: Any*)

Bases: `BaseProvider`, *FileMixin*

PDF file provider.

Usage example:

from faker_file.providers.pdf_file import PdfFileProvider

file = PdfFileProvider(None).pdf_file()

Usage example with options:

from faker_file.providers.pdf_file import PdfFileProvider

file = PdfFileProvider(None).pdf_file(
prefix="zzz", max_nb_chars=100_000, wrap_chars_after=80,

)

Usage example with *FileSystemStorage* storage (for *Django*):

from django.conf import settings from faker_file.storages.filesystem import FileSystemStorage

file = PdfFileProvider(Faker()).pdf_file(

storage=FileSystemStorage(
root_path=settings.MEDIA_ROOT, rel_path="tmp",

), prefix="zzz", max_nb_chars=100_000, wrap_chars_after=80,

)

`extension: str = 'pdf'`

`pdf_file`(*storage: Optional[BaseStorage] = None*, *prefix: Optional[str] = None*, *max_nb_chars: int = 10000*, *wrap_chars_after: Optional[int] = None*, *content: Optional[str] = None*, ***kwargs*) → *StringValue*

Generate a PDF file with random text.

Parameters

- **storage** – Storage. Defaults to *FileSystemStorage*.
- **prefix** – File name prefix.
- **max_nb_chars** – Max number of chars for the content.
- **wrap_chars_after** – If given, the output string would be separated by line breaks after the given position.
- **content** – File content. Might contain dynamic elements, which are then replaced by correspondent fixtures.

Returns
Relative path (from root directory) of the generated file.

12.4.1.1.1.10 faker_file.providers.png_file module

`class faker_file.providers.png_file.PngFileProvider`(*generator: Any*)

Bases: `BaseProvider`, *ImageMixin*

PNG file provider.

Usage example:

from faker import Faker from faker_file.providers.png_file import PngFileProvider

file = PngFileProvider(Faker()).png_file()

Usage example with options:

file = PngFileProvider(Faker()).png_file(
prefix="zzz", max_nb_chars=100_000, wrap_chars_after=80,

)

Usage example with *FileSystemStorage* storage (for *Django*):

from django.conf import settings from faker_file.storages.filesystem import FileSystemStorage

file = PngFileProvider(Faker()).png_file(

storage=FileSystemStorage(
root_path=settings.MEDIA_ROOT, rel_path="tmp",

), prefix="zzz", max_nb_chars=100_000, wrap_chars_after=80,

)

extension: str = 'png'

png_file(*storage: Optional[BaseStorage] = None*, *prefix: Optional[str] = None*, *max_nb_chars: int = 5000*, *wrap_chars_after: Optional[int] = None*, *content: Optional[str] = None*, ***kwargs*) → *StringValue*

Generate a PNG file with random text.

Parameters

- **storage** – Storage. Defaults to *FileSystemStorage*.
- **prefix** – File name prefix.
- **max_nb_chars** – Max number of chars for the content.
- **wrap_chars_after** – If given, the output string would be separated by line breaks after the given position.
- **content** – File content. Might contain dynamic elements, which are then replaced by correspondent fixtures.

Returns

Relative path (from root directory) of the generated file.

12.4.1.1.1.11 faker_file.providers.pptx_file module

class faker_file.providers.pptx_file.**PptxFileProvider**(*generator: Any*)

Bases: BaseProvider, *FileMixin*

PPTX file provider.

Usage example:

from faker_file.providers.pptx_file import PptxFileProvider

file = PptxFileProvider(None).pptx_file()

Usage example with options:

from faker_file.providers.pptx_file import PptxFileProvider

file = PptxFileProvider(None).pptx_file(
prefix="zzz", max_nb_chars=100_000, wrap_chars_after=80,

)

Usage example with *FileSystemStorage* storage (for *Django*):

from django.conf import settings from faker_file.storages.filesystem import FileSystemStorage

file = PptxFileProvider(Faker()).pptx_file(

storage=FileSystemStorage(
root_path=settings.MEDIA_ROOT, rel_path="tmp",

), prefix="zzz", max_nb_chars=100_000, wrap_chars_after=80,

)

extension: str = 'pptx'

pptx_file(*storage: Optional[BaseStorage] = None*, *prefix: Optional[str] = None*, *max_nb_chars: int = 10000*, *wrap_chars_after: Optional[int] = None*, *content: Optional[str] = None*, ***kwargs*) → *StringValue*

Generate a file with random text.

Parameters

- **storage** – Storage. Defaults to *FileSystemStorage*.
- **prefix** – File name prefix.
- **max_nb_chars** – Max number of chars for the content.
- **wrap_chars_after** – If given, the output string would be separated by line breaks after the given position.
- **content** – File content. Might contain dynamic elements, which are then replaced by correspondent fixtures.

Returns

Relative path (from root directory) of the generated file.

12.4.1.1.1.12 faker_file.providers.random_file_from_dir module

class faker_file.providers.random_file_from_dir.**RandomFileFromDirProvider**(*generator: Any*)

Bases: BaseProvider, *FileMixin*

Random file from given directory provider.

Usage example:

```
from faker_file.providers.random_file_from_dir import (
    RandomFileFromDirProvider,

)

file = RandomFileFromDirProvider(None).random_file_from_dir(
    source_dir_path="/tmp/tmp/",

)
```

Usage example with options:

```
from faker_file.providers.random_file_from_dir import (
    RandomFileFromDirProvider,

)

file = RandomFileFromDirProvider(None).random_file_from_dir(
    source_dir_path="/tmp/tmp/", prefix="zzz",

)
```

extension: *str* = ''

random_file_from_dir(*source_dir_path: str*, *root_path: Optional[str] = None*, *rel_path: str = 'tmp'*, *prefix: Optional[str] = None*, ***kwargs*) → *StringValue*

Pick a random file from given directory.

Parameters

- **source_dir_path** – Source files directory.

- **root_path** – Path of your files root directory (in case of Django it would be *settings.MEDIA_ROOT*).
- **rel_path** – Relative path (from root directory).
- **prefix** – File name prefix.

Returns
Relative path (from root directory) of the generated file.

12.4.1.1.1.13 faker_file.providers.svg_file module

class faker_file.providers.svg_file.**SvgFileProvider**(*generator: Any*)

Bases: BaseProvider, *ImageMixin*

SVG file provider.

Usage example:

from faker import Faker from faker_file.providers.svg_file import SvgFileProvider

file = SvgFileProvider(Faker()).svg_file()

Usage example with options:

file = SvgFileProvider(Faker()).svg_file(
prefix="zzz", max_nb_chars=100_000, wrap_chars_after=80,

)

Usage example with *FileSystemStorage* storage (for *Django*):

from django.conf import settings from faker_file.storages.filesystem import FileSystemStorage

file = SvgFileProvider(Faker()).svg_file(

storage=FileSystemStorage(
root_path=settings.MEDIA_ROOT, rel_path="tmp",

), prefix="zzz", max_nb_chars=100_000, wrap_chars_after=80,

)

extension: str = 'svg'

svg_file(*storage: Optional[BaseStorage] = None*, *prefix: Optional[str] = None*, *max_nb_chars: int = 5000*, *wrap_chars_after: Optional[int] = None*, *content: Optional[str] = None*, ***kwargs*) → *StringValue*

Generate an SVG file with random text.

Parameters

- **storage** – Storage. Defaults to *FileSystemStorage*.
- **prefix** – File name prefix.
- **max_nb_chars** – Max number of chars for the content.
- **wrap_chars_after** – If given, the output string would be separated by line breaks after the given position.
- **content** – File content. Might contain dynamic elements, which are then replaced by correspondent fixtures.

Returns
Relative path (from root directory) of the generated file.

12.4.1.1.1.14 faker_file.providers.txt_file module

class faker_file.providers.txt_file.**TxtFileProvider**(*generator: Any*)

Bases: BaseProvider, *FileMixin*

TXT file provider.

Usage example:

from faker_file.providers.txt_file import TxtFileProvider

file = TxtFileProvider(None).txt_file()

Usage example with options:

from faker_file.providers.txt_file import TxtFileProvider

file = TxtFileProvider(None).txt_file(
prefix="zzz", max_nb_chars=100_000, wrap_chars_after=80,

)

Usage example with *FileSystemStorage* storage (for *Django*):

from django.conf import settings from faker_file.storages.filesystem import FileSystemStorage

file = TxtFileProvider(Faker()).txt_file(

storage=FileSystemStorage(
root_path=settings.MEDIA_ROOT, rel_path="tmp",

), prefix="zzz", max_nb_chars=100_000, wrap_chars_after=80,

)

extension: str = 'txt'

txt_file(*storage: Optional[BaseStorage] = None*, *prefix: Optional[str] = None*, *max_nb_chars: int = 10000*, *wrap_chars_after: Optional[int] = None*, *content: Optional[str] = None*, ***kwargs*) → *StringValue*

Generate a TXT file with random text.

Parameters

- **storage** – Storage. Defaults to *FileSystemStorage*.
- **prefix** – File name prefix.
- **max_nb_chars** – Max number of chars for the content.
- **wrap_chars_after** – If given, the output string would be separated by line breaks after the given position.
- **content** – File content. Might contain dynamic elements, which are then replaced by correspondent fixtures.

Returns
Relative path (from root directory) of the generated file.

12.4.1.1.1.15 faker_file.providers.webp_file module

class faker_file.providers.webp_file.**WebpFileProvider**(*generator: Any*)

Bases: BaseProvider, *ImageMixin*

WEBP file provider.

Usage example:

from faker import Faker from faker_file.providers.webp_file import WebpFileProvider

file = WebpFileProvider(Faker()).webp_file()

Usage example with options:

file = WebpFileProvider(Faker()).webp_file(
prefix="zzz", max_nb_chars=100_000, wrap_chars_after=80,

)

Usage example with *FileSystemStorage* storage (for *Django*):

from django.conf import settings from faker_file.storages.filesystem import FileSystemStorage

file = WebpFileProvider(Faker()).webp_file(

storage=FileSystemStorage(
root_path=settings.MEDIA_ROOT, rel_path="tmp",

), prefix="zzz", max_nb_chars=100_000, wrap_chars_after=80,

)

extension: str = 'webp'

webp_file(*storage: Optional[BaseStorage] = None*, *prefix: Optional[str] = None*, *max_nb_chars: int = 5000*, *wrap_chars_after: Optional[int] = None*, *content: Optional[str] = None*, ***kwargs*) → *StringValue*

Generate a WEBP file with random text.

Parameters

- **storage** – Storage. Defaults to *FileSystemStorage*.
- **prefix** – File name prefix.
- **max_nb_chars** – Max number of chars for the content.
- **wrap_chars_after** – If given, the output string would be separated by line breaks after the given position.
- **content** – File content. Might contain dynamic elements, which are then replaced by correspondent fixtures.

Returns
Relative path (from root directory) of the generated file.

12.4.1.1.1.16 faker_file.providers.xlsx_file module

class faker_file.providers.xlsx_file.**XlsxFileProvider**(*generator: Any*)

Bases: BaseProvider, *FileMixin*

XLSX file provider.

Usage example:

from faker import Faker from faker_file.providers.xlsx_file import XlsxFileProvider

file = XlsxFileProvider(Faker()).xlsx_file()

Usage example with options:

from faker import Faker from faker_file.providers.xlsx_file import XlsxFileProvider

file = XlsxFileProvider(Faker()).xlsx_file(
prefix="zzz", num_rows=100, data_columns={

"name": "{{name}}", "residency": "{{address}}",

}, include_row_ids=True,

)

Usage example with *FileSystemStorage* storage (for *Django*):

from django.conf import settings from faker_file.storages.filesystem import FileSystemStorage

file = XlsxFileProvider(Faker()).xlsx_file(

storage=FileSystemStorage(
root_path=settings.MEDIA_ROOT, rel_path="tmp",

), prefix="zzz", num_rows=100, data_columns={

"name": "{{name}}", "residency": "{{address}}",

}, include_row_ids=True,

)

extension: str = 'xlsx'

xlsx_file(*storage: Optional[BaseStorage] = None*, *prefix: Optional[str] = None*, *data_columns: Optional[Dict[str, str]] = None*, *num_rows: int = 10*, *content: Optional[str] = None*, ***kwargs*) → *StringValue*

Generate a XLSX file with random text.

Parameters

- **storage** – Storage. Defaults to *FileSystemStorage*.
- **data_columns** – The `data_columns` argument expects a list or a tuple of string tokens, and these string tokens will be passed to `pystr_format()` for data generation. Argument Groups are used to pass arguments to the provider methods. Both `header` and `data_columns` must be of the same length.
- **num_rows** – The `num_rows` argument controls how many rows of data to generate, and the `include_row_ids` argument may be set to `True` to include a sequential row ID column.
- **prefix** – File name prefix.
- **content** – List of dicts with content (JSON-like format). If given, used as is.

Returns
Relative path (from root directory) of the generated file.

12.4.1.1.1.17 faker_file.providers.zip_file module

class faker_file.providers.zip_file.**ZipFileProvider**(*generator: Any*)

Bases: BaseProvider, *FileMixin*

ZIP file provider.

Usage example:

from faker import Faker from faker_file.providers.zip_file import ZipFileProvider

FAKER = Faker()

file = ZipFileProvider(FAKER).zip_file()

Usage example with options:

from faker_file.providers.zip_file import (
ZipFileProvider, create_inner_docx_file

)

file = ZipFileProvider(FAKER).zip_file(
prefix="**zzz_archive_**", options={

"count": 5, "create_inner_file_func": create_inner_docx_file, "create_inner_file_args": {

"prefix": "**zzz_docx_file_**", "max_nb_chars": 1_024,

}, "directory": "zzz",

}

)

Usage example of nested ZIPs:

file = ZipFileProvider(FAKER).zip_file(

options={
"create_inner_file_func": create_inner_zip_file, "create_inner_file_args": {

"options": {
"create_inner_file_func": create_inner_docx_file,

}

}

}

)

If you want to see, which files were included inside the zip, check the `file.data["files"]`.

extension: str = 'zip'

zip_file(*storage: Optional[BaseStorage] = None*, *prefix: Optional[str] = None*, *options: Optional[Dict[str, Any]] = None*, ***kwargs*) → *StringValue*

Generate a ZIP file with random text.

Parameters

- **storage** – Storage. Defaults to *FileSystemStorage*.
- **prefix** – File name prefix.
- **options** – Options (non-structured) for complex types, such as zip.

Returns

Relative path (from root directory) of the generated file.

faker_file.providers.zip_file.**create_inner_bin_file**(*storage: Optional[BaseStorage] = None*, *prefix: Optional[str] = None*, *generator: Optional[Union[Provider, Faker]] = None*, *length: int = 1048576*, *content: Optional[str] = None*, ***kwargs*) → *StringValue*

Create inner BIN file.

faker_file.providers.zip_file.**create_inner_csv_file**(*storage: Optional[BaseStorage] = None*, *prefix: Optional[str] = None*, *generator: Optional[Union[Provider, Faker]] = None*, *header: Optional[Sequence[str]] = None*, *data_columns: Tuple[str, str] = ('{{name}}', '{{address}}')*, *num_rows: int = 10*, *include_row_ids: bool = False*, *content: Optional[str] = None*, ***kwargs*) → *StringValue*

Create inner CSV file.

faker_file.providers.zip_file.**create_inner_docx_file**(*storage: Optional[BaseStorage] = None*, *prefix: Optional[str] = None*, *generator: Optional[Union[Provider, Faker]] = None*, *max_nb_chars: int = 10000*, *wrap_chars_after: Optional[int] = None*, *content: Optional[str] = None*, ***kwargs*) → *StringValue*

Create inner DOCX file.

faker_file.providers.zip_file.**create_inner_ico_file**(*storage: Optional[BaseStorage] = None*, *prefix: Optional[str] = None*, *generator: Optional[Union[Provider, Faker]] = None*, *max_nb_chars: int = 5000*, *wrap_chars_after: Optional[int] = None*, *content: Optional[str] = None*, ***kwargs*) → *StringValue*

Create inner ICO file.

faker_file.providers.zip_file.**create_inner_jpeg_file**(*storage: Optional[BaseStorage] = None*, *prefix: Optional[str] = None*, *generator: Optional[Union[Provider, Faker]] = None*, *max_nb_chars: int = 5000*, *wrap_chars_after: Optional[int] = None*, *content: Optional[str] = None*, ***kwargs*) → *StringValue*

Create inner JPEG file.

faker_file.providers.zip_file.**create_inner_ods_file**(*storage: Optional[BaseStorage] = None*, *prefix: Optional[str] = None*, *generator: Optional[Union[Provider, Faker]] = None*, *data_columns: Optional[Dict[str, str]] = None*, *num_rows: int = 10*, *content: Optional[str] = None*, ***kwargs*) → *StringValue*

Create inner ODS file.

faker_file.providers.zip_file.**create_inner_pdf_file**(*storage: Optional[BaseStorage] = None*, *prefix: Optional[str] = None*, *generator: Optional[Union[Provider, Faker]] = None*, *max_nb_chars: int = 10000*, *wrap_chars_after: Optional[int] = None*, *content: Optional[str] = None*, ***kwargs*) → *StringValue*

Create inner PDF file.

faker_file.providers.zip_file.**create_inner_png_file**(*storage: Optional[BaseStorage] = None*, *prefix: Optional[str] = None*, *generator: Optional[Union[Provider, Faker]] = None*, *max_nb_chars: int = 5000*, *wrap_chars_after: Optional[int] = None*, *content: Optional[str] = None*, ***kwargs*) → *StringValue*

Create inner PNG file.

faker_file.providers.zip_file.**create_inner_pptx_file**(*storage: Optional[BaseStorage] = None*, *prefix: Optional[str] = None*, *generator: Optional[Union[Provider, Faker]] = None*, *max_nb_chars: int = 10000*, *wrap_chars_after: Optional[int] = None*, *content: Optional[str] = None*, ***kwargs*) → *StringValue*

Create inner PPTX file.

faker_file.providers.zip_file.**create_inner_svg_file**(*storage: Optional[BaseStorage] = None*, *prefix: Optional[str] = None*, *generator: Optional[Union[Provider, Faker]] = None*, *max_nb_chars: int = 5000*, *wrap_chars_after: Optional[int] = None*, *content: Optional[str] = None*, ***kwargs*) → *StringValue*

Create inner SVG file.

faker_file.providers.zip_file.**create_inner_txt_file**(*storage: Optional[BaseStorage] = None*, *prefix: Optional[str] = None*, *generator: Optional[Union[Provider, Faker]] = None*, *max_nb_chars: int = 10000*, *wrap_chars_after: Optional[int] = None*, *content: Optional[str] = None*, ***kwargs*) → *StringValue*

Create inner TXT file.

faker_file.providers.zip_file.**create_inner_webp_file**(*storage: Optional[BaseStorage] = None*, *prefix: Optional[str] = None*, *generator: Optional[Union[Provider, Faker]] = None*, *max_nb_chars: int = 5000*, *wrap_chars_after: Optional[int] = None*, *content: Optional[str] = None*, ***kwargs*) → *StringValue*

Create inner WEBP file.

faker_file.providers.zip_file.**create_inner_xlsx_file**(*storage: Optional[BaseStorage] = None*, *prefix: Optional[str] = None*, *generator: Optional[Union[Provider, Faker]] = None*, *data_columns: Optional[Dict[str, str]] = None*, *num_rows: int = 10*, *content: Optional[str] = None*, ***kwargs*) → *StringValue*

Create inner XLSX file.

faker_file.providers.zip_file.**create_inner_zip_file**(*storage: Optional[BaseStorage] = None*, *prefix: Optional[str] = None*, *generator: Optional[Union[Provider, Faker]] = None*, *options: Optional[Dict[str, Any]] = None*, ***kwargs*) → *StringValue*

Create inner ZIP file.

12.4.1.1.1.18 Module contents

12.4.1.1.2 faker_file.storages package

12.4.1.1.2.1 Submodules

12.4.1.1.2.2 faker_file.storages.aws_s3 module

class faker_file.storages.aws_s3.**AWSS3Storage**(*bucket_name: str*, *rel_path: Optional[str] = 'tmp'*, *credentials: Optional[Dict[str, Any]] = None*, **args*, ***kwargs*)

Bases: *CloudStorage*

AWS S3 Storage.

Usage example:

from faker_file.storages.aws_s3 import AWSS3Storage

s3_storage = AWSS3Storage(
bucket_name="artur-testing-1", rel_path="tmp",

) file = s3_storage.generate_filename(prefix="zzz_", extension="docx") s3_storage.write_text(file, "Lorem ipsum") s3_storage.write_bytes(file, b"Lorem ipsum")

authenticate(*key_id: str*, *key_secret: str*, ***kwargs*) → None

Authenticate to AWS S3.

schema: str = 's3'

12.4.1.1.2.3 faker_file.storages.azure_cloud_storage module

class faker_file.storages.azure_cloud_storage.**AzureCloudStorage**(*bucket_name: str*, *rel_path: Optional[str] = 'tmp'*, *credentials: Optional[Dict[str, Any]] = None*, **args*, ***kwargs*)

Bases: *CloudStorage*

Azure Cloud Storage.

Usage example:

from faker_file.storages.azure_cloud_storage import AzureCloudStorage

azure_storage = AzureCloudStorage(
bucket_name="artur-testing-1", rel_path="tmp",

) file = azure_storage.generate_filename(prefix="zzz_", extension="docx") azure_storage.write_text(file, "Lorem ipsum") azure_storage.write_bytes(file, b"Lorem ipsum")

authenticate(*connection_string: str*, ***kwargs*) → None

Authenticate to Azure Cloud Storage.

bucket: Pathy

bucket_name: str

credentials: Dict[str, str]

schema: str = 'azure'

12.4.1.1.2.4 faker_file.storages.base module

class faker_file.storages.base.**BaseStorage**(**args*, ***kwargs*)

Bases: `object`

Base storage.

abspath(*filename: Any*) → str

Return absolute path.

exists(*filename: Any*) → bool

Check if file exists.

generate_filename(*prefix: str*, *extension: str*) → Any

Generate filename.

generate_temporary_local_filename(*prefix: str*, *extension: str*) → str

relpath(*filename: Any*) → str

Return relative path.

write_bytes(*filename: Any*, *data: bytes*) → int

Write bytes.

write_text(*filename: Any*, *data: str*, *encoding: Optional[str] = None*) → int

Write text.

12.4.1.1.2.5 faker_file.storages.cloud module

class faker_file.storages.cloud.**CloudStorage**(*bucket_name: str*, *rel_path: Optional[str] = 'tmp'*, *credentials: Optional[Dict[str, Any]] = None*, **args*, ***kwargs*)

Bases: *BaseStorage*

Base cloud storage.

Usage example:

from faker_file.storages.cloud import CloudStorage

storage = CloudStorage(
schema="s3", bucket_name="artur-testing-1", rel_path="tmp",

) file = storage.generate_filename(prefix="zzz_", extension="docx") storage.write_text(file, "Lorem ipsum") storage.write_bytes(file, b"Lorem ipsum")

abspath(*filename: Pathy*) → str

Return relative path.

authenticate(***kwargs*)

bucket: Pathy

bucket_name: str

credentials: Dict[str, str]

exists(*filename: Union[Pathy, str]*) → bool

Check if file exists.

generate_filename(*prefix: str*, *extension: str*) → Pathy

Generate filename.

relpath(*filename: Pathy*) → str

Return relative path.

schema: str = None

write_bytes(*filename: Pathy*, *data: bytes*) → int

Write bytes.

write_text(*filename: Pathy*, *data: str*, *encoding: Optional[str] = None*) → int

Write text.

class faker_file.storages.cloud.**PathyFileSystemStorage**(*bucket_name: str*, *rel_path: Optional[str] = 'tmp'*, *credentials: Optional[Dict[str, Any]] = None*, **args*, ***kwargs*)

Bases: *CloudStorage*

Pathy FileSystem Storage.

Usage example:

from faker_file.storages.cloud import PathyFileSystemStorage

fs_storage = PathyFileSystemStorage(
bucket_name="artur-testing-1", rel_path="tmp",

) file = fs_storage.generate_filename(prefix="zzz_", extension="docx") fs_storage.write_text(file, "Lorem ipsum") fs_storage.write_bytes(file, b"Lorem ipsum")

authenticate(***kwargs*) → None

Authenticate. Does nothing.

schema: str = 'file'

12.4.1.1.2.6 faker_file.storages.filesystem module

class faker_file.storages.filesystem.**FileSystemStorage**(*root_path: Optional[str] = '/tmp'*, *rel_path: Optional[str] = 'tmp'*, **args*, ***kwargs*)

Bases: *BaseStorage*

File storage.

Usage example:

from faker_file.storages.filesystem import FileSystemStorage

storage = FileSystemStorage() file = storage.generate_filename(prefix="zzz_", extension="docx") storage.write_text(file, "Lorem ipsum") storage.write_bytes(file, b"Lorem ipsum")

Initialization with params:

storage = FileSystemStorage()

abspath(*filename: str*) → str

Return absolute path.

exists(*filename: str*) → bool

Write bytes.

generate_filename(*prefix: str*, *extension: str*) → str

Generate filename.

relpath(*filename: str*) → str

Return relative path.

write_bytes(*filename: str*, *data: bytes*) → int

Write bytes.

write_text(*filename: str*, *data: str*, *encoding: Optional[str] = None*) → int

Write text.

12.4.1.1.2.7 faker_file.storages.google_cloud_storage module

class faker_file.storages.google_cloud_storage.**GoogleCloudStorage**(*bucket_name: str*, *rel_path: Optional[str] = 'tmp'*, *credentials: Optional[Dict[str, Any]] = None*, **args*, ***kwargs*)

Bases: *CloudStorage*

Google Cloud Storage.

Usage example:

from faker_file.storages.google_cloud_storage import GoogleCloudStorage

gs_storage = GoogleCloudStorage(
bucket_name="artur-testing-1", rel_path="tmp",

) file = gs_storage.generate_filename(prefix="zzz_", extension="docx") gs_storage.write_text(file, "Lorem ipsum") gs_storage.write_bytes(file, b"Lorem ipsum")

authenticate(*json_file_path: str*, ***kwargs*) → None
Authenticate to Google Cloud Storage.

bucket: Pathy

bucket_name: str

credentials: Dict[str, str]

schema: str = 'gs'

12.4.1.1.2.8 Module contents

12.4.1.1.3 faker_file.tests package

12.4.1.1.3.1 Submodules

12.4.1.1.3.2 faker_file.tests.test_django_integration module

class faker_file.tests.test_django_integration.**DjangoIntegrationTestCase**(*methodName='runTest'*)
Bases: TestCase

Django integration test case.

FAKER: Faker

FS_STORAGE = <faker_file.storages.filesystem.FileSystemStorage object>

test_file

12.4.1.1.3.3 faker_file.tests.test_providers module

class faker_file.tests.test_providers.**ProvidersTestCase**(*methodName='runTest'*)
Bases: TestCase

Providers test case.

FAKER: Faker

setUp(**args*, ***kwargs*)
Hook method for setting up the test fixture before exercising it.

test_broken_imports

test_faker

test_generate_filename_failure() → None
Test generate filename failure.

test_standalone_providers

test_standalone_providers_allow_failures

test_standalone_zip_file

test_standalone_zip_file_allow_failures

12.4.1.1.3.4 faker_file.tests.test_storages module

class faker_file.tests.test_storages.**TestStoragesTestCase**(*methodName='runTest'*)

Bases: TestCase

Test storages.

test_storage

test_storage_exceptions

12.4.1.1.3.5 Module contents

12.4.1.2 Submodules

12.4.1.3 faker_file.base module

class faker_file.base.**FileMixin**

Bases: object

File mixin.

extension: str

formats: List[str]

generator: Union[Provider, Faker]

numerify: Callable

random_element: Callable

class faker_file.base.**StringValue**

Bases: str

data: Dict[str, Any] = {}

12.4.1.4 faker_file.constants module

12.4.1.5 faker_file.helpers module

faker_file.helpers.**wrap_text**(*text: str*, *wrap_chars_after: int*) → str

12.4.1.6 Module contents

CHAPTER

THIRTEEN

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

f

INDEX

D

E

F

G

I

J

M

N

O

P

R

S

T

W

X

Z