
faker-file Documentation

Release 0.4

Artur Barseghyan <artur.barseghyan@gmail.com>

Feb 10, 2023

CONTENTS

1	Prerequisites	3
2	Documentation	5
3	Installation	7
4	Supported file types	9
5	Usage examples	11
5.1	With Faker	11
5.1.1	One way	11
5.1.2	Or another	11
5.2	With factory_boy	11
5.2.1	upload/models.py	11
5.2.2	upload/factory.py	12
6	Testing	13
7	Writing documentation	15
8	License	17
9	Support	19
10	Author	21
11	Project documentation	23
11.1	Quick start	24
11.1.1	Installation	24
11.1.2	Usage	24
11.1.2.1	With Faker	24
11.1.2.2	With factory_boy	25
11.1.2.2.1	upload/models.py	25
11.1.2.2.2	upload/factories.py	25
11.2	Recipes	26
11.2.1	When using standalone	26
11.2.1.1	Prerequisites	26
11.2.1.1.1	Create a TXT file with static content	26
11.2.1.1.2	Create a DOCX file with dynamically generated content	27
11.2.1.1.3	Create a ZIP file consisting of TXT files with static content	27

11.2.1.1.4	Create a ZIP file consisting of 3 DOCX files with dynamically generated content	27
11.2.2	When using with Faker	28
11.2.2.1	Create a TXT file with static content	28
11.2.2.2	Create a DOCX file with dynamically generated content	28
11.2.2.3	Create a PDF file with predefined template containing dynamic fixtures	28
11.2.3	When using with Django	29
11.2.3.1	Basic example	29
11.2.3.1.1	Imaginary Django model	29
11.2.3.1.2	Correspondent factory_boy factory	30
11.2.3.2	Randomise provider choice	30
11.3	Release history and notes	31
11.3.1	0.4	31
11.3.2	0.3	31
11.3.3	0.2	32
11.3.4	0.1	32
11.4	Package	32
11.4.1	faker_file package	32
11.4.1.1	Subpackages	32
11.4.1.1.1	faker_file.providers package	32
11.4.1.1.1.1	Submodules	32
11.4.1.1.1.2	faker_file.providers.bin_file module	32
11.4.1.1.1.3	faker_file.providers.csv_file module	33
11.4.1.1.1.4	faker_file.providers.docx_file module	34
11.4.1.1.1.5	faker_file.providers.ico_file module	35
11.4.1.1.1.6	faker_file.providers.jpeg_file module	36
11.4.1.1.1.7	faker_file.providers.pdf_file module	36
11.4.1.1.1.8	faker_file.providers.png_file module	37
11.4.1.1.1.9	faker_file.providers.pptx_file module	38
11.4.1.1.1.10	faker_file.providers.svg_file module	39
11.4.1.1.1.11	faker_file.providers.txt_file module	40
11.4.1.1.1.12	faker_file.providers.webp_file module	40
11.4.1.1.1.13	faker_file.providers.xlsx_file module	41
11.4.1.1.1.14	faker_file.providers.zip_file module	42
11.4.1.1.1.15	Module contents	44
11.4.1.1.2	faker_file.tests package	44
11.4.1.1.2.1	Submodules	44
11.4.1.1.2.2	faker_file.tests.test_django_integration module	44
11.4.1.1.2.3	faker_file.tests.test_providers module	45
11.4.1.1.2.4	Module contents	45
11.4.1.2	Submodules	45
11.4.1.3	faker_file.base module	45
11.4.1.4	faker_file.constants module	46
11.4.1.5	faker_file.content_generators module	46
11.4.1.6	faker_file.helpers module	46
11.4.1.7	Module contents	46
12	Indices and tables	47
	Python Module Index	49
	Index	51

Generate fake files

PREREQUISITES

- Core package requires Python 3.7, 3.8, 3.9, 3.10 and 3.11.
- Django integration (with `factory_boy`) has been tested with Django 2.2, 3.0, 3.1, 3.2, 4.0 and 4.1.
- *DOCX* file support requires `python-docx`.
- *ICO*, *JPEG*, *PNG*, *SVG* and *WEBP* files support requires `imgkit`.
- *PDF* file support requires `pdftk`.
- *PPTX* file support requires `python-pptx`.
- *XLSX* file support requires `openpyxl`.

DOCUMENTATION

Documentation is available on [Read the Docs](#).

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
```


SUPPORTED FILE TYPES

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

USAGE EXAMPLES

5.1 With Faker

5.1.1 One way

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

FAKER = Faker()

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

5.1.2 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
```


TESTING

Simply type:

```
pytest -vvv
```

Or use tox:

```
tox
```

Or use tox to check specific env:

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


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  
*****
```


LICENSE

MIT

SUPPORT

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

For overall issues, go to [GitHub](#).

AUTHOR

Artur Barseghyan <artur.barseghyan@gmail.com>

PROJECT DOCUMENTATION

Contents:

Table of Contents

- *faker-file*
 - *Prerequisites*
 - *Documentation*
 - *Installation*
 - *Supported file types*
 - *Usage examples*
 - * *With Faker*
 - *One way*
 - *Or another*
 - * *With factory_boy*
 - *upload/models.py*
 - *upload/factory.py*
 - *Testing*
 - *Writing documentation*
 - *License*
 - *Support*
 - *Author*
 - *Project documentation*
 - *Indices and tables*

11.1 Quick start

11.1.1 Installation

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

11.1.2 Usage

11.1.2.1 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.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(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()
pdf_file = FAKER.pdf_file()
png_file = FAKER.png_file()
pptx_file = FAKER.pptx_file()
svg_file = FAKER.svg_file()
txt_file = FAKER.txt_file()
```

(continues on next page)

(continued from previous page)

```
webp_file = FAKER.webp_file()
xlsx_file = FAKER.xlsx_file()
zip_file = FAKER.zip_file()
```

11.1.2.2 With factory_boy

11.1.2.2.1 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
```

11.1.2.2.2 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

# Add all providers we want to use
Faker.add_provider(DocxFileProvider)
```

(continues on next page)

(continued from previous page)

```
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
```

11.2 Recipes

11.2.1 When using standalone

11.2.1.1 Prerequisites

Imports and initializations

```
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()
```

11.2.1.1.1 Create a TXT file with static content

- Content of the file is Lorem ipsum.

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

11.2.1.1.2 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,
)
```

11.2.1.1.3 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"})
```

11.2.1.1.4 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,
        "max_nb_chars": 1_024,
        "prefix": "xxx_",
        "directory": "yyy",
    }
)
```

11.2.2 When using with Faker

Imports and initializations

```
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)
```

11.2.2.1 Create a TXT file with static content

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

11.2.2.2 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,
)
```

11.2.2.3 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}}
```

(continues on next page)

(continued from previous page)

```

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

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

Address: {{address}}

Best regards,

{{name}}
{{address}}
{{phone_number}}
"""
file = FAKER("pdf_file", content=template, wrap_chars_after=80)

```

11.2.3 When using with Django

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.

11.2.3.1 Basic example

11.2.3.1.1 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
    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

```

11.2.3.1.2 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)

    class Meta:
        model = Upload
```

11.2.3.2 Randomise provider choice

```
from random import choice

from factory import LazyAttribute

PROVIDER_CHOICES = [
    lambda: DocxFileProvider(None).docx_file(root_path=settings.MEDIA_ROOT),
    lambda: PdfFileProvider(None).pdf_file(root_path=settings.MEDIA_ROOT),
    lambda: PptxFileProvider(None).pptx_file(root_path=settings.MEDIA_ROOT),
    lambda: TxtFileProvider(None).txt_file(root_path=settings.MEDIA_ROOT),
```

(continues on next page)

(continued from previous page)

```

    lambda: ZipFileProvider(None).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)
    # ...

```

11.3 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.

11.3.1 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.

11.3.2 0.3

2022-12-08

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

11.3.3 0.2

2022-12-07

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

11.3.4 0.1

2022-12-06

- Initial beta release.

11.4 Package

Contents:

Table of Contents

- *Package*

11.4.1 faker_file package

11.4.1.1 Subpackages

11.4.1.1.1 faker_file.providers package

11.4.1.1.1.1 Submodules

11.4.1.1.1.2 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_file.providers.bin_file import BinFileProvider
file = BinFileProvider(None).bin_file()
```

Usage example with options:

```
from faker_file.providers.bin_file import BinFileProvider
file = BinFileProvider(None).bin_file(
    prefix="zzz", length=1024**2,
)
```

bin_file(*root_path: Optional[str] = None, rel_path: str = 'tmp', prefix: Optional[str] = None, length: int = 1048576, content: Optional[bytes] = None, **kwargs*) → *StringValue*

Generate a CSV file with random text.

Parameters

- **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.
- **length** –
- **content** – File content. If given, used as is.

Returns

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

extension: `str = 'bin'`

11.4.1.1.1.3 `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_file.providers.csv_file import CsvFileProvider
file = CsvFileProvider(None).csv_file()
```

Usage example with options:

```
from faker_file.providers.csv_file import CsvFileProvider

file = CsvFileProvider(None).csv_file(
    prefix="zzz", num_rows=100, data_columns=('{{name}}', '{{sentence}}', '{{address}}'),
    include_row_ids=True,
)
```

csv_file(*root_path: Optional[str] = None, rel_path: str = 'tmp', 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

- **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).
- **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** –
- **prefix** – File name prefix.
- **content** – File content. If given, used as is.

Returns

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

extension: `str = 'csv'`

11.4.1.1.1.4 `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_file.providers.docx_file import DocxFileProvider
```

```
file = DocxFileProvider(None).docx_file()
```

Usage example with options:

```
from faker_file.providers.docx_file import DocxFileProvider
```

```
file = DocxFileProvider(None).docx_file(  
    prefix="zzz", max_nb_chars=100_000, wrap_chars_after=80,  
)
```

docx_file(*root_path: Optional[str] = None, rel_path: str = 'tmp', 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

- **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.
- **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'`

11.4.1.1.1.5 `faker_file.providers.ico_file` module

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

Bases: `BaseProvider`, `FileMixin`

ICO file provider.

Usage example:

```
from faker_file.providers.png_file import IcoFileProvider
file = IcoFileProvider(None).ico_file()
```

Usage example with options:

```
from faker_file.providers.ico_file import IcoFileProvider

file = IcoFileProvider(None).ico_file(
    prefix="zzz", max_nb_chars=100_000, wrap_chars_after=80,
)
```

extension: `str = 'ico'`

ico_file(*root_path: Optional[str] = None, rel_path: str = 'tmp', 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

- **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.
- **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.

11.4.1.1.1.6 `faker_file.providers.jpeg_file` module

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

Bases: `BaseProvider`, `FileMixin`

JPEG file provider.

Usage example:

```
from faker_file.providers.jpeg_file import JpegFileProvider
file = JpegFileProvider(None).jpeg_file()
```

Usage example with options:

```
from faker_file.providers.jpeg_file import JpegFileProvider
file = JpegFileProvider(None).jpeg_file(
    prefix="zzz", max_nb_chars=100_000, wrap_chars_after=80,
)
```

extension: `str = 'jpg'`

jpeg_file(*root_path: Optional[str] = None, rel_path: str = 'tmp', 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

- **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.
- **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.

11.4.1.1.1.7 `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,
)

extension:  str = 'pdf'

pdf_file(root_path: Optional[str] = None, rel_path: str = 'tmp', 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

- **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.
- **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.

11.4.1.1.1.8 faker_file.providers.png_file module

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

Bases: *BaseProvider*, *FileMixin*

PNG file provider.

Usage example:

```

from faker_file.providers.png_file import PngFileProvider

file = PngFileProvider(None).png_file()

```

Usage example with options:

```

from faker_file.providers.png_file import PngFileProvider

file = PngFileProvider(None).png_file(
    prefix="zzz", max_nb_chars=100_000, wrap_chars_after=80,
)

extension:  str = 'png'

png_file(root_path: Optional[str] = None, rel_path: str = 'tmp', 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

- **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.
- **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.

11.4.1.1.1.9 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,
)
```

extension: `str = 'pptx'`

```
pptx_file(root_path: Optional[str] = None, rel_path: str = 'tmp', 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

- **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.
- **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.

11.4.1.1.10 faker_file.providers.svg_file module

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

Bases: `BaseProvider`, `FileMixin`

SVG file provider.

Usage example:

```
from faker_file.providers.svg_file import SvgFileProvider
```

```
file = SvgFileProvider(None).svg_file()
```

Usage example with options:

```
from faker_file.providers.svg_file import SvgFileProvider
```

```
file = SvgFileProvider(None).svg_file(
    prefix="zzz", max_nb_chars=100_000, wrap_chars_after=80,
)
```

extension: `str = 'svg'`

svg_file(*root_path: Optional[str] = None, rel_path: str = 'tmp', 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

- **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.
- **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.

11.4.1.1.1.11 `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,
)
```

extension: `str = 'txt'`

txt_file(*root_path: Optional[str] = None, rel_path: str = 'tmp', 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

- **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.
- **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.

11.4.1.1.1.12 `faker_file.providers.webp_file` module

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

Bases: `BaseProvider`, `FileMixin`

WEBP file provider.

Usage example:

```
from faker_file.providers.webp_file import WebpFileProvider
file = WebpFileProvider(None).webp_file()
```

Usage example with options:

```

from faker_file.providers.webp_file import WebpFileProvider

file = WebpFileProvider(None).webp_file(
    prefix="zzz", max_nb_chars=100_000, wrap_chars_after=80,
)

extension: str = 'webp'

webp_file(root_path: Optional[str] = None, rel_path: str = 'tmp', 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

- **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.
- **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.

11.4.1.1.13 `faker_file.providers.xlsx_file` module

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

Bases: `BaseProvider`, *FileMixin*

CSV file provider.

Usage example:

```

from faker_file.providers.xlsx_file import XlsxFileProvider

file = XlsxFileProvider(None).xlsx_file()

```

Usage example with options:

```

from faker_file.providers.xlsx_file import XlsxFileProvider

file = XlsxFileProvider(None).xlsx_file(
    prefix="zzz", num_rows=100, data_columns={
        "name": "{{name}}", "residency": "{{address}}",
    }, include_row_ids=True,
)

extension: str = 'xlsx'

```

```
xlsx_file(root_path: Optional[str] = None, rel_path: str = 'tmp', 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

- **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).
- **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.

11.4.1.1.14 `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_file.providers.zip_file import ZipFileProvider
file = ZipFileProvider(None).zip_file()
```

Usage example with options:

```
from faker_file.providers.zip_file import (
    ZipFileProvider, create_inner_docx_file
)

file = ZipFileProvider(None).zip_file(
    prefix="zzz_archive_", options={
        "count": 5, "create_inner_file_func": create_inner_docx_file, "max_nb_chars": 1_024,
        "prefix": "zzz_docx_file_", "directory": "zzz",
    }
)
```

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

```
extension: str = 'zip'
```

zip_file(*root_path: Optional[str] = None, rel_path: str = 'tmp', prefix: Optional[str] = None, options: Optional[Dict[str, Any]] = None, **kwargs*) → *StringValue*

Generate a ZIP file with random text.

Parameters

- **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.
- **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(*root_path: Optional[str] = None, rel_path: str = 'tmp', prefix: Optional[str] = None, length: int = 1048576, content: Optional[str] = None, **kwargs*) → *StringValue*

Create inner BIN file.

faker_file.providers.zip_file.create_inner_csv_file(*root_path: Optional[str] = None, rel_path: str = 'tmp', 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*

Create inner CSV file.

faker_file.providers.zip_file.create_inner_docx_file(*root_path: Optional[str] = None, rel_path: str = 'tmp', prefix: Optional[str] = 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(*root_path: Optional[str] = None, rel_path: str = 'tmp', prefix: Optional[str] = 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(*root_path: Optional[str] = None, rel_path: str = 'tmp', prefix: Optional[str] = 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_pdf_file(*root_path: Optional[str] = None, rel_path: str = 'tmp', prefix: Optional[str] = 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(root_path: Optional[str] = None, rel_path: str =  
                                                    'tmp', prefix: Optional[str] = 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(root_path: Optional[str] = None, rel_path: str =  
                                                    'tmp', prefix: Optional[str] = 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(root_path: Optional[str] = None, rel_path: str =  
                                                    'tmp', prefix: Optional[str] = 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(root_path: Optional[str] = None, rel_path: str =  
                                                    'tmp', prefix: Optional[str] = None,  
                                                    max_nb_chars: int = 10000, wrap_chars_after:  
Optional[int] = None, content: Optional[str] =  
None, **kwargs) → StringValue
```

```
faker_file.providers.zip_file.create_inner_webp_file(root_path: Optional[str] = None, rel_path: str =  
                                                    'tmp', prefix: Optional[str] = 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(root_path: Optional[str] = None, rel_path: str =  
                                                    'tmp', prefix: Optional[str] = None,  
                                                    data_columns: Optional[Dict[str, str]] = None,  
                                                    num_rows: int = 10, content: Optional[str] =  
None, **kwargs) → StringValue
```

Create inner XLSX file.

11.4.1.1.15 Module contents

11.4.1.1.2 faker_file.tests package

11.4.1.1.2.1 Submodules

11.4.1.1.2.2 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
    test_file
```

11.4.1.1.2.3 `faker_file.tests.test_providers` module

```
class faker_file.tests.test_providers.ProvidersTestCase(methodName='runTest')
    Bases: TestCase
    Providers test case.
    FAKER: Faker
    test_faker
    test_standalone_providers
    test_standalone_zip_file
```

11.4.1.1.2.4 Module contents

11.4.1.2 Submodules

11.4.1.3 `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] = {}
```

11.4.1.4 `faker_file.constants` module

11.4.1.5 `faker_file.content_generators` module

11.4.1.6 `faker_file.helpers` module

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

11.4.1.7 Module contents

INDICES AND TABLES

- `genindex`
- `modindex`
- `search`

PYTHON MODULE INDEX

f

- [faker_file](#), 46
- [faker_file.base](#), 45
- [faker_file.constants](#), 46
- [faker_file.helpers](#), 46
- [faker_file.providers](#), 44
 - [bin_file](#), 32
 - [csv_file](#), 33
 - [docx_file](#), 34
 - [ico_file](#), 35
 - [jpeg_file](#), 36
 - [pdf_file](#), 36
 - [png_file](#), 37
 - [pptx_file](#), 38
 - [svg_file](#), 39
 - [txt_file](#), 40
 - [webp_file](#), 40
 - [xlsx_file](#), 41
 - [zip_file](#), 42
- [faker_file.tests](#), 45
 - [test_django_integration](#), 44
 - [test_providers](#), 45

INDEX

B

`bin_file()` (*faker_file.providers.bin_file.BinFileProvider* method), 32

`BinFileProvider` (class in *faker_file.providers.bin_file*), 32

C

`create_inner_bin_file()` (in *faker_file.providers.zip_file*), 43

`create_inner_csv_file()` (in *faker_file.providers.zip_file*), 43

`create_inner_docx_file()` (in *faker_file.providers.zip_file*), 43

`create_inner_ico_file()` (in *faker_file.providers.zip_file*), 43

`create_inner_jpeg_file()` (in *faker_file.providers.zip_file*), 43

`create_inner_pdf_file()` (in *faker_file.providers.zip_file*), 43

`create_inner_png_file()` (in *faker_file.providers.zip_file*), 44

`create_inner_pptx_file()` (in *faker_file.providers.zip_file*), 44

`create_inner_svg_file()` (in *faker_file.providers.zip_file*), 44

`create_inner_txt_file()` (in *faker_file.providers.zip_file*), 44

`create_inner_webp_file()` (in *faker_file.providers.zip_file*), 44

`create_inner_xlsx_file()` (in *faker_file.providers.zip_file*), 44

`csv_file()` (*faker_file.providers.csv_file.CsvFileProvider* method), 33

`CsvFileProvider` (class in *faker_file.providers.csv_file*), 33

D

`data` (*faker_file.base.StringValue* attribute), 45

`DjangoIntegrationTestCase` (class in *faker_file.tests.test_django_integration*), 44

`docx_file()` (*faker_file.providers.docx_file.DocxFileProvider* method), 34

`DocxFileProvider` (class in *faker_file.providers.docx_file*), 34

E

`extension` (*faker_file.base.FileMixin* attribute), 45

module `extension` (*faker_file.providers.bin_file.BinFileProvider* attribute), 33

module `extension` (*faker_file.providers.csv_file.CsvFileProvider* attribute), 34

module `extension` (*faker_file.providers.docx_file.DocxFileProvider* attribute), 35

module `extension` (*faker_file.providers.ico_file.IcoFileProvider* attribute), 35

module `extension` (*faker_file.providers.jpeg_file.JpegFileProvider* attribute), 36

module `extension` (*faker_file.providers.pdf_file.PdfFileProvider* attribute), 37

module `extension` (*faker_file.providers.png_file.PngFileProvider* attribute), 37

module `extension` (*faker_file.providers.pptx_file.PptxFileProvider* attribute), 38

module `extension` (*faker_file.providers.svg_file.SvgFileProvider* attribute), 39

module `extension` (*faker_file.providers.txt_file.TxtFileProvider* attribute), 40

module `extension` (*faker_file.providers.webp_file.WebpFileProvider* attribute), 41

module `extension` (*faker_file.providers.xlsx_file.XlsxFileProvider* attribute), 41

`extension` (*faker_file.providers.zip_file.ZipFileProvider* attribute), 42

F

`FAKER` (*faker_file.tests.test_django_integration.DjangoIntegrationTestCase* attribute), 45

`FAKER` (*faker_file.tests.test_providers.ProvidersTestCase* attribute), 45

`faker_file` module, 46

`faker_file.base`

- module, 45
- `faker_file.constants`
 - module, 46
- `faker_file.helpers`
 - module, 46
- `faker_file.providers`
 - module, 44
- `faker_file.providers.bin_file`
 - module, 32
- `faker_file.providers.csv_file`
 - module, 33
- `faker_file.providers.docx_file`
 - module, 34
- `faker_file.providers.ico_file`
 - module, 35
- `faker_file.providers.jpeg_file`
 - module, 36
- `faker_file.providers.pdf_file`
 - module, 36
- `faker_file.providers.png_file`
 - module, 37
- `faker_file.providers.pptx_file`
 - module, 38
- `faker_file.providers.svg_file`
 - module, 39
- `faker_file.providers.txt_file`
 - module, 40
- `faker_file.providers.webp_file`
 - module, 40
- `faker_file.providers.xlsx_file`
 - module, 41
- `faker_file.providers.zip_file`
 - module, 42
- `faker_file.tests`
 - module, 45
- `faker_file.tests.test_django_integration`
 - module, 44
- `faker_file.tests.test_providers`
 - module, 45
- `FileMixin` (class in `faker_file.base`), 45
- `formats` (`faker_file.base.FileMixin` attribute), 45

G

- `generator` (`faker_file.base.FileMixin` attribute), 45

I

- `ico_file()` (`faker_file.providers.ico_file.IcoFileProvider` method), 35
- `IcoFileProvider` (class in `faker_file.providers.ico_file`), 35

J

- `jpeg_file()` (`faker_file.providers.jpeg_file.JpegFileProvider` method), 36

- `JpegFileProvider` (class in `faker_file.providers.jpeg_file`), 36

M

- module
 - `faker_file`, 46
 - `faker_file.base`, 45
 - `faker_file.constants`, 46
 - `faker_file.helpers`, 46
 - `faker_file.providers`, 44
 - `faker_file.providers.bin_file`, 32
 - `faker_file.providers.csv_file`, 33
 - `faker_file.providers.docx_file`, 34
 - `faker_file.providers.ico_file`, 35
 - `faker_file.providers.jpeg_file`, 36
 - `faker_file.providers.pdf_file`, 36
 - `faker_file.providers.png_file`, 37
 - `faker_file.providers.pptx_file`, 38
 - `faker_file.providers.svg_file`, 39
 - `faker_file.providers.txt_file`, 40
 - `faker_file.providers.webp_file`, 40
 - `faker_file.providers.xlsx_file`, 41
 - `faker_file.providers.zip_file`, 42
 - `faker_file.tests`, 45
 - `faker_file.tests.test_django_integration`, 44
 - `faker_file.tests.test_providers`, 45

N

- `numerify` (`faker_file.base.FileMixin` attribute), 45

P

- `pdf_file()` (`faker_file.providers.pdf_file.PdfFileProvider` method), 37
- `PdfFileProvider` (class in `faker_file.providers.pdf_file`), 36
- `png_file()` (`faker_file.providers.png_file.PngFileProvider` method), 37
- `PngFileProvider` (class in `faker_file.providers.png_file`), 37
- `pptx_file()` (`faker_file.providers.pptx_file.PptxFileProvider` method), 38
- `PptxFileProvider` (class in `faker_file.providers.pptx_file`), 38
- `ProvidersTestCase` (class in `faker_file.tests.test_providers`), 45

R

- `random_element` (`faker_file.base.FileMixin` attribute), 45

S

- `StringValue` (class in `faker_file.base`), 45

`svg_file()` (*faker_file.providers.svg_file.SvgFileProvider*
method), [39](#)
`SvgFileProvider` (class in
faker_file.providers.svg_file), [39](#)

T

`test_faker` (*faker_file.tests.test_providers.ProvidersTestCase*
attribute), [45](#)
`test_file` (*faker_file.tests.test_django_integration.DjangoIntegrationTestCase*
attribute), [45](#)
`test_standalone_providers`
(faker_file.tests.test_providers.ProvidersTestCase
attribute), [45](#)
`test_standalone_zip_file`
(faker_file.tests.test_providers.ProvidersTestCase
attribute), [45](#)
`txt_file()` (*faker_file.providers.txt_file.TxtFileProvider*
method), [40](#)
`TxtFileProvider` (class in *faker_file.providers.txt_file*),
[40](#)

W

`webp_file()` (*faker_file.providers.webp_file.WebpFileProvider*
method), [41](#)
`WebpFileProvider` (class in
faker_file.providers.webp_file), [40](#)
`wrap_text()` (in module *faker_file.helpers*), [46](#)

X

`xlsx_file()` (*faker_file.providers.xlsx_file.XlsxFileProvider*
method), [41](#)
`XlsxFileProvider` (class in
faker_file.providers.xlsx_file), [41](#)

Z

`zip_file()` (*faker_file.providers.zip_file.ZipFileProvider*
method), [42](#)
`ZipFileProvider` (class in *faker_file.providers.zip_file*),
[42](#)