

---

# **faker-file Documentation**

***Release 0.16.4***

**Artur Barseghyan <artur.barseghyan@gmail.com>**

**Jul 13, 2023**



# CONTENTS

<b>1</b>	<b>Prerequisites</b>	<b>3</b>
<b>2</b>	<b>Documentation</b>	<b>5</b>
<b>3</b>	<b>Online demos</b>	<b>7</b>
<b>4</b>	<b>Installation</b>	<b>9</b>
4.1	Latest stable version from PyPI . . . . .	9
4.2	Or development version from GitHub . . . . .	10
<b>5</b>	<b>Features</b>	<b>11</b>
5.1	Supported file types . . . . .	11
5.2	Additional providers . . . . .	12
5.3	Supported file storages . . . . .	12
<b>6</b>	<b>Usage examples</b>	<b>13</b>
6.1	With Faker . . . . .	13
6.2	With factory_boy . . . . .	13
6.2.1	upload/models.py . . . . .	13
6.2.2	upload/factories.py . . . . .	14
<b>7</b>	<b>File storages</b>	<b>15</b>
7.1	Usage example with storages . . . . .	15
7.1.1	FileSystemStorage example . . . . .	15
7.1.2	PathyFileSystemStorage example . . . . .	16
7.1.3	AWSS3Storage example . . . . .	16
<b>8</b>	<b>Testing</b>	<b>19</b>
<b>9</b>	<b>Writing documentation</b>	<b>21</b>
<b>10</b>	<b>License</b>	<b>23</b>
<b>11</b>	<b>Support</b>	<b>25</b>
<b>12</b>	<b>Author</b>	<b>27</b>
<b>13</b>	<b>Citation</b>	<b>29</b>
<b>14</b>	<b>Project documentation</b>	<b>31</b>
14.1	Quick start . . . . .	32
14.1.1	Installation . . . . .	32

14.1.2	Usage	32
14.1.2.1	With Faker	32
14.1.2.2	With factory_boy	34
14.1.2.2.1	upload/models.py	35
14.1.2.2.2	upload/factories.py	36
14.1.2.2.3	Usage example	38
14.2	Recipes	38
14.2.1	When using with Faker	38
14.2.1.1	Imports and initializations	38
14.2.1.2	Create a TXT file with static content	39
14.2.1.3	Create a DOCX file with dynamically generated content	39
14.2.1.4	Create a ZIP file consisting of TXT files with static content	39
14.2.1.5	Create a ZIP file consisting of 3 DOCX files with dynamically generated content	39
14.2.1.6	Create a ZIP file of 9 DOCX files with content generated from template	40
14.2.1.7	Create a nested ZIP file	40
14.2.1.8	Create a ZIP file with variety of different file types within	41
14.2.1.9	Another way to create a ZIP file with variety of different file types within	42
14.2.1.10	Create a EML file consisting of TXT files with static content	43
14.2.1.11	Create a EML file consisting of 3 DOCX files with dynamically generated content	43
14.2.1.12	Create a nested EML file	44
14.2.1.13	Create an EML file with variety of different file types within	44
14.2.1.14	Create a TXT file with static content	45
14.2.1.15	Create a DOCX file with dynamically generated content	45
14.2.1.16	Create a PDF file with predefined template containing dynamic fixtures	46
14.2.1.17	Create a DOCX file with table and image using <code>DynamicTemplate</code>	46
14.2.1.18	Create a ODT file with table and image using <code>DynamicTemplate</code>	47
14.2.1.19	Create a PDF using <i>reportlab</i> generator	50
14.2.1.20	Create a PDF using <i>pdfkit</i> generator	50
14.2.1.21	Create a MP3 file	50
14.2.1.22	Create a MP3 file by explicitly specifying MP3 generator class	50
14.2.1.22.1	Google Text-to-Speech	50
14.2.1.22.2	Microsoft Edge Text-to-Speech	51
14.2.1.23	Create a MP3 file with custom MP3 generator	52
14.2.1.24	Pick a random file from a directory given	53
14.2.1.25	File from path given	53
14.2.1.26	Generate a file of a certain size	53
14.2.1.26.1	BIN	53
14.2.1.26.2	TXT	54
14.2.1.27	Generate a lot of files using multiprocessing	54
14.2.1.27.1	Generate 100 DOCX files	54
14.2.1.27.2	Randomize the file format	54
14.2.1.28	Generating files from existing documents using NLP augmentation	56
14.2.1.29	Using <i>raw=True</i> features in tests	57
14.2.1.30	Create a HTML file predefined template	58
14.2.1.31	Working with storages	58
14.2.1.31.1	AWS S3 storage	58
14.2.1.31.2	SFTP storage	59
14.2.2	When using with Django (and factory_boy)	59
14.2.2.1	Basic example	59
14.2.2.1.1	Imaginary Django model	59
14.2.2.1.2	Correspondent <code>factory_boy</code> factory	60
14.2.2.2	Randomize provider choice	62
14.2.2.3	Use a different locale	63
14.2.2.4	Other Django usage examples	63

14.3	Creating PDF . . . . .	65
14.3.1	Building PDF using pdftk . . . . .	65
14.3.2	Building PDFs using reportlab . . . . .	67
14.4	Creating DOCX . . . . .	68
14.5	Creating ODT . . . . .	69
14.6	CLI . . . . .	71
14.6.1	List available provider options . . . . .	71
14.6.2	List options for a certain provider . . . . .	72
14.6.3	Generate a file using certain provider . . . . .	73
14.6.4	Shell auto-completion . . . . .	73
14.7	Security Policy . . . . .	73
14.7.1	Reporting a Vulnerability . . . . .	73
14.7.2	Supported Versions . . . . .	74
14.8	Contributor Covenant Code of Conduct . . . . .	74
14.8.1	Our Pledge . . . . .	74
14.8.2	Our Standards . . . . .	74
14.8.3	Enforcement Responsibilities . . . . .	75
14.8.4	Scope . . . . .	75
14.8.5	Enforcement . . . . .	75
14.8.6	Enforcement Guidelines . . . . .	75
14.8.6.1	1. Correction . . . . .	75
14.8.6.2	2. Warning . . . . .	75
14.8.6.3	3. Temporary Ban . . . . .	76
14.8.6.4	4. Permanent Ban . . . . .	76
14.8.7	Attribution . . . . .	76
14.9	Contributor guidelines . . . . .	76
14.9.1	Developer prerequisites . . . . .	76
14.9.1.1	pre-commit . . . . .	76
14.9.2	Code standards . . . . .	76
14.9.3	Requirements . . . . .	77
14.9.4	Virtual environment . . . . .	77
14.9.5	Documentation . . . . .	77
14.9.6	Testing . . . . .	77
14.9.7	Pull requests . . . . .	77
14.9.8	Questions . . . . .	78
14.9.9	Issues . . . . .	78
14.10	Release history and notes . . . . .	78
14.10.1	0.16.4 . . . . .	78
14.10.2	0.16.3 . . . . .	78
14.10.3	0.16.2 . . . . .	79
14.10.4	0.16.1 . . . . .	79
14.10.5	0.16 . . . . .	79
14.10.6	0.15.5 . . . . .	79
14.10.7	0.15.4 . . . . .	80
14.10.8	0.15.3 . . . . .	80
14.10.9	0.15.2 . . . . .	80
14.10.10	0.15.1 . . . . .	80
14.10.11	0.15 . . . . .	80
14.10.12	0.14.5 . . . . .	81
14.10.13	0.14.4 . . . . .	81
14.10.14	0.14.3 . . . . .	81
14.10.15	0.14.2 . . . . .	81
14.10.16	0.14.1 . . . . .	81
14.10.17	0.14 . . . . .	81

14.10.180.13	82
14.10.190.12.6	82
14.10.200.12.5	82
14.10.210.12.4	82
14.10.220.12.3	83
14.10.230.12.2	83
14.10.240.12.1	83
14.10.250.12	83
14.10.260.11.5	83
14.10.270.11.4	83
14.10.280.11.3	84
14.10.290.11.2	84
14.10.300.11.1	84
14.10.310.11	84
14.10.320.10.12	84
14.10.330.10.11	85
14.10.340.10.10	85
14.10.350.10.9	85
14.10.360.10.8	85
14.10.370.10.7	85
14.10.380.10.6	86
14.10.390.10.5	86
14.10.400.10.4	86
14.10.410.10.3	86
14.10.420.10.2	86
14.10.430.10.1	87
14.10.440.10	87
14.10.450.9.3	87
14.10.460.9.2	87
14.10.470.9.1	87
14.10.480.9	87
14.10.490.8	88
14.10.500.7	88
14.10.510.6	88
14.10.520.5	88
14.10.530.4	89
14.10.540.3	89
14.10.550.2	89
14.10.560.1	89
14.11 Package	90
14.11.1 faker_file package	90
14.11.1.1 Subpackages	90
14.11.1.1.1 faker_file.providers package	90
14.11.1.1.1.1 Subpackages	90
14.11.1.1.1.2 faker_file.providers.augment_file_from_dir package	90
14.11.1.1.1.3 Subpackages	90
14.11.1.1.1.4 faker_file.providers.augment_file_from_dir.augmenters package	90
14.11.1.1.1.5 Submodules	90
14.11.1.1.1.6 faker_file.providers.augment_file_from_dir.augmenters.nlpaug_augmenter module	90
14.11.1.1.1.7 Module contents	90
14.11.1.1.1.8 faker_file.providers.augment_file_from_dir.extractors package	90
14.11.1.1.1.9 Submodules	90

14.11.1.1.1.10	faker_file.providers.augment_file_from_dir.extractors.tika_extractor module . . . . .	90
14.11.1.1.1.11	Module contents . . . . .	90
14.11.1.1.1.12	Module contents . . . . .	90
14.11.1.1.1.13	faker_file.providers.base package . . . . .	90
14.11.1.1.1.14	Submodules . . . . .	90
14.11.1.1.1.15	faker_file.providers.base.mp3_generator module . . . . .	90
14.11.1.1.1.16	faker_file.providers.base.pdf_generator module . . . . .	91
14.11.1.1.1.17	faker_file.providers.base.text_augmenter module . . . . .	91
14.11.1.1.1.18	faker_file.providers.base.text_extractor module . . . . .	91
14.11.1.1.1.19	Module contents . . . . .	92
14.11.1.1.1.20	faker_file.providers.helpers package . . . . .	92
14.11.1.1.1.21	Submodules . . . . .	92
14.11.1.1.1.22	faker_file.providers.helpers.inner module . . . . .	92
14.11.1.1.1.23	Module contents . . . . .	107
14.11.1.1.1.24	faker_file.providers.mixins package . . . . .	107
14.11.1.1.1.25	Submodules . . . . .	107
14.11.1.1.1.26	faker_file.providers.mixins.image_mixin module . . . . .	107
14.11.1.1.1.27	faker_file.providers.mixins.tablular_data_mixin module . . . . .	107
14.11.1.1.1.28	Module contents . . . . .	108
14.11.1.1.1.29	faker_file.providers.mp3_file package . . . . .	108
14.11.1.1.1.30	Subpackages . . . . .	108
14.11.1.1.1.31	faker_file.providers.mp3_file.generators package . . . . .	108
14.11.1.1.1.32	Submodules . . . . .	108
14.11.1.1.1.33	faker_file.providers.mp3_file.generators.edge_tts_generator module . . . . .	108
14.11.1.1.1.34	faker_file.providers.mp3_file.generators.gtts_generator module . . . . .	109
14.11.1.1.1.35	Module contents . . . . .	109
14.11.1.1.1.36	Module contents . . . . .	109
14.11.1.1.1.37	faker_file.providers.pdf_file package . . . . .	111
14.11.1.1.1.38	Subpackages . . . . .	111
14.11.1.1.1.39	faker_file.providers.pdf_file.generators package . . . . .	111
14.11.1.1.1.40	Submodules . . . . .	111
14.11.1.1.1.41	faker_file.providers.pdf_file.generators.pdftk_generator module . . . . .	111
14.11.1.1.1.42	faker_file.providers.pdf_file.generators.reportlab_generator module . . . . .	114
14.11.1.1.1.43	Module contents . . . . .	116
14.11.1.1.1.44	Module contents . . . . .	116
14.11.1.1.1.45	Submodules . . . . .	118
14.11.1.1.1.46	faker_file.providers.bin_file module . . . . .	118
14.11.1.1.1.47	faker_file.providers.csv_file module . . . . .	119
14.11.1.1.1.48	faker_file.providers.docx_file module . . . . .	120
14.11.1.1.1.49	faker_file.providers.eml_file module . . . . .	122
14.11.1.1.1.50	faker_file.providers.epub_file module . . . . .	124
14.11.1.1.1.51	faker_file.providers.file_from_path module . . . . .	125
14.11.1.1.1.52	faker_file.providers.generic_file module . . . . .	126
14.11.1.1.1.53	faker_file.providers.ico_file module . . . . .	127
14.11.1.1.1.54	faker_file.providers.jpeg_file module . . . . .	128
14.11.1.1.1.55	faker_file.providers.odp_file module . . . . .	130
14.11.1.1.1.56	faker_file.providers.ods_file module . . . . .	131
14.11.1.1.1.57	faker_file.providers.odt_file module . . . . .	132
14.11.1.1.1.58	faker_file.providers.png_file module . . . . .	135
14.11.1.1.1.59	faker_file.providers.pptx_file module . . . . .	136
14.11.1.1.1.60	faker_file.providers.random_file_from_dir module . . . . .	137
14.11.1.1.1.61	faker_file.providers.rtf_file module . . . . .	138
14.11.1.1.1.62	faker_file.providers.svg_file module . . . . .	139

14.11.1.1.1.63	faker_file.providers.tar_file module . . . . .	140
14.11.1.1.1.64	faker_file.providers.txt_file module . . . . .	141
14.11.1.1.1.65	faker_file.providers.webp_file module . . . . .	142
14.11.1.1.1.66	faker_file.providers.xlsx_file module . . . . .	144
14.11.1.1.1.67	faker_file.providers.xml_file module . . . . .	145
14.11.1.1.1.68	faker_file.providers.zip_file module . . . . .	147
14.11.1.1.1.69	Module contents . . . . .	148
14.11.1.1.2	faker_file.storages package . . . . .	148
14.11.1.1.2.1	Submodules . . . . .	148
14.11.1.1.2.2	faker_file.storages.aws_s3 module . . . . .	148
14.11.1.1.2.3	faker_file.storages.azure_cloud_storage module . . . . .	149
14.11.1.1.2.4	faker_file.storages.base module . . . . .	149
14.11.1.1.2.5	faker_file.storages.cloud module . . . . .	150
14.11.1.1.2.6	faker_file.storages.filesystem module . . . . .	151
14.11.1.1.2.7	faker_file.storages.google_cloud_storage module . . . . .	151
14.11.1.1.2.8	Module contents . . . . .	152
14.11.1.1.3	faker_file.tests package . . . . .	152
14.11.1.1.3.1	Submodules . . . . .	152
14.11.1.1.3.2	faker_file.tests.data module . . . . .	152
14.11.1.1.3.3	faker_file.tests.helpers module . . . . .	152
14.11.1.1.3.4	faker_file.tests.test_augment_file_from_dir_provider module . . . . .	152
14.11.1.1.3.5	faker_file.tests.test_base module . . . . .	152
14.11.1.1.3.6	faker_file.tests.test_cli module . . . . .	152
14.11.1.1.3.7	faker_file.tests.test_data_integrity module . . . . .	153
14.11.1.1.3.8	faker_file.tests.test_django_integration module . . . . .	153
14.11.1.1.3.9	faker_file.tests.test_providers module . . . . .	153
14.11.1.1.3.10	faker_file.tests.test_sqlalchemy_integration module . . . . .	154
14.11.1.1.3.11	faker_file.tests.test_storages module . . . . .	154
14.11.1.1.3.12	faker_file.tests.texts module . . . . .	154
14.11.1.1.3.13	Module contents . . . . .	154
14.11.1.2	Submodules . . . . .	154
14.11.1.3	faker_file.base module . . . . .	154
14.11.1.4	faker_file.cli module . . . . .	155
14.11.1.5	faker_file.constants module . . . . .	155
14.11.1.6	faker_file.helpers module . . . . .	155
14.11.1.7	Module contents . . . . .	156
14.12	Indices and tables . . . . .	156

<b>Python Module Index</b>	<b>157</b>
----------------------------	------------

<b>Index</b>	<b>159</b>
--------------	------------



**Create files with fake data.** In many formats. With no efforts.



## 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*-, *Apache 2*- or *GPL* licensed. All licenses are mentioned below between the brackets.

- Core package requires Python 3.7, 3.8, 3.9, 3.10 or 3.11.
- [Faker](#) (*MIT*) is the only required dependency.
- [Django](#) (*BSD*) integration with [factory\\_boy](#) (*MIT*) has been tested with Django starting from version 2.2 to 4.2 (although only maintained versions of Django are currently being tested against).
- DOCX file support requires [python-docx](#) (*MIT*).
- EPUB file support requires [xml2epub](#) (*MIT*) and [Jinja2](#) (*BSD*).
- ICO, JPEG, PNG, SVG and WEBP files support requires [imgkit](#) (*MIT*) and [wkhtmltopdf](#) (*LGPLv3*).
- MP3 file support requires [gTTS](#) (*MIT*) or [edge-tts](#) (*GPLv3*).
- PDF file support requires either combination of [pdftkit](#) (*MIT*) and [wkhtmltopdf](#) (*LGPLv3*), or [reportlab](#) (*BSD*).
- PPTX file support requires [python-pptx](#) (*MIT*).
- ODP file support requires [odfpy](#) (*Apache 2*).
- ODS file support requires [tablib](#) (*MIT*) and [odfpy](#) (*Apache 2*).
- ODT file support requires [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*).
- `SFTPStorage` storage support requires [paramiko](#) (*LGPLv2.1*).
- `AugmentFileFromDirProvider` provider requires [nlpaug](#) (*MIT*), [PyTorch](#) (*BSD*), [transformers](#) (*Apache 2*), [numpy](#) (*BSD*), [pandas](#) (*BSD*), [tika](#) (*Apache 2*) and [Apache Tika](#) (*Apache 2*).



## DOCUMENTATION

- Documentation is available on [Read the Docs](#).
- For bootstrapping check the [Quick start](#).
- For various ready to use code examples see the [Recipes](#).
- For tips on PDF creation see [Creating PDF](#).
- For tips on DOCX creation see [Creating DOCX](#).
- For tips on ODT creation see [Creating ODT](#).
- For CLI options see the [CLI](#).
- For guidelines on contributing check the [Contributor guidelines](#).



## ONLINE DEMOS

Check the demo(s):

- [REST API demo](#) (based on [faker-file-api](#) REST API)
- [UI frontend demo](#) (based on [faker-file-ui](#) UI frontend)
- [WASM frontend demo](#) (based on [faker-file-wasm](#) WASM frontend)





## INSTALLATION

### 4.1 Latest stable version from PyPI

#### With all dependencies

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

#### Only core

```
pip install faker-file
```

#### With most common dependencies

*Everything, except ML libraries which are required for data augmentation only*

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

#### With DOCX support

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

#### With EPUB support

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

#### With images support

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

#### With PDF support

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

#### With MP3 support

```
pip install faker-file[mp3]
```

#### With XLSX support

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

#### With ODS support

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

**With ODT support**

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

**With data augmentation support**

```
pip install faker-file[data-augmentation]
```

**With GoogleCloudStorage support**

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

**With AzureCloudStorage support**

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

**With AWSS3Storage support**

```
pip install faker-file[s3]
```

## 4.2 Or development version from GitHub

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

## FEATURES

### 5.1 Supported file types

- BIN
- CSV
- DOCX
- EML
- EPUB
- ICO
- JPEG
- MP3
- ODS
- ODT
- ODP
- PDF
- PNG
- RTF
- PPTX
- SVG
- TAR
- TXT
- WEBP
- XLSX
- XML
- ZIP

## 5.2 Additional providers

- `AugmentFileFromDirProvider`: Make an augmented copy of randomly picked file from given directory. The following types are supported : DOCX, EML, EPUB, ODT, PDF, RTF and TXT.
- `GenericFileProvider`: Create files in any format from raw bytes or a predefined template.
- `RandomFileFromDirProvider`: Pick a random file from given directory.
- `FileFromPathProvider`: File from given path.

## 5.3 Supported file storages

- Native file system storage
- AWS S3 storage
- Azure Cloud Storage
- Google Cloud Storage
- SFTP storage

## USAGE EXAMPLES

### 6.1 With Faker

One way

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

FAKER = Faker()

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

If you just need bytes back (instead of creating the file), provide the `raw=True` argument (works with all provider classes and inner functions):

```
raw = TxtFileProvider(FAKER).txt_file(raw=True)
```

Or another

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

FAKER = Faker()
FAKER.add_provider(TxtFileProvider)

file = FAKER.txt_file()
```

If you just need bytes back:

```
raw = FAKER.txt_file(raw=True)
```

### 6.2 With factory\_boy

#### 6.2.1 upload/models.py

```
from django.db import models

class Upload(models.Model):
```

(continues on next page)

(continued from previous page)

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

### 6.2.2 upload/factories.py

Note, that when using `faker-file` with Django and native file system storages, you need to pass your `MEDIA_ROOT` setting as `root_path` value to the chosen file storage as show below.

```
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 faker_file.storages.filesystem import FileSystemStorage  
  
from upload.models import Upload  
  
FS_STORAGE = FileSystemStorage(  
    root_path=settings.MEDIA_ROOT,  
    rel_path="tmp"  
)  
factory.Faker.add_provider(DocxFileProvider)  
  
class UploadFactory(DjangoModelFactory):  
  
    # ...  
    file = Faker("docx_file", storage=FS_STORAGE)  
  
    class Meta:  
        model = Upload
```

## FILE 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.
- `SFTPStorage`: Requires `paramiko` and related dependencies.

## 7.1 Usage example with storages

### 7.1.1 *FileSystemStorage* example

Native file system storage. Does not have dependencies.

- `root_path`: Path to the root directory. Given the example of Django, this would be the path to the `MEDIA_ROOT` directory. It's important to know, that `root_path` will not be embedded into the string representation of the file. Only `rel_path` will.
- `rel_path`: Relative path from the root directory. Given the example of Django, this would be the rest of the path to the file.

```
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=tempfile.gettempdir(), # Use settings.MEDIA_ROOT for Django
    rel_path="tmp",
)

FAKER = Faker()

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

(continues on next page)

(continued from previous page)

```
FS_STORAGE.exists(file)
```

### 7.1.2 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="bucket_name",
    root_path="tmp"
    rel_path="sub-tmp",
)

FAKER = Faker()

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

PATHY_FS_STORAGE.exists(file)
```

### 7.1.3 AWSS3Storage example

AWS S3 storage. Requires pathy and boto3.

```
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",
    root_path="tmp", # Optional
    rel_path="sub-tmp", # Optional
    # Credentials are optional too. If your AWS credentials are properly
    # set in the ~/.aws/credentials, you don't need to send them
    # explicitly.
    credentials={
        "key_id": "YOUR KEY ID",
        "key_secret": "YOUR KEY SECRET"
    },
)

FAKER = Faker()

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

(continues on next page)



(continued from previous page)

```
S3_STORAGE.exists(file)
```



## TESTING

Simply type:

```
pytest -vrx
```

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

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



---

CHAPTER  
**TWELVE**

---

**AUTHOR**

Artur Barseghyan <[artur.barseghyan@gmail.com](mailto:artur.barseghyan@gmail.com)>



## CITATION

Please, use the following entry when citing `faker-file` in your research:

```
@software{faker-file,  
  author = {Artur Barseghyan},  
  title = {faker-file: Create files with fake data. In many formats. With no efforts.},  
  year = {2023},  
  publisher = {GitHub},  
  journal = {GitHub repository},  
  howpublished = {https://github.com/barseghyanartur/faker-file},  
}
```



## PROJECT DOCUMENTATION

Contents:

### Table of Contents

- *faker-file*
  - *Prerequisites*
  - *Documentation*
  - *Online demos*
  - *Installation*
    - \* *Latest stable version from PyPI*
    - \* *Or development version from GitHub*
  - *Features*
    - \* *Supported file types*
    - \* *Additional providers*
    - \* *Supported file storages*
  - *Usage examples*
    - \* *With Faker*
    - \* *With factory\_boy*
      - *upload/models.py*
      - *upload/factories.py*
  - *File storages*
    - \* *Usage example with storages*
      - *FileSystemStorage example*
      - *PathyFileSystemStorage example*
      - *AWSS3Storage example*
  - *Testing*
  - *Writing documentation*
  - *License*

- *Support*
- *Author*
- *Citation*
- *Project documentation*

## 14.1 Quick start

### 14.1.1 Installation

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

### 14.1.2 Usage

#### 14.1.2.1 With Faker

##### Imports and initialization

```
from faker import Faker
from faker_file.providers.augment_file_from_dir import AugmentFileFromDirProvider
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.eml_file import EmlFileProvider
from faker_file.providers.epub_file import EpubFileProvider
from faker_file.providers.ico_file import IcoFileProvider
from faker_file.providers.jpeg_file import JpegFileProvider
from faker_file.providers.mp3_file import Mp3FileProvider
from faker_file.providers.odp_file import OdpFileProvider
from faker_file.providers.ods_file import OdsFileProvider
from faker_file.providers.odt_file import OdtFileProvider
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.random_file_from_dir import RandomFileFromDirProvider
from faker_file.providers.rtf_file import RtfFileProvider
from faker_file.providers.svg_file import SvgFileProvider
from faker_file.providers.tar_file import TarFileProvider
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(AugmentFileFromDirProvider)
FAKER.add_provider(BinFileProvider)
FAKER.add_provider(CsvFileProvider)
FAKER.add_provider(DocxFileProvider)
```

(continues on next page)



(continued from previous page)

```

FAKER.add_provider(EmlFileProvider)
FAKER.add_provider(EpubFileProvider)
FAKER.add_provider(IcoFileProvider)
FAKER.add_provider(JpegFileProvider)
FAKER.add_provider(Mp3FileProvider)
FAKER.add_provider(OdpFileProvider)
FAKER.add_provider(OdsFileProvider)
FAKER.add_provider(OdtFileProvider)
FAKER.add_provider(PdfFileProvider)
FAKER.add_provider(PngFileProvider)
FAKER.add_provider(PptxFileProvider)
FAKER.add_provider(RandomFileFromDirProvider)
FAKER.add_provider(RtfFileProvider)
FAKER.add_provider(SvgFileProvider)
FAKER.add_provider(TarFileProvider)
FAKER.add_provider(TxtFileProvider)
FAKER.add_provider(WebpFileProvider)
FAKER.add_provider(XlsxFileProvider)
FAKER.add_provider(ZipFileProvider)

```

### Usage examples

```

augmented_file = FAKER.augment_file_from_dir(source_dir_path="/path/to/source/",)
bin_file = FAKER.bin_file()
csv_file = FAKER.csv_file()
docx_file = FAKER.docx_file()
eml_file = FAKER.eml_file()
epub_file = FAKER.epub_file()
ico_file = FAKER.ico_file()
jpeg_file = FAKER.jpeg_file()
mp3_file = FAKER.mp3_file()
odp_file = FAKER.odp_file()
ods_file = FAKER.ods_file()
odt_file = FAKER.odt_file()
pdf_file = FAKER.pdf_file()
png_file = FAKER.png_file()
pptx_file = FAKER.pptx_file()
random_file = FAKER.random_file_from_dir(source_dir_path="/path/to/source/",)
rtf_file = FAKER.rtf_file()
svg_file = FAKER.svg_file()
tar_file = FAKER.tar_file()
txt_file = FAKER.txt_file()
webp_file = FAKER.webp_file()
xlsx_file = FAKER.xlsx_file()
zip_file = FAKER.zip_file()

```

If you just need bytes back (instead of creating the file), provide the `raw=True` argument (works with all provider classes and inner functions):

```

augmented_raw = FAKER.augment_file_from_dir(
    source_dir_path="/path/to/source/",
    raw=True,

```

(continues on next page)

(continued from previous page)

```

)
bin_raw = FAKER.bin_file(raw=True)
csv_raw = FAKER.csv_file(raw=True)
docx_raw = FAKER.docx_file(raw=True)
eml_raw = FAKER.eml_file(raw=True)
epub_raw = FAKER.epub_file(raw=True)
ico_raw = FAKER.ico_file(raw=True)
jpeg_raw = FAKER.jpeg_file(raw=True)
mp3_raw = FAKER.mp3_file(raw=True)
odp_raw = FAKER.odp_file(raw=True)
ods_raw = FAKER.ods_file(raw=True)
odt_raw = FAKER.odt_file(raw=True)
pdf_raw = FAKER.pdf_file(raw=True)
png_raw = FAKER.png_file(raw=True)
pptx_raw = FAKER.pptx_file(raw=True)
random_raw = FAKER.random_file_from_dir(
    source_dir_path="/path/to/source/",
    raw=True,
)
rtf_raw = FAKER.rtf_file(raw=True)
svg_raw = FAKER.svg_file(raw=True)
tar_raw = FAKER.tar_file(raw=True)
txt_raw = FAKER.txt_file(raw=True)
webp_raw = FAKER.webp_file(raw=True)
xlsx_raw = FAKER.xlsx_file(raw=True)
zip_raw = FAKER.zip_file(raw=True)

```

#### 14.1.2.2 With factory\_boy

##### Imports and initialization

```

from factory import Faker

from faker_file.providers.augment_file_from_dir import AugmentFileFromDirProvider
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.eml_file import EmlFileProvider
from faker_file.providers.epub_file import EpubFileProvider
from faker_file.providers.ico_file import IcoFileProvider
from faker_file.providers.jpeg_file import JpegFileProvider
from faker_file.providers.mp3_file import Mp3FileProvider
from faker_file.providers.odp_file import OdpFileProvider
from faker_file.providers.ods_file import OdsFileProvider
from faker_file.providers.odt_file import OdtFileProvider
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.random_file_from_dir import RandomFileFromDirProvider
from faker_file.providers.rtf_file import RtfFileProvider
from faker_file.providers.svg_file import SvgFileProvider

```

(continues on next page)

(continued from previous page)

```

from faker_file.providers.tar_file import TarFileProvider
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.add_provider(AugmentFileFromDirProvider)
Faker.add_provider(BinFileProvider)
Faker.add_provider(CsvFileProvider)
Faker.add_provider(DocxFileProvider)
Faker.add_provider(EmlFileProvider)
Faker.add_provider(EpubFileProvider)
Faker.add_provider(IcoFileProvider)
Faker.add_provider(JpegFileProvider)
Faker.add_provider(Mp3FileProvider)
Faker.add_provider(OdpFileProvider)
Faker.add_provider(OdsFileProvider)
Faker.add_provider(OdtFileProvider)
Faker.add_provider(PdfFileProvider)
Faker.add_provider(PngFileProvider)
Faker.add_provider(PptxFileProvider)
Faker.add_provider(RandomFileFromDirProvider)
Faker.add_provider(RtfFileProvider)
Faker.add_provider(SvgFileProvider)
Faker.add_provider(TarFileProvider)
Faker.add_provider(TxtFileProvider)
Faker.add_provider(WebpFileProvider)
Faker.add_provider(XlsxFileProvider)
Faker.add_provider(ZipFileProvider)

```

#### 14.1.2.2.1 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)

    # File
    file = models.FileField(null=True)

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

    def __str__(self):
        return self.name

```

#### 14.1.2.2.2 upload/factories.py

```
from django.conf import settings

from factory import Faker
from factory.django import DjangoModelFactory

from factory import Faker

# Import all needed providers
from faker_file.providers.augment_file_from_dir import (
    AugmentFileFromDirProvider,
)
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.eml_file import EmlFileProvider
from faker_file.providers.epub_file import EpubFileProvider
from faker_file.providers.ico_file import IcoFileProvider
from faker_file.providers.jpeg_file import JpegFileProvider
from faker_file.providers.mp3_file import Mp3FileProvider
from faker_file.providers.odp_file import OdpFileProvider
from faker_file.providers.ods_file import OdsFileProvider
from faker_file.providers.odt_file import OdtFileProvider
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.random_file_from_dir import (
    RandomFileFromDirProvider,
)
from faker_file.providers.rtf_file import RtfFileProvider
from faker_file.providers.svg_file import SvgFileProvider
from faker_file.providers.tar_file import TarFileProvider
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

# Import file storage, because we need to customize things in
# order for it to work with Django.
from faker_file.storages.filesystem import FileSystemStorage

from upload.models import Upload

# Add all needed providers
Faker.add_provider(AugmentFileFromDirProvider)
Faker.add_provider(BinFileProvider)
Faker.add_provider(CsvFileProvider)
Faker.add_provider(DocxFileProvider)
Faker.add_provider(EmlFileProvider)
Faker.add_provider(EpubFileProvider)
Faker.add_provider(IcoFileProvider)
```

(continues on next page)

(continued from previous page)

```

Faker.add_provider(JpegFileProvider)
Faker.add_provider(Mp3FileProvider)
Faker.add_provider(OdpFileProvider)
Faker.add_provider(OdsFileProvider)
Faker.add_provider(OdtFileProvider)
Faker.add_provider(PdfFileProvider)
Faker.add_provider(PngFileProvider)
Faker.add_provider(PptxFileProvider)
Faker.add_provider(RandomFileFromDirProvider)
Faker.add_provider(RtfFileProvider)
Faker.add_provider(SvgFileProvider)
Faker.add_provider(TarFileProvider)
Faker.add_provider(TxtFileProvider)
Faker.add_provider(WebpFileProvider)
Faker.add_provider(XlsxFileProvider)
Faker.add_provider(ZipFileProvider)

# Define a file storage.
STORAGE = FileSystemStorage(
    root_path=settings.MEDIA_ROOT,
    rel_path="tmp"
)

# Define the upload factory
class UploadFactory(DjangoModelFactory):
    """Upload factory."""

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

    class Meta:
        model = Upload

    class Params:
        bin_file = Trait(file=Faker("bin_file", storage=STORAGE))
        csv_file = Trait(file=Faker("csv_file", storage=STORAGE))
        docx_file = Trait(file=Faker("docx_file", storage=STORAGE))
        eml_file = Trait(file=Faker("eml_file", storage=STORAGE))
        epub_file = Trait(file=Faker("epub_file", storage=STORAGE))
        ico_file = Trait(file=Faker("ico_file", storage=STORAGE))
        jpeg_file = Trait(file=Faker("jpeg_file", storage=STORAGE))
        mp3_file = Trait(file=Faker("mp3_file", storage=STORAGE))
        odp_file = Trait(file=Faker("odp_file", storage=STORAGE))
        ods_file = Trait(file=Faker("ods_file", storage=STORAGE))
        odt_file = Trait(file=Faker("odt_file", storage=STORAGE))
        pdf_file = Trait(file=Faker("pdf_file", storage=STORAGE))
        png_file = Trait(file=Faker("png_file", storage=STORAGE))
        pptx_file = Trait(file=Faker("pptx_file", storage=STORAGE))
        rtf_file = Trait(file=Faker("rtf_file", storage=STORAGE))
        svg_file = Trait(file=Faker("svg_file", storage=STORAGE))
        tar_file = Trait(file=Faker("tar_file", storage=STORAGE))
        txt_file = Trait(file=Faker("txt_file", storage=STORAGE))

```

(continues on next page)

(continued from previous page)

```
webp_file = Trait(file=Faker("webp_file", storage=STORAGE))
xlsx_file = Trait(file=Faker("xlsx_file", storage=STORAGE))
zip_file = Trait(file=Faker("zip_file", storage=STORAGE))
```

#### 14.1.2.2.3 Usage example

```
UploadFactory(bin_file=True) # Upload with BIN file
UploadFactory(docx_file=True) # Upload with DOCX file
UploadFactory(jpeg_file=True) # Upload with JPEG file
UploadFactory(zip_file=True) # Upload with ZIP file
```

## 14.2 Recipes

### 14.2.1 When using with Faker

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

#### 14.2.1.1 Imports and initializations

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")
```

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)
```

(continues on next page)

(continued from previous page)

```
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.

#### 14.2.1.2 Create a TXT file with static content

- Content of the file is Lorem ipsum.

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

#### 14.2.1.3 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,
)
```

#### 14.2.1.4 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={"create_inner_file_args": {"content": "Lorem ipsum"}}
)
```

#### 14.2.1.5 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.helpers.inner 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",
    }
)
```

#### 14.2.1.6 Create a ZIP file of 9 DOCX files with content generated from template

- 9 DOCX files in the ZIP archive.
- Content is generated dynamically from given template.

```
from faker_file.providers.helpers.inner import create_inner_docx_file

TEMPLATE = "Hey {{name}},\n{{text}},\nBest regards\n{{name}}"

file = ZipFileProvider(FAKER).zip_file(
    options={
        "count": 9,
        "create_inner_file_func": create_inner_docx_file,
        "create_inner_file_args": {
            "content": TEMPLATE,
        },
    }
)
```

#### 14.2.1.7 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.helpers.inner import (
    create_inner_docx_file,
    create_inner_zip_file,
)
```

(continues on next page)



(continued from previous page)

```

file = ZipFileProvider(FAKER).zip_file(
    prefix="nested_level_0_",
    options={
        "create_inner_file_func": create_inner_zip_file,
        "create_inner_file_args": {
            "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,
                    }
                }
            }
        },
    },
)

```

#### 14.2.1.8 Create a ZIP file with variety of different file types within

- 50 files in the ZIP archive (limited to DOCX, EPUB and TXT types).
- Content is generated dynamically.
- Prefix the filename of the archive itself with zzz\_archive\_.
- Inside the ZIP, put all files in directory zzz.

```

from faker import Faker
from faker_file.providers.helpers.inner import (
    create_inner_docx_file,
    create_inner_epub_file,
    create_inner_txt_file,
    fuzzy_choice_create_inner_file,
)
from faker_file.providers.zip_file import ZipFileProvider
from faker_file.storages.filesystem import FileSystemStorage

FAKER = Faker()
STORAGE = FileSystemStorage()

kwargs = {"storage": STORAGE, "generator": FAKER}
file = ZipFileProvider(FAKER).zip_file(
    prefix="zzz_archive_",
    options={
        "count": 50,
        "create_inner_file_func": fuzzy_choice_create_inner_file,
        "create_inner_file_args": {
            "func_choices": [
                (create_inner_docx_file, kwargs),
                (create_inner_epub_file, kwargs),
            ]
        }
    }
)

```

(continues on next page)

(continued from previous page)

```
        (create_inner_txt_file, kwargs),
    ],
    },
    "directory": "zzz",
}
)
```

#### 14.2.1.9 Another way to create a ZIP file with variety of different file types within

- 3 files in the ZIP archive (1 DOCX, and 2 XML types).
- Content is generated dynamically.
- Filename of the archive itself is `alice-looking-through-the-glass.zip`.
- Files inside the archive have fixed name (passed with `basename` argument).

```
from faker import Faker
from faker_file.providers.helpers.inner import (
    create_inner_docx_file,
    create_inner_xml_file,
    list_create_inner_file,
)
from faker_file.providers.zip_file import ZipFileProvider
from faker_file.storages.filesystem import FileSystemStorage

FAKER = Faker()
STORAGE = FileSystemStorage()

kwargs = {"storage": STORAGE, "generator": FAKER}
file = ZipFileProvider(FAKER).zip_file(
    basename="alice-looking-through-the-glass",
    options={
        "create_inner_file_func": list_create_inner_file,
        "create_inner_file_args": {
            "func_list": [
                (create_inner_docx_file, {"basename": "doc"}),
                (create_inner_xml_file, {"basename": "doc_metadata"}),
                (create_inner_xml_file, {"basename": "doc_isbn"}),
            ],
        },
    },
)
)
```

Note, that `count` argument (not shown in the example, but commonly accepted by inner functions) will be simply ignored here.

#### 14.2.1.10 Create a EML file consisting of TXT files with static content

- 5 TXT files in the EML email (default value is 5).
- Content of all files is Lorem ipsum.

```
from faker import Faker
from faker_file.providers.eml_file import EmlFileProvider

FAKER = Faker()

file = EmlFileProvider(FAKER).eml_file(
    options={"create_inner_file_args": {"content": "Lorem ipsum"}}
)
```

#### 14.2.1.11 Create a EML file consisting of 3 DOCX files with dynamically generated content

- 3 DOCX files in the EML email.
- Content is generated dynamically.
- Content is limited to 1024 chars.
- Prefix the filenames in email with xxx\_.
- Prefix the filename of the email itself with zzz.

```
from faker import Faker
from faker_file.providers.eml_file import EmlFileProvider
from faker_file.providers.helpers.inner import create_inner_docx_file

FAKER = Faker()

file = EmlFileProvider(FAKER).eml_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,
        },
    },
)
```

#### 14.2.1.12 Create a nested EML file

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

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

```
from faker import Faker
from faker_file.providers.eml_file import EmlFileProvider
from faker_file.providers.helpers.inner import (
    create_inner_docx_file,
    create_inner_eml_file,
)

FAKER = Faker()

file = EmlFileProvider(FAKER).eml_file(
    prefix="nested_level_0_",
    options={
        "create_inner_file_func": create_inner_eml_file,
        "create_inner_file_args": {
            "prefix": "nested_level_1_",
            "options": {
                "create_inner_file_func": create_inner_eml_file,
                "create_inner_file_args": {
                    "prefix": "nested_level_2_",
                    "options": {
                        "create_inner_file_func": create_inner_docx_file,
                    }
                }
            }
        },
    },
)
```

#### 14.2.1.13 Create an EML file with variety of different file types within

- 10 files in the EML file (limited to DOCX, EPUB and TXT types).
- Content is generated dynamically.
- Prefix the filename of the EML itself with `zzz`.

```
from faker import Faker
from faker_file.providers.helpers.inner import (
    create_inner_docx_file,
    create_inner_epub_file,
    create_inner_txt_file,
```

(continues on next page)

(continued from previous page)

```

        fuzzy_choice_create_inner_file,
    )
    from faker_file.providers.eml_file import EmlFileProvider
    from faker_file.storages.filesystem import FileSystemStorage

    FAKER = Faker()
    STORAGE = FileSystemStorage()

    kwargs = {"storage": STORAGE, "generator": FAKER}

    file = EmlFileProvider(FAKER).eml_file(
        prefix="zzz",
        options={
            "count": 10,
            "create_inner_file_func": fuzzy_choice_create_inner_file,
            "create_inner_file_args": {
                "func_choices": [
                    (create_inner_docx_file, kwargs),
                    (create_inner_epub_file, kwargs),
                    (create_inner_txt_file, kwargs),
                ],
            },
        },
    )

```

#### 14.2.1.14 Create a TXT file with static content

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

#### 14.2.1.15 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,
)

```

#### 14.2.1.16 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}}

Address: {{address}}

Best regards,

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

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

#### 14.2.1.17 Create a DOCX file with table and image using DynamicTemplate

When pre-defined templating and dynamic fixtures are not enough and full control is needed, you can use DynamicTemplate wrapper. It takes a list of content modifiers (tuples): (func: Callable, kwargs: dict). Each callable should accept the following arguments:

- provider: Faker Generator instance or Faker instance.
- document: Document instance. Implementation specific.
- data: Dictionary. Used primarily for observability.
- counter: Integer. Index number of the content modifier.
- **\*\*kwargs**: Dictionary. Useful to pass implementation-specific arguments.

The following example shows how to generate a DOCX file with table and image.

```
from io import BytesIO

from faker import Faker
from faker_file.base import DynamicTemplate
from faker_file.providers.docx_file import DocxFileProvider
from faker_file.providers.jpeg_file import JpegFileProvider

def docx_add_table(provider, document, data, counter, **kwargs):
    """Callable responsible for the table generation."""
    table = document.add_table(
```

(continues on next page)

(continued from previous page)

```

        kwargs.get("rows", 3),
        kwargs.get("cols", 4),
    )
    # Modifications of `data` is not required for generation
    # of the file, but is useful for when you want to get
    # the text content of the file.
    data.setdefault("content_modifiers", {})
    data["content_modifiers"].setdefault("add_table", {})
    data["content_modifiers"]["add_table"].setdefault(counter, [])

    for row in table.rows:
        for cell in row.cells:
            text = provider.generator.paragraph()
            cell.text = text
            # Useful when you want to get the text content of the file.
            data["content_modifiers"]["add_table"][counter].append(text)
            data["content"] += ("\r\n" + text)

def docx_add_picture(provider, document, data, counter, **kwargs):
    """Callable responsible for the picture generation."""
    jpeg_file = JpegFileProvider(provider.generator).jpeg_file(raw=True)
    document.add_picture(BytesIO(jpeg_file))

    # Modifications of `data` is not required for generation
    # of the file, but is useful for when you want to get
    # the text content of the file.
    data.setdefault("content_modifiers", {})
    data["content_modifiers"].setdefault("add_picture", {})
    data["content_modifiers"]["add_picture"].setdefault(counter, [])
    data["content_modifiers"]["add_picture"][counter].append(
        jpeg_file.data["content"]
    )
    data["content"] += ("\r\n" + jpeg_file.data["content"])

file = DocxFileProvider(Faker()).docx_file(
    content=DynamicTemplate([(docx_add_table, {}), (docx_add_picture, {})])
)

```

#### 14.2.1.18 Create a ODT file with table and image using DynamicTemplate

Similarly to previous section, the following example shows how to generate an ODT file with table and image.

```

from faker import Faker
from faker_file.providers.odt_file import OdtFileProvider
from faker_file.base import DynamicTemplate
from faker_file.providers.jpeg_file import JpegFileProvider
from odf.draw import Frame, Image
from odf.style import (
    Style, TextProperties,

```

(continues on next page)

(continued from previous page)

```

    TableColumnProperties,
    TableRowProperties,
    TableCellProperties,
    GraphicProperties,
)
from odf.table import Table, TableRow, TableCell, TableColumn
from odf.text import P

FAKER = Faker()

def odt_add_table(provider, document, data, counter, **kwargs):
    """Callable responsible for the table generation."""
    table = Table()
    rows = kwargs.get("rows", 3)
    cols = kwargs.get("cols", 4)
    table_col_style = Style(name="TableColumn", family="table-column")
    table_col_style.addElement(
        TableColumnProperties(columnwidth="2cm")
    )
    document.automaticstyles.addElement(table_col_style)

    table_row_style = Style(name="TableRow", family="table-row")
    table_row_style.addElement(TableRowProperties(rowheight="1cm"))
    document.automaticstyles.addElement(table_row_style)

    # Modifications of `data` is not required for generation
    # of the file, but is useful for when you want to get
    # the text content of the file.
    data.setdefault("content_modifiers", {})
    data["content_modifiers"].setdefault("add_table", {})
    data["content_modifiers"]["add_table"].setdefault(counter, [])

    table_cell_style = Style(name="TableCell", family="table-cell")
    table_cell_style.addElement(
        TableCellProperties(
            padding="0.1cm", border="0.05cm solid #000000"
        )
    )
    document.automaticstyles.addElement(table_cell_style)

    # Create table
    table = Table()
    for i in range(rows):
        table.addElement(TableColumn(stylename=table_col_style))

    for row in range(cols):
        tr = TableRow(stylename=table_row_style)
        table.addElement(tr)
        for col in range(4):
            tc = TableCell(stylename=table_cell_style)
            tr.addElement(tc)

```

(continues on next page)



(continued from previous page)

```

        text = provider.generator.paragraph()
        p = P(text=text)
        tc.addElement(p)
        # Useful when you want to get the text content of the file.
        data["content_modifiers"]["add_table"][counter].append(text)
        data["content"] += "\r\n" + text

    document.text.addElement(table)

def odt_add_picture(provider, document, data, counter, **kwargs):
    """Callable responsible for the picture generation."""
    width = kwargs.get("width", "10cm")
    height = kwargs.get("height", "5cm")
    paragraph = P()
    document.text.addElement(paragraph)
    jpeg_file = JpegFileProvider(provider.generator).jpeg_file()
    image_data = jpeg_file.data["content"]
    image_frame = Frame(
        width=width,
        height=height,
        x="56pt",
        y="56pt",
        anchortype="paragraph",
    )
    href = document.addPicture(jpeg_file.data["filename"])
    image_frame.addElement(Image(href=href))
    paragraph.addElement(image_frame)

    # Modifications of `data` is not required for generation
    # of the file, but is useful for when you want to get
    # the text content of the file.
    data["content"] += "\r\n" + jpeg_file.data["content"]
    data.setdefault("content_modifiers", {})
    data["content_modifiers"].setdefault("add_picture", {})
    data["content_modifiers"]["add_picture"].setdefault(counter, [])
    data["content_modifiers"]["add_picture"][counter].append(
        jpeg_file.data["content"]
    )

file = OdtFileProvider(FAKER).odt_file(
    content=DynamicTemplate([(odt_add_table, {}), (odt_add_picture, {})])
)

```

#### 14.2.1.19 Create a PDF using *reportlab* generator

```
from faker_file.providers.pdf_file.generators.reportlab_generator import (
    ReportlabPdfGenerator,
)

file = FAKER.pdf_file(pdf_generator_cls=ReportlabPdfGenerator)
```

#### 14.2.1.20 Create a PDF using *pdftk* generator

Note, that at the moment, *pdftk* is the default generator. However, you could set it explicitly as follows:

```
from faker_file.providers.pdf_file.generators.pdftk_generator import (
    PdftkPdfGenerator,
)

file = FAKER.pdf_file(pdf_generator_cls=PdftkPdfGenerator)
```

#### 14.2.1.21 Create a MP3 file

```
file = FAKER.mp3_file()
```

#### 14.2.1.22 Create a MP3 file by explicitly specifying MP3 generator class

##### 14.2.1.22.1 Google Text-to-Speech

```
from faker import Faker
from faker_file.providers.mp3_file import Mp3FileProvider
from faker_file.providers.mp3_file.generators.gtts_generator import (
    GttsMp3Generator,
)

FAKER = Faker()

file = Mp3FileProvider(FAKER).mp3_file(mp3_generator_cls=GttsMp3Generator)
```

You can tune arguments too:

```
from faker import Faker
from faker_file.providers.mp3_file import Mp3FileProvider
from faker_file.providers.mp3_file.generators.gtts_generator import (
    GttsMp3Generator,
)

FAKER = Faker()

file = Mp3FileProvider(FAKER).mp3_file(
    mp3_generator_cls=GttsMp3Generator,
    mp3_generator_kwargs={
```

(continues on next page)

(continued from previous page)

```

        "lang": "en",
        "tld": "co.uk",
    }
)

```

Refer to <https://gtts.readthedocs.io/en/latest/module.html#languages-gtts-lang> for list of accepted values for `lang` argument.

Refer to <https://gtts.readthedocs.io/en/latest/module.html#localized-accents> for list of accepted values for `tld` argument.

#### 14.2.1.22.2 Microsoft Edge Text-to-Speech

```

from faker import Faker
from faker_file.providers.mp3_file import Mp3FileProvider
from faker_file.providers.mp3_file.generators.edge_tts_generator import (
    EdgeTtsMp3Generator,
)

FAKER = Faker()

file = Mp3FileProvider(FAKER).mp3_file(mp3_generator_cls=EdgeTtsMp3Generator)

```

You can tune arguments too:

```

from faker import Faker
from faker_file.providers.mp3_file import Mp3FileProvider
from faker_file.providers.mp3_file.generators.edge_tts_generator import (
    EdgeTtsMp3Generator,
)

FAKER = Faker()

file = Mp3FileProvider(FAKER).mp3_file(
    mp3_generator_cls=EdgeTtsMp3Generator,
    mp3_generator_kwargs={
        "voice": "en-GB-LibbyNeural",
    }
)

```

Run `edge-tts -l` from terminal for list of available voices.

### 14.2.1.23 Create a MP3 file with custom MP3 generator

Default MP3 generator class is `GttsMp3Generator` which uses Google Text-to-Speech services to generate an MP3 file from given or randomly generated text. It does not require additional services to run and the only dependency here is the `gtts` package. You can however implement your own custom MP3 generator class and pass it to the `mp3_file` method in `mp3_generator_cls` argument instead of the default `GttsMp3Generator`. Read about quotas of Google Text-to-Speech services [here](#).

Usage with custom MP3 generator class.

```
# Imaginary `marytts` Python library
from marytts import MaryTTS

# Import BaseMp3Generator
from faker_file.providers.base.mp3_generator import (
    BaseMp3Generator,
)

# Define custom MP3 generator
class MaryTtsMp3Generator(BaseMp3Generator):

    locale: str = "cmu-rms-hsmm"
    voice: str = "en_US"

    def handle_kwargs(self, **kwargs) -> None:
        # Since it's impossible to unify all TTS systems it's allowed
        # to pass arbitrary arguments to the `BaseMp3Generator`
        # constructor. Each implementation class contains its own
        # additional tuning arguments. Check the source code of the
        # implemented MP3 generators as an example.
        if "locale" in kwargs:
            self.locale = kwargs["locale"]
        if "voice" in kwargs:
            self.voice = kwargs["voice"]

    def generate(self) -> bytes:
        # Your implementation here. Note, that `self.content`
        # in this context is the text to make MP3 from.
        # `self.generator` would be the `Faker` or `Generator`
        # instance from which you could extract information on
        # active locale.
        # What comes below is pseudo implementation.
        mary_tts = MaryTTS(locale=self.locale, voice=self.voice)
        return mary_tts.synth_mp3(self.content)

# Generate MP3 file from random text
file = FAKER.mp3_file(
    mp3_generator_cls=MaryTtsMp3Generator,
)
```

See exact implementation of `marytts_mp3_generator` in the examples.

#### 14.2.1.24 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",
)
```

#### 14.2.1.25 File from path given

- Create an exact copy of a file under a different name.
- Prefix of the destination file would be zzz.
- `path` is the absolute path to the file to copy.

```
from faker_file.providers.file_from_path import (
    FileFromPathProvider,
)

file = FileFromPathProvider(FAKER).file_from_path(
    path="/path/to/file.docx",
    prefix="zzz",
)
```

#### 14.2.1.26 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.

##### 14.2.1.26.1 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
```

#### 14.2.1.26.2 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
```

#### 14.2.1.27 Generate a lot of files using multiprocessing

##### 14.2.1.27.1 Generate 100 DOCX files

- Use template.
- Generate 100 DOCX files.

```
from multiprocessing import Pool
from faker import Faker
from faker_file.providers.helpers.inner import create_inner_docx_file
from faker_file.storages.filesystem import FileSystemStorage

FAKER = Faker()
STORAGE = FileSystemStorage()

# Document template
TEMPLATE = "Hey {{name}},\n{{text}},\nBest regards\n{{name}}"
```

```
with Pool(processes=8) as pool:
    for _ in range(100): # Number of times we want to run our function
        pool.apply_async(
            create_inner_docx_file,
            # Apply async doesn't support kwargs. We have to pass all
            # arguments.
            [STORAGE, "mp", FAKER, None, None, TEMPLATE],
        )
    pool.close()
    pool.join()
```

##### 14.2.1.27.2 Randomize the file format

```
from multiprocessing import Pool

from faker import Faker
from faker_file.providers.helpers.inner import (
    create_inner_docx_file,
    create_inner_epub_file,
    create_inner_pdf_file,
    create_inner_txt_file,
```

(continues on next page)

(continued from previous page)

```

        fuzzy_choice_create_inner_file,
    )
    from faker_file.storages.filesystem import FileSystemStorage

    FAKER = Faker()
    STORAGE = FileSystemStorage()

    # Document template
    TEMPLATE = """
    {{date}} {{city}}, {{country}}

    Hello {{name}},

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

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

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

    Address: {{address}}

    Best regards,

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

    kwargs = {"storage": STORAGE, "generator": FAKER, "content": TEMPLATE}

    with Pool(processes=8) as pool:
        for _ in range(100): # Number of times we want to run our function
            pool.apply_async(
                fuzzy_choice_create_inner_file,
                [
                    [
                        (create_inner_docx_file, kwargs),
                        (create_inner_epub_file, kwargs),
                        (create_inner_pdf_file, kwargs),
                        (create_inner_txt_file, kwargs),
                    ]
                ],
            )
        pool.close()
        pool.join()

```

### 14.2.1.28 Generating files from existing documents using NLP augmentation

See the following example:

```
from faker import Faker
from faker_file.providers.augment_file_from_dir import (
    AugmentFileFromDirProvider,
)

FAKER = Faker()

file = AugmentFileFromDirProvider(FAKER).augment_file_from_dir(
    source_dir_path="/path/to/source/",
)
```

Generated file will resemble text of the original document, but will not be the same. This is useful when you don't want to test on text generated by Faker, but rather something that makes more sense for your use case, still want to ensure uniqueness of the documents.

The following file types are supported:

- DOCX
- EML
- EPUB
- ODT
- PDF
- RTF
- TXT

By default, all supported files are eligible for random selection. You could however narrow that list by providing extensions argument:

```
file = AugmentFileFromDirProvider(FAKER).augment_file_from_dir(
    source_dir_path="/path/to/source/",
    extensions={"docx", "pdf"}, # Pick only DOCX or PDF
)
```

By default bert-base-multilingual-cased model is used, which is pretrained on the top 104 languages with the largest Wikipedia using a masked language modeling (MLM) objective. If you want to use a different model, specify the proper identifier in the model\_path argument. Some well working options for model\_path are:

- bert-base-multilingual-cased
- bert-base-multilingual-uncased
- bert-base-cased
- bert-base-uncased
- bert-base-german-cased
- GroNLP/bert-base-dutch-cased

```
from faker_file.providers.augment_file_from_dir.augmenters import (
    nlpaug_augmenter
```

(continues on next page)



(continued from previous page)

```

)

file = AugmentFileFromDirProvider(FAKER).augment_file_from_dir(
    text_augmenter_cls=(
        nlpaug_augmenter.ContextualWordEmbeddingsAugmenter
    ),
    text_augmenter_kwargs={
        "model_path": "bert-base-cased",
        "action": "substitute", # or "insert"
    }
)

```

Refer to [nlpaug docs](#) and check *Textual augmenters* examples.

#### 14.2.1.29 Using `raw=True` features in tests

If you pass `raw=True` argument to any provider or inner function, instead of creating a file, you will get bytes back (or to be totally correct, bytes-like object `BytesValue`, which is basically bytes enriched with meta-data). You could then use the bytes content of the file to build a test payload as shown in the example test below:

```

import os
from io import BytesIO

from django.test import TestCase
from django.urls import reverse
from faker import Faker
from faker_file.providers.docx_file import DocxFileProvider
from rest_framework.status import HTTP_201_CREATED
from upload.models import Upload

FAKER = Faker()
FAKER.add_provider(DocxFileProvider)

class UploadTestCase(TestCase):
    """Upload test case."""

    def test_create_docx_upload(self) -> None:
        """Test create an Upload."""
        url = reverse("api:upload-list")

        raw = FAKER.docx_file(raw=True)
        test_file = BytesIO(raw)
        test_file.name = os.path.basename(raw.data["filename"])

        payload = {
            "name": FAKER.word(),
            "description": FAKER.paragraph(),
            "file": test_file,
        }

        response = self.client.post(url, payload, format="json")

```

(continues on next page)

(continued from previous page)

```
# Test if request is handled properly (HTTP 201)
self.assertEqual(response.status_code, HTTP_201_CREATED)

test_upload = Upload.objects.get(id=response.data["id"])

# Test if the name is properly recorded
self.assertEqual(str(test_upload.name), payload["name"])

# Test if file name recorded properly
self.assertEqual(str(test_upload.file.name), test_file.name)
```

#### 14.2.1.30 Create a HTML file predefined template

If you want to generate a file in a format that is not (yet) supported, you can try to use `GenericFileProvider`. In the following example, an HTML file is generated from a template.

```
from faker import Faker
from faker_file.providers.generic_file import GenericFileProvider

file = GenericFileProvider(Faker()).generic_file(
    content="<html><body><p>{{text}}</p></body></html>",
    extension="html",
)
```

#### 14.2.1.31 Working with storages

##### 14.2.1.31.1 AWS S3 storage

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

FAKER = Faker()
AWS_S3_STORAGE = AWSS3Storage(
    bucket_name="your-bucket-name",
    root_path="",
    rel_path="",
)
FAKER.add_provider(TxtFileProvider)

txt_file = FAKER.txt_file(storage=AWS_S3_STORAGE)
```

### 14.2.1.31.2 SFTP storage

```
from faker import Faker
from faker_file.providers.txt_file import TxtFileProvider
from faker_file.storages.sftp import SFTPStorage

FAKER = Faker()
SFTP_STORAGE = SFTPStorage(
    host="your-sftp-host.domain",
    port: 22,
    username: "your-sftp-username",
    password: "your-sftp-password",
    root_path: "/dir-name",
)
FAKER.add_provider(TxtFileProvider)

txt_file = FAKER.txt_file(storage=SFTP_STORAGE)
```

## 14.2.2 When using with Django (and factory\_boy)

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

### 14.2.2.1 Basic example

#### 14.2.2.1.1 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)

    # File
    file = models.FileField(null=True)

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

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

#### 14.2.2.1.2 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.bin_file import BinFileProvider
from faker_file.providers.csv_file import CsvFileProvider
from faker_file.providers.docx_file import DocxFileProvider
from faker_file.providers.eml_file import EmlFileProvider
from faker_file.providers.epub_file import EpubFileProvider
from faker_file.providers.ico_file import IcoFileProvider
from faker_file.providers.jpeg_file import JpegFileProvider
from faker_file.providers.mp3_file import Mp3FileProvider
from faker_file.providers.ods_file import OdsFileProvider
from faker_file.providers.odt_file import OdtFileProvider
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.rtf_file import RtfFileProvider
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

# Import file storage, because we need to customize things in order for it
# to work with Django.
from faker_file.storages.filesystem import FileSystemStorage

from upload.models import Upload

# Add all providers we want to use
Faker.add_provider(BinFileProvider)
Faker.add_provider(CsvFileProvider)
Faker.add_provider(DocxFileProvider)
Faker.add_provider(EmlFileProvider)
Faker.add_provider(EpubFileProvider)
Faker.add_provider(IcoFileProvider)
Faker.add_provider(JpegFileProvider)
Faker.add_provider(Mp3FileProvider)
Faker.add_provider(OdsFileProvider)
Faker.add_provider(OdtFileProvider)
Faker.add_provider(PdfFileProvider)
Faker.add_provider(PngFileProvider)
Faker.add_provider(PptxFileProvider)
Faker.add_provider(RtfFileProvider)
Faker.add_provider(SvgFileProvider)
Faker.add_provider(TxtFileProvider)
Faker.add_provider(WebpFileProvider)
```

(continues on next page)

(continued from previous page)

```

Faker.add_provider(XlsxFileProvider)
Faker.add_provider(ZipFileProvider)

# Define a file storage. When working with Django and FileSystemStorage
# you need to set the value of `root_path` argument to
# `settings.MEDIA_ROOT`.
STORAGE = FileSystemStorage(
    root_path=settings.MEDIA_ROOT,
    rel_path="tmp"
)

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

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

    class Meta:
        model = Upload

    class Params:
        bin_file = Trait(file=Faker("bin_file", storage=STORAGE))
        csv_file = Trait(file=Faker("csv_file", storage=STORAGE))
        docx_file = Trait(file=Faker("docx_file", storage=STORAGE))
        eml_file = Trait(file=Faker("eml_file", storage=STORAGE))
        epub_file = Trait(file=Faker("epub_file", storage=STORAGE))
        ico_file = Trait(file=Faker("ico_file", storage=STORAGE))
        jpeg_file = Trait(file=Faker("jpeg_file", storage=STORAGE))
        mp3_file = Trait(file=Faker("mp3_file", storage=STORAGE))
        ods_file = Trait(file=Faker("ods_file", storage=STORAGE))
        odt_file = Trait(file=Faker("odt_file", storage=STORAGE))
        pdf_file = Trait(file=Faker("pdf_file", storage=STORAGE))
        png_file = Trait(file=Faker("png_file", storage=STORAGE))
        pptx_file = Trait(file=Faker("pptx_file", storage=STORAGE))
        rtf_file = Trait(file=Faker("rtf_file", storage=STORAGE))
        svg_file = Trait(file=Faker("svg_file", storage=STORAGE))
        txt_file = Trait(file=Faker("txt_file", storage=STORAGE))
        webp_file = Trait(file=Faker("webp_file", storage=STORAGE))
        xlsx_file = Trait(file=Faker("xlsx_file", storage=STORAGE))
        zip_file = Trait(file=Faker("zip_file", storage=STORAGE))

```

And then somewhere in your code:

```

UploadFactory(bin_file=True) # Upload with BIN file
UploadFactory(docx_file=True) # Upload with DOCX file
UploadFactory(jpeg_file=True) # Upload with JPEG file
UploadFactory(zip_file=True) # Upload with ZIP file

```

### 14.2.2.2 Randomize provider choice

```
from factory import LazyAttribute
from faker import Faker
from random import choice

FAKER = Faker()

PROVIDER_CHOICES = [
    lambda: BinFileProvider(FAKER).bin_file(storage=STORAGE),
    lambda: CsvFileProvider(FAKER).csv_file(storage=STORAGE),
    lambda: DocxFileProvider(FAKER).docx_file(storage=STORAGE),
    lambda: EmlFileProvider(FAKER).eml_file(storage=STORAGE),
    lambda: EpubFileProvider(FAKER).epub_file(storage=STORAGE),
    lambda: IcoFileProvider(FAKER).ico_file(storage=STORAGE),
    lambda: JpegFileProvider(FAKER).jpeg_file(storage=STORAGE),
    lambda: Mp3FileProvider(FAKER).mp3_file(storage=STORAGE),
    lambda: OdsFileProvider(FAKER).ods_file(storage=STORAGE),
    lambda: OdtFileProvider(FAKER).odt_file(storage=STORAGE),
    lambda: PdfFileProvider(FAKER).pdf_file(storage=STORAGE),
    lambda: PngFileProvider(FAKER).png_file(storage=STORAGE),
    lambda: PptxFileProvider(FAKER).pptx_file(storage=STORAGE),
    lambda: RtfFileProvider(FAKER).rtf_file(storage=STORAGE),
    lambda: SvgFileProvider(FAKER).svg_file(storage=STORAGE),
    lambda: TxtFileProvider(FAKER).txt_file(storage=STORAGE),
    lambda: XlsxFileProvider(FAKER).xlsx_file(storage=STORAGE),
    lambda: ZipFileProvider(FAKER).zip_file(storage=STORAGE),
]

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

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

    # ...
    class Params:
        # ...
        random_file = Trait(file=LazyAttribute(pick_random_provider))
        # ...
```

And then somewhere in your code:

```
UploadFactory(random_file=True) # Upload with random file
```

### 14.2.2.3 Use a different locale

```

from factory import Faker
from factory.django import DjangoModelFactory
from faker_file.providers.ods_file import OdsFileProvider

Faker._DEFAULT_LOCALE = "hy_AM" # Set locale to Armenian

Faker.add_provider(OdsFileProvider)

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

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

    class Meta:
        """Meta class."""

        model = Upload

```

### 14.2.2.4 Other Django usage examples

#### Faker example with AWS S3 storage

```

from django.conf import settings
from faker import Faker
from faker_file.providers.pdf_file import PdfFileProvider
from faker_file.storages.aws_s3 import AWSS3Storage

FAKER = Faker()
STORAGE = AWSS3Storage(
    bucket_name=settings.AWS_STORAGE_BUCKET_NAME,
    root_path="",
    rel_path="",
)
FAKER.add_provider(PdfFileProvider)

file = FAKER.pdf_file(storage=STORAGE)

```

#### factory-boy example with AWS S3 storage

```

import factory

from django.conf import settings
from factory import Faker
from factory.django import DjangoModelFactory
from faker_file.storages.aws_s3 import AWSS3Storage

from upload.models import Upload

```

(continues on next page)

(continued from previous page)

```

STORAGE = AWSS3Storage(
    bucket_name=settings.AWS_STORAGE_BUCKET_NAME,
    root_path="",
    rel_path="",
)

```

```
Faker.add_provider(PdfFileProvider)
```

```

class UploadFactory(DjangoModelFactory):
    name = Faker('word')
    description = Faker('text')
    file = Faker("pdf_file", storage=STORAGE)

```

```

class Meta:
    model = Upload

```

```
upload = UploadFactory()
```

### Flexible storage selection

```

from django.conf import settings
from django.core.files.storage import default_storage
from faker_file.storages.aws_s3 import AWSS3Storage
from faker_file.storages.filesystem import FileSystemStorage
from storages.backends.s3boto3 import S3Boto3Storage

# Faker doesn't know anything about Django. That's why, if we want to
# support remote storages, we need to manually check which file storage
# backend is used. If `Boto3` storage backend (of the `django-storages`
# package) is used we use the correspondent `AWSS3Storage` class of the
# `faker-file`.
# Otherwise, fall back to native file system storage (`FileSystemStorage`)
# of the `faker-file`.
if isinstance(default_storage, S3Boto3Storage):
    STORAGE = AWSS3Storage(
        bucket_name=settings.AWS_STORAGE_BUCKET_NAME,
        credentials={
            "key_id": settings.AWS_ACCESS_KEY_ID,
            "key_secret": settings.AWS_SECRET_ACCESS_KEY,
        },
        rel_path="tmp",
    )
else:
    STORAGE = FileSystemStorage(
        root_path=settings.MEDIA_ROOT,
        rel_path="tmp",
    )

```



## 14.3 Creating PDF

PDF is certainly one of the most complicated formats out there. And certainly one of the formats most of the developers will be having trouble with, as there are many versions and dialects. That makes it almost challenging to create one way of making PDF files. That's why, creation of PDF files have been delegated to flexible abstraction layer - PDF generators. If you don't like how PDF files are generated, you can create your own layer, using your favourite library.

Currently, there are two PDF generators:

- PdfkitPdfGenerator (default), built on top of the `pdfkit` and `wkhtmltopdf`.
- ReportlabPdfGenerator, build on top of the famous `reportlab`.

### 14.3.1 Building PDF using pdfkit

While `pdfkit` generator is a heavier and has `wkhtmltopdf` as a system dependency, it's produces better quality PDF and has no issues with fonts or unicode characters.

See the following full functional snippet for generating PDF using `pdfkit`.

```
# Imports
from faker import Faker
from faker_file.providers.pdf_file import PdfFileProvider
from faker_file.providers.pdf_file.generators.pdfkit_generator import (
    PdfkitPdfGenerator,
)

FAKER = Faker() # Initialize Faker
FAKER.add_provider(PdfFileProvider) # Register PdfFileProvider

# Generate PDF file using `pdfkit`
pdf_file = FAKER.pdf_file(pdf_generator_cls=PdfkitPdfGenerator)
```

The generated PDF will have 10,000 characters of text, which is about 2 pages.

If you want PDF with more pages, you could either:

- Increase the value of `max_nb_chars` accordingly.
- Set value of `wrap_chars_after` to 80 characters to force longer pages.
- Insert manual page breaks and other content.

See the example below for `max_nb_chars` tweak:

```
# Generate PDF file of 20,000 characters
pdf_file = FAKER.pdf_file(
    pdf_generator_cls=PdfkitPdfGenerator, max_nb_chars=20_000
)
```

See the example below for `wrap_chars_after` tweak:

```
# Generate PDF file, wrapping each line after 80 characters
pdf_file = FAKER.pdf_file(
    pdf_generator_cls=PdfkitPdfGenerator, wrap_chars_after=80
)
```

As mentioned above, it's possible to diversify the generated context with images, paragraphs, tables, manual text break and pretty much everything that is supported by PDF format specification, although currently only images, paragraphs, tables and manual text breaks are supported out of the box. In order to customise the blocks PDF file is built from, the `DynamicTemplate` class is used. See the example below for usage examples:

```
# Additional imports
from faker_file.base import DynamicTemplate
from faker_file.contrib.pdf_file.pdfkit_snippets import (
    add_page_break,
    add_paragraph,
    add_picture,
    add_table,
)

# Create a PDF file with paragraph, picture, table and manual page breaks
# in between the mentioned elements. The ``DynamicTemplate`` simply
# accepts a list of callables (such as ``add_paragraph``,
# ``add_page_break``) and dictionary to be later on fed to the callables
# as keyword arguments for customising the default values.
pdf_file = FAKER.pdf_file(
    pdf_generator_cls=PdfkitPdfGenerator,
    content=DynamicTemplate(
        [
            (add_paragraph, {}), # Add paragraph
            (add_page_break, {}), # Add page break
            (add_picture, {}), # Add picture
            (add_page_break, {}), # Add page break
            (add_table, {}), # Add table
            (add_page_break, {}), # Add page break
        ]
    )
)

# You could make the list as long as you like or simply multiply for
# easier repetition as follows:
pdf_file = FAKER.pdf_file(
    pdf_generator_cls=PdfkitPdfGenerator,
    content=DynamicTemplate(
        [
            (add_paragraph, {}), # Add paragraph
            (add_page_break, {}), # Add page break
            (add_picture, {}), # Add picture
            (add_page_break, {}), # Add page break
            (add_table, {}), # Add table
            (add_page_break, {}), # Add page break
        ] * 100 # Will repeat your config 100 times
    )
)
```

### 14.3.2 Building PDFs using reportlab

While `reportlab` generator is much lighter than the `pdfkit` and does not have system dependencies, but might produce PDF files with questionable encoding when generating unicode text.

See the following full functional snippet for generating PDF using `reportlab`.

```
# Imports
from faker import Faker
from faker_file.providers.pdf_file import PdfFileProvider
from faker_file.providers.pdf_file.generators.reportlab_generator import (
    ReportlabPdfGenerator,
)

# Generate PDF file using `reportlab`
pdf_file = FAKER.pdf_file(pdf_generator_cls=ReportlabPdfGenerator)
```

All examples shown for `pdfkit` apply for `reportlab` generator, however when building PDF files from blocks (paragraphs, images, tables and page breaks), the imports shall be adjusted:

As mentioned above, it's possible to diversify the generated context with images, paragraphs, tables, manual text break and pretty much everything that is supported by PDF format specification, although currently only images, paragraphs, tables and manual text breaks are supported. In order to customise the blocks PDF file is built from, the `DynamicTemplate` class is used. See the example below for usage examples:

```
# Additional imports
from faker_file.base import DynamicTemplate
from faker_file.contrib.pdf_file.reportlab_snippets import (
    add_page_break,
    add_paragraph,
    add_picture,
    add_table,
)

# Create a PDF file with paragraph, picture, table and manual page breaks
# in between the mentioned elements. The ``DynamicTemplate`` simply
# accepts a list of callables (such as ``add_paragraph``,
# ``add_page_break``) and dictionary to be later on fed to the callables
# as keyword arguments for customising the default values.
pdf_file = FAKER.pdf_file(
    pdf_generator_cls=ReportlabPdfGenerator,
    content=DynamicTemplate(
        [
            (add_paragraph, {}), # Add paragraph
            (add_page_break, {}), # Add page break
            (add_picture, {}), # Add picture
            (add_page_break, {}), # Add page break
            (add_table, {}), # Add table
            (add_page_break, {}), # Add page break
        ]
    )
)

# You could make the list as long as you like or simply multiply for
# easier repetition as follows:
```

(continues on next page)

(continued from previous page)

```
pdf_file = FAKER.pdf_file(  
    pdf_generator_cls=ReportlabPdfGenerator,  
    content=DynamicTemplate(  
        [  
            (add_paragraph, {}), # Add paragraph  
            (add_page_break, {}), # Add page break  
            (add_picture, {}), # Add picture  
            (add_page_break, {}), # Add page break  
            (add_table, {}), # Add table  
            (add_page_break, {}), # Add page break  
        ] * 100  
    )  
)
```

## 14.4 Creating DOCX

See the following full functional snippet for generating DOCX.

```
# Imports  
from faker import Faker  
from faker_file.providers.docx_file import DocxFileProvider  
  
FAKER = Faker() # Initialize Faker  
FAKER.add_provider(DocxFileProvider) # Register DocxFileProvider  
  
# Generate DOCX file  
docx_file = FAKER.docx_file()
```

The generated DOCX will have 10,000 characters of text, which is about 5 pages.

If you want DOCX with more pages, you could either:

- Increase the value of `max_nb_chars` accordingly.
- Set value of `wrap_chars_after` to 80 characters to force longer pages.
- Insert manual page breaks and other content.

See the example below for `max_nb_chars` tweak:

```
# Generate DOCX file of 20,000 characters  
docx_file = FAKER.docx_file(max_nb_chars=20_000)
```

See the example below for `wrap_chars_after` tweak:

```
# Generate DOCX file, wrapping each line after 80 characters  
docx_file = FAKER.docx_file(wrap_chars_after=80)
```

As mentioned above, it's possible to diversify the generated context with images, paragraphs, tables, manual text break and pretty much everything that is supported by DOCX format specification, although currently only images, paragraphs, tables and manual text breaks are supported out of the box. In order to customise the blocks DOCX file is built from, the `DynamicTemplate` class is used. See the example below for usage examples:

```

# Additional imports
from faker_file.base import DynamicTemplate
from faker_file.contrib.docx_file import (
    add_page_break,
    add_paragraph,
    add_picture,
    add_table,
)

# Create a DOCX file with paragraph, picture, table and manual page breaks
# in between the mentioned elements. The ``DynamicTemplate`` simply
# accepts a list of callables (such as ``add_paragraph``,
# ``add_page_break``) and dictionary to be later on fed to the callables
# as keyword arguments for customising the default values.
docx_file = FAKER.docx_file(
    content=DynamicTemplate(
        [
            (add_paragraph, {}), # Add paragraph
            (add_page_break, {}), # Add page break
            (add_picture, {}), # Add picture
            (add_page_break, {}), # Add page break
            (add_table, {}), # Add table
            (add_page_break, {}), # Add page break
        ]
    )
)

# You could make the list as long as you like or simply multiply for
# easier repetition as follows:
docx_file = FAKER.docx_file(
    content=DynamicTemplate(
        [
            (add_paragraph, {}), # Add paragraph
            (add_page_break, {}), # Add page break
            (add_picture, {}), # Add picture
            (add_page_break, {}), # Add page break
            (add_table, {}), # Add table
            (add_page_break, {}), # Add page break
        ] * 100 # Will repeat your config 100 times
    )
)

```

## 14.5 Creating ODT

See the following full functional snippet for generating ODT.

```

# Imports
from faker import Faker
from faker_file.providers.odt_file import OdtFileProvider

FAKER = Faker() # Initialize Faker

```

(continues on next page)

(continued from previous page)

```
FAKER.add_provider(OdtFileProvider)  # Register OdtFileProvider

# Generate ODT file
odt_file = FAKER.odt_file()
```

The generated ODT will have 10,000 characters of text, which is about 5 pages.

If you want ODT with more pages, you could either:

- Increase the value of `max_nb_chars` accordingly.
- Set value of `wrap_chars_after` to 80 characters to force longer pages.
- Insert manual page breaks and other content.

See the example below for `max_nb_chars` tweak:

```
# Generate ODT file of 20,000 characters
odt_file = FAKER.odt_file(max_nb_chars=20_000)
```

See the example below for `wrap_chars_after` tweak:

```
# Generate DOCX file, wrapping each line after 80 characters
docx_file = FAKER.odt_file(wrap_chars_after=80)
```

As mentioned above, it's possible to diversify the generated context with images, paragraphs, tables, manual text break and pretty much everything that is supported by ODT format specification, although currently only images, paragraphs, tables and manual text breaks are supported out of the box. In order to customise the blocks DOCX file is built from, the `DynamicTemplate` class is used. See the example below for usage examples:

```
# Additional imports
from faker_file.base import DynamicTemplate
from faker_file.contrib.odt_file import (
    add_page_break,
    add_paragraph,
    add_picture,
    add_table,
)

# Create a ODT file with paragraph, picture, table and manual page breaks
# in between the mentioned elements. The ``DynamicTemplate`` simply
# accepts a list of callables (such as ``add_paragraph``,
# ``add_page_break``) and dictionary to be later on fed to the callables
# as keyword arguments for customising the default values.
odt_file = FAKER.odt_file(
    content=DynamicTemplate(
        [
            (add_paragraph, {}), # Add paragraph
            (add_page_break, {}), # Add page break
            (add_picture, {}), # Add picture
            (add_page_break, {}), # Add page break
            (add_table, {}), # Add table
            (add_page_break, {}), # Add page break
        ]
    )
)
```

(continues on next page)

(continued from previous page)

```

)

# You could make the list as long as you like or simply multiply for
# easier repetition as follows:
odt_file = FAKER.odt_file(
    content=DynamicTemplate(
        [
            (add_paragraph, {}), # Add paragraph
            (add_page_break, {}), # Add page break
            (add_picture, {}), # Add picture
            (add_page_break, {}), # Add page break
            (add_table, {}), # Add table
            (add_page_break, {}), # Add page break
        ] * 100 # Will repeat your config 100 times
    )
)

```

## 14.6 CLI

It's possible to generate files from CLI.

---

**Note:** For using CLI you should install all common dependencies.

---

Install using `pipx` (recommended):

```
pipx install faker-file[common]
```

Install using `pip`.

```
pip install faker-file[common] --user
```

### 14.6.1 List available provider options

```
faker-file --help
```

Output:

```

usage: faker-file [-h] [-o OUTPUT_DIR]
                  {bin_file, csv_file, docx_file, eml_file, epub_file, generic_file, ico_file,
  ↪ jpeg_file, mp3_file, odp_file, ods_file, odt_file, pdf_file, png_file, pptx_file, rtf_file, svg_
  ↪ file, tar_file, txt_file, webp_file, xlsx_file, xml_file, zip_file}
                  ...

CLI for the faker-file package.

positional arguments:
  {bin_file, csv_file, docx_file, eml_file, epub_file, generic_file, ico_file, jpeg_file, mp3_
  ↪ file, odp_file, ods_file, odt_file, pdf_file, png_file, pptx_file, rtf_file, svg_file, tar_file,

```

(continues on next page)

(continued from previous page)

```
↪txt_file,webp_file,xlsx_file,xml_file,zip_file}
```

	Available file providers.
bin_file	Generate a bin file.
csv_file	Generate a csv file.
docx_file	Generate a docx file.
eml_file	Generate a eml file.
epub_file	Generate a epub file.
generic_file	Generate a generic file.
ico_file	Generate a ico file.
jpeg_file	Generate a jpeg file.
mp3_file	Generate a mp3 file.
odp_file	Generate a odp file.
ods_file	Generate a ods file.
odt_file	Generate a odt file.
pdf_file	Generate a pdf file.
png_file	Generate a png file.
pptx_file	Generate a pptx file.
rtf_file	Generate a rtf file.
svg_file	Generate a svg file.
tar_file	Generate a tar file.
txt_file	Generate a txt file.
webp_file	Generate a webp file.
xlsx_file	Generate a xlsx file.
xml_file	Generate a xml file.
zip_file	Generate a zip file.

options:

-h, --help	show this help message and exit
------------	---------------------------------

## 14.6.2 List options for a certain provider

```
faker-file docx_file --help
```

Output:

```
usage: faker-file docx_file [-h] [--prefix PREFIX] [--max_nb_chars MAX_NB_CHARS] [--wrap_
↪chars_after WRAP_CHARS_AFTER] [--content CONTENT] [--nb_files NB_FILES]
```

options:

-h, --help	show this help message and exit
--prefix PREFIX	prefix (default: None)
--max_nb_chars MAX_NB_CHARS	max_nb_chars (default: 10000)
--wrap_chars_after WRAP_CHARS_AFTER	wrap_chars_after (default: None)
--content CONTENT	content (default: None)
--nb_files NB_FILES	number of files to generate (default: 1)



### 14.6.3 Generate a file using certain provider

```
faker-file docx_file
```

Output:

```
Generated docx_file file: tmp/tmpva0mp3lp.docx
```

### 14.6.4 Shell auto-completion

First, generate shell auto-completion file.

```
faker-file generate-completion
```

Then, source the generated file:

```
source ~/faker_file_completion.sh
```

Now you can use auto-completion. Simply type `faker-file [tab-tab]` to see the list of available options:

```
$ faker-file
bin_file      generic_file  ods_file      rtf_file      xlsx_file
csv_file      ico_file      odt_file      svg_file      xml_file
docx_file     jpeg_file     pdf_file      tar_file      zip_file
eml_file      mp3_file      png_file      txt_file
epub_file     odp_file      pptx_file     webp_file
```

It works with sub options too:

```
$ faker-file docx_file --
--content      --max_nb_chars  --prefix      --wrap_chars_after  --nb_files
```

To update the completion script, simply run the `generate-completion` command again and source the `~/faker_file_completion.sh` as already shown above.

## 14.7 Security Policy

### 14.7.1 Reporting a Vulnerability

**Do not report security issues on GitHub!**

Please report security issues by emailing Artur Barseghyan <[artur.barseghyan@gmail.com](mailto:artur.barseghyan@gmail.com)>.

## 14.7.2 Supported Versions

### Make sure to use the latest version.

The two most recent `faker-file` release series receive security support.

For example, during the development cycle leading to the release of `faker-file` 0.16.x, support will be provided for `faker-file` 0.15.x.

Upon the release of `faker-file` 0.17.x, security support for `faker-file` 0.15.x will end.

Version	Supported	
0.16.x	Yes	
0.15.x	Yes	
< 0.15	No	

## 14.8 Contributor Covenant Code of Conduct

### 14.8.1 Our Pledge

We as members, contributors, and leaders pledge to make participation in our community a harassment-free experience for everyone, regardless of age, body size, visible or invisible disability, ethnicity, sex characteristics, gender identity and expression, level of experience, education, socio-economic status, nationality, personal appearance, race, religion, or sexual identity and orientation.

We pledge to act and interact in ways that contribute to an open, welcoming, diverse, inclusive, and healthy community.

### 14.8.2 Our Standards

Examples of behavior that contributes to a positive environment for our community include:

- Demonstrating empathy and kindness toward other people
- Being respectful of differing opinions, viewpoints, and experiences
- Giving and gracefully accepting constructive feedback
- Accepting responsibility and apologizing to those affected by our mistakes, and learning from the experience
- Focusing on what is best not just for us as individuals, but for the overall community

Examples of unacceptable behavior include:

- The use of sexualized language or imagery, and sexual attention or advances of any kind
- Trolling, insulting or derogatory comments, and personal or political attacks
- Public or private harassment
- Publishing others' private information, such as a physical or email address, without their explicit permission
- Other conduct which could reasonably be considered inappropriate in a professional setting

### 14.8.3 Enforcement Responsibilities

Community leaders are responsible for clarifying and enforcing our standards of acceptable behavior and will take appropriate and fair corrective action in response to any behavior that they deem inappropriate, threatening, offensive, or harmful.

Community leaders have the right and responsibility to remove, edit, or reject comments, commits, code, wiki edits, issues, and other contributions that are not aligned to this Code of Conduct, and will communicate reasons for moderation decisions when appropriate.

### 14.8.4 Scope

This Code of Conduct applies within all community spaces, and also applies when an individual is officially representing the community in public spaces. Examples of representing our community include using an official e-mail address, posting via an official social media account, or acting as an appointed representative at an online or offline event.

### 14.8.5 Enforcement

Instances of abusive, harassing, or otherwise unacceptable behavior may be reported to the community leaders responsible for enforcement at [artur.barseghyan@gmail.com](mailto:artur.barseghyan@gmail.com). All complaints will be reviewed and investigated promptly and fairly.

All community leaders are obligated to respect the privacy and security of the reporter of any incident.

### 14.8.6 Enforcement Guidelines

Community leaders will follow these Community Impact Guidelines in determining the consequences for any action they deem in violation of this Code of Conduct:

#### 14.8.6.1 1. Correction

**Community Impact:** Use of inappropriate language or other behavior deemed unprofessional or unwelcome in the community.

**Consequence:** A private, written warning from community leaders, providing clarity around the nature of the violation and an explanation of why the behavior was inappropriate. A public apology may be requested.

#### 14.8.6.2 2. Warning

**Community Impact:** A violation through a single incident or series of actions.

**Consequence:** A warning with consequences for continued behavior. No interaction with the people involved, including unsolicited interaction with those enforcing the Code of Conduct, for a specified period of time. This includes avoiding interactions in community spaces as well as external channels like social media. Violating these terms may lead to a temporary or permanent ban.

### 14.8.6.3 3. Temporary Ban

**Community Impact:** A serious violation of community standards, including sustained inappropriate behavior.

**Consequence:** A temporary ban from any sort of interaction or public communication with the community for a specified period of time. No public or private interaction with the people involved, including unsolicited interaction with those enforcing the Code of Conduct, is allowed during this period. Violating these terms may lead to a permanent ban.

### 14.8.6.4 4. Permanent Ban

**Community Impact:** Demonstrating a pattern of violation of community standards, including sustained inappropriate behavior, harassment of an individual, or aggression toward or disparagement of classes of individuals.

**Consequence:** A permanent ban from any sort of public interaction within the community.

## 14.8.7 Attribution

This Code of Conduct is adapted from the Contributor Covenant, version 2.0, available at [https://www.contributor-covenant.org/version/2/0/code\\_of\\_conduct.html](https://www.contributor-covenant.org/version/2/0/code_of_conduct.html).

Community Impact Guidelines were inspired by Mozilla's code of conduct enforcement ladder.

For answers to common questions about this code of conduct, see the FAQ at <https://www.contributor-covenant.org/faq>. Translations are available at <https://www.contributor-covenant.org/translations>.

## 14.9 Contributor guidelines

### 14.9.1 Developer prerequisites

#### 14.9.1.1 pre-commit

Refer to `pre-commit` for installation instructions.

TL;DR:

```
pip install pipx --user # Install pipx
pipx install pre-commit # Install pre-commit
pre-commit install # Install pre-commit hooks
```

Installing `pre-commit` will ensure you adhere to the project code quality standards.

### 14.9.2 Code standards

`black`, `isort`, `ruff` and `doc8` will be automatically triggered by `pre-commit`. Still, if you want to run checks manually:

```
./scripts/black.sh
./scripts/doc8.sh
./scripts/isort.sh
./scripts/ruff.sh
```

### 14.9.3 Requirements

Requirements are compiled using [pip-tools](#).

```
./scripts/compile_requirements.sh
```

### 14.9.4 Virtual environment

You are advised to work in virtual environment.

TL;DR:

```
python -m venv env
pip install -e .
pip install -r examples/requirements/django_3_2_and_flask.txt
```

### 14.9.5 Documentation

Check [documentation](#).

### 14.9.6 Testing

Check [testing](#).

If you introduce changes or fixes, make sure to test them locally using all supported environments. For that use `tox`.

```
tox
```

In any case, GitHub Actions will catch potential errors, but using `tox` speeds things up.

### 14.9.7 Pull requests

You can contribute to the project by making a [pull request](#).

For example:

- To fix documentation typos.
- To improve documentation (for instance, to add new recipe or fix an existing recipe that doesn't seem to work).
- To introduce a new feature (for instance, add support for a non-supported file type).

**Good to know:**

- Test suite makes extensive use of parametrization. Make sure you have added your changes in the right place.

**General list to go through:**

- Does your change require documentation update?
- Does your change require update to tests?
- Did you test both Latin and Unicode characters?
- Does your change rely on third-party cloud based service? If so, please make sure it's added to tests that should be retried a couple of times. Example: `@pytest.mark.flaky(reruns=5)`.

**When fixing bugs (in addition to the general list):**

- Make sure to add regression tests.

**When adding a new feature (in addition to the general list):**

- Check the licenses of added dependencies carefully and make sure to list them in [prerequisites](#).
- Make sure to update the documentation (check whether the [installation](#), [features](#), [recipes](#) and [quick start](#) require changes).

## 14.9.8 Questions

Questions can be asked on GitHub [discussions](#).

## 14.9.9 Issues

For reporting a bug or filing a feature request use GitHub [issues](#).

**Do not report security issues on GitHub.** Check the [support](#) section.

## 14.10 Release history and notes

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

<code>major.minor[.revision]</code>
-------------------------------------

- 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.

### 14.10.1 0.16.4

2023-07-01

- Documentation improvements. Added a dedicated section for creating ODT files.
- Adding `add_paragraph` and `add_page_break` to ODT contrib module.

### 14.10.2 0.16.3

2023-06-30

- Documentation improvements. Added a dedicated section for creating PDF files. Added a dedicated section for creating DOCX files.
- Adding `add_paragraph` and `add_page_break` to DOCX contrib module.

### 14.10.3 0.16.2

2023-06-28

- Moving some of the snippets from tests to a `contrib` module to improve usability. The snippets are generic enough to be used in tests and if you don't like the way they work, you could always make a new one. New snippets to insert page breaks and paragraphs into PDF (using both `pdftkit` and `reportlab` generators) have been added.

### 14.10.4 0.16.1

2023-06-23

- Better error handling in CLI.

### 14.10.5 0.16

2023-06-21

---

**Note:** This release is dedicated to my beloved son - Tigran, who turned 11!

---

---

**Note:** This release introduces minor backwards incompatible changes.

---

- Minor improvements in PDF generation. If you have been using `DynamicTemplate` to generate complex PDFs, you are likely affected by the change. Make sure to at least add an additional argument named `generator` to the functions passed to the `DynamicTemplate` class. See the example below:

Old:

```
def add_pb(provider, story, data, counter, **kwargs):
```

New:

```
def add_pb(provider, generator, story, data, counter, **kwargs):
```

- Add code examples of how to generate a PDF with 100 pages with both `PdftkitPdfGenerator` and `ReportlabPdfGenerator` PDF generator classes.
- Add `version` CLI command.
- Add `generate-completion` and `version` commands to the CLI auto-completion.

### 14.10.6 0.15.5

2023-06-18

- Minor fixes and documentation improvements.

### **14.10.7 0.15.4**

2023-06-15

- Improved SFTPStorage tests.
- Stop testing against Python 3.7.
- Stop testing against Django 4.0.

### **14.10.8 0.15.3**

2023-06-14

- Add SFTPStorage and correspondent tests.

### **14.10.9 0.15.2**

2023-06-08

- Add optional subject argument to the EmlFileProvider. Update tests accordingly.
- Add data integrity tests.

### **14.10.10 0.15.1**

2023-06-06

- Added FileFromPathProvider provider, which simply picks a file from path given. Add correspondent `create_inner_file_from_path` inner function.

### **14.10.11 0.15**

2023-06-05

- Added `format_func` argument to most of the providers. This allows to control which formatter function will be used as a default formatter. Previously it has been `faker.provider.Python.pystr_format`, which has been changed to `faker.provider.Python.parse`, since the latter is more convenient (as it does not transform characters like `?`, `!`, `#` into something else using `bothify` method). To revert this behaviour, make sure to pass a callable function `faker_file.base.pystr_format_func` in `format_func` argument to each correspondent provider or inner function.
- Added `create_inner_random_file_from_dir` inner function.
- Tested against Django 4.2.
- Stop testing against Django 2.2.



### 14.10.12 0.14.5

2023-05-11

- Minor fixes in `xml_file` provider.

### 14.10.13 0.14.4

2023-05-11

- Changed type of `data_columns` for `xml_file` provider from `Sequence[Tuple[str, str]]` to `Dict[str, str]`.
- In the `pdf_file` provider, changed default value of `pdf_generator_cls` from concrete `PdfkitPdfGenerator` value to its' string representation `faker_file.providers.pdf_file.generators.pdfkit_generator.PdfkitPdfGenerator`.
- In the `mp3_file` provider, changed default value of `mp3_generator_cls` from concrete `GttsMp3Generator` value to its' string representation `faker_file.providers.mp3_file.generators.gtts_generator.GttsMp3Generator`.

### 14.10.14 0.14.3

2023-05-10

- Minor fixes in the `GenericFileProvider`.

### 14.10.15 0.14.2

2023-05-09

- Add `create_inner_generic_file` inner function.
- Add `generic_file` support to CLI.

### 14.10.16 0.14.1

2023-05-08

- Add support for `list_create_inner_file`-like functions to the EML file provider. If you are using CLI and CLI-completion, make sure to re-generate the completion file.
- Add `GenericFileProvider` provider to support generic file types.

### 14.10.17 0.14

2023-05-07

---

**Note:** This release introduces minor backwards incompatible changes.

---

- A new argument `basename` has been added to all providers, inner functions and storage classes. If you have customized things or created your own providers, make sure to make appropriate changes in your code. See the source code for more implementation examples. If you are using CLI and CLI-completion, make sure to re-generate the completion file.

- A new inner function `list_create_inner_file` has been added, using which it's possible to create just a list of given files (ignoring `count` value) using given arguments. The amount of files is determined by the `func_list` (each pair (`Callable`, `kwargs`) corresponds to a single file. Both `ZipFileProvider` and `TarFileProvider` have been altered to reflect these changes.
- Added to support for XML files through `XmlFileProvider`.

### 14.10.18 0.13

2023-05-05

---

**Note:** This release introduces minor backwards incompatible changes.

---

- Display full path to the created file in the CLI.
- Added `DynamicTemplate` support for PDF file. The `generate` method of the `BasePdfGenerator` and classes derived from it, got two new arguments: `data` (`Dict[str, Any]`), and `provider` (`Union[Faker, Generator, Provider]`). If you have implemented custom generators for PDF (`pdf_file` provider), make sure to reflect mentioned changes in your code.

### 14.10.19 0.12.6

2023-05-02

- Added `DynamicTemplate` support for DOCX and ODT files.

### 14.10.20 0.12.5

2023-04-24

---

**Note:** In memory of the victims of the [Armenian Genocide](#).

---

- Expose `mp3_generator_cls` and `pdf_generator_cls` CLI options for `mp3_file` and `pdf_file` respectively.
- Add `num_files` CLI option for all providers.

### 14.10.21 0.12.4

2023-04-22

- Make it possible to load classes from strings for passing as arguments to `mp3_file` and `pdf_file` providers.

### 14.10.22 0.12.3

2023-04-21

- Fixes in CLI options.

### 14.10.23 0.12.2

2023-04-20

- Fixes in CLI options.

### 14.10.24 0.12.1

2023-04-19

- Added CLI options.

### 14.10.25 0.12

2023-02-24

*Note, that this release introduces breaking changes!*

- Make it easy to use a different PDF library with PdfFileProvider by adding `pdf_generator_cls` and `pdf_generator_kwargs` optional arguments to the `pdf_file` method. Added `ReportlabPdfGenerator` class based on the famous `reportlab` library. Default is still `PdfkitPdfGenerator`. Since encoding was something specific for `pdfkit` library, it was moved from `pdf_file` method to `PdfkitPdfGenerator`, to which it can be passed in `pdf_generator_kwargs`. If you have passed the `encoding` argument explicitly, make sure to make correspondent changes. Note, that using the new `ReportlabPdfGenerator` class could speed-up PDF generation by about 40 times.

### 14.10.26 0.11.5

2023-02-20

- Fixes in typing of `CsvFileProvider`. `Tuple[str, str]` becomes `Tuple[str, ...]`.

### 14.10.27 0.11.4

2023-02-16

---

**Note:** Release dedicated to my dear valentine - Anahit.

---

- Added `filename` to data property of values returned by `Mp3FileProvider` provider (`StringValue`, `BytesValue`).

### **14.10.28 0.11.3**

2023-02-10

- Moved several interface classes from one location to another. If you haven't implemented custom generators, this won't affect you. If you did, make sure to update your imports:
  - `BaseTextAugmenter` has been moved from `faker_file.providers.augment_file_from_dir.augmenters.base` to `faker_file.providers.base.text_augmenter`.
  - `BaseTextExtractor` has been moved from `faker_file.providers.augment_file_from_dir.extractors.base` to `faker_file.providers.base.text_extractor`.
  - `BaseMp3Generator` has been moved from `faker_file.providers.mp3_file.generators.base` to `faker_file.providers.base.mp3_generator`.

### **14.10.29 0.11.2**

2023-02-07

- Add `filename` to data property of values returned by providers (`StringValue`, `BytesValue`).

### **14.10.30 0.11.1**

2023-01-31

- Documentation improvements.
- MyPy fixes.

### **14.10.31 0.11**

2023-01-25

- Allow returning binary contents of the file by providing the `raw=True` argument (`False` by default, works with all provider classes and inner functions). If you have subclassed or overridden provider classes or written custom inner functions, make sure to reflect the changes in your code.

### **14.10.32 0.10.12**

2023-01-21

- Add `TarFileProvider` and `create_inner_tar_file` function.
- Add `OdpFileProvider` and `create_inner_odp_file` function.

### 14.10.33 0.10.11

2023-01-20

- Improve EPUB document layout.
- Improve PDF document layout.
- Minor documentation improvements.

### 14.10.34 0.10.10

2023-01-19

- Allow passing `model_name` and `action` arguments to the `ContextualWordEmbeddingsAugmenter`.
- Replace `bert-base-cased` with `bert-base-multilingual-cased` as a default model for `ContextualWordEmbeddingsAugmenter`.
- Improve PPTX document layout.
- Minor fixes in documentation.

### 14.10.35 0.10.9

2023-01-18

- Add an installation directive `[common]` to install everything except ML libraries.
- Added testing of UTF8 content.

### 14.10.36 0.10.8

2023-01-16

- Switch to PyPI releases of `gtts`.
- Stop testing against Django 3.0 and 3.1.
- Documentation improvements.
- Tests improvements.

### 14.10.37 0.10.7

2023-01-13

- Add `OdtFileProvider` and `create_inner_odt_file` function.
- Documentation improvements.
- Async related deprecation fixes in `EdgeTtsMp3Generator`.
- Optimize example factories.

### **14.10.38 0.10.6**

2023-01-11

- Add `AugmentFileFromDirProvider` provider for making augmented copies of randomly picked files from given directory.
- Documentation improvements.
- Fixes in setup.

### **14.10.39 0.10.5**

2023-01-09

- Add `fuzzy_choice_create_inner_file` inner function for easy diversion of files within archives (ZIP, EML).
- Documentation improvements.
- Add `MaryTTS` example (another MP3 generator for `Mp3FileProvider`).

### **14.10.40 0.10.4**

2023-01-08

- Add missing `mp3_generator_kwargs` argument to the `create_inner_mp3_file` function.
- Clean-up.

### **14.10.41 0.10.3**

2023-01-07

Improvements of the `Mp3FileProvider` module:

- Pass active generator to the `Mp3FileProvider` in the `generator` argument if `BaseMp3Generator` (and all implementations).
- Introduce `handle_kwargs` method in the `BaseMp3Generator` to handle arbitrary provider specific tuning.
- Add `EdgeTtsMp3Generator` MP3 generator.
- Add `mp3_generator_kwargs` argument to the `Mp3FileProvider.mp3_file` method.

### **14.10.42 0.10.2**

2023-01-06

- Add `Mp3FileProvider`.
- Add `create_inner_mp3_file` inner function.

### 14.10.43 0.10.1

2023-01-05

- Fixes in ZipFileProvider.

### 14.10.44 0.10

2023-01-04

*Note, that this release introduces breaking changes!*

- Move all `create_inner*_file` functions from `faker_file.providers.zip_file` to `faker_file.providers.helpers.inner` module. Adjust your imports accordingly.
- Add `EmlFileProvider`.
- Add `create_inner_eml_file` inner function.

### 14.10.45 0.9.3

2023-01-03

- Add `EpubFileProvider` provider.

### 14.10.46 0.9.2

2022-12-23

- Add `RrfFileProvider`.
- Added SQLAlchemy factory example.

### 14.10.47 0.9.1

2022-12-19

- Fixes in cloud storage.
- Documentation fixes.

### 14.10.48 0.9

2022-12-17

- Add optional encoding argument to `CsvFileProvider` and `PdfFileProvider` providers.
- Add `root_path` argument to cloud storages.
- Moved all image related code (`IcoFileProvider`, `JpegFileProvider`, `PngFileProvider`, `SvgFileProvider`, `WebpFileProvider`) to `ImageMixin`. Moved all tabular data related code (`OdsFileProvider`, `XlsxFileProvider`) to `TabularDataMixin`.
- Documentation improvements.

### 14.10.49 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.

### 14.10.50 0.7

2022-12-12

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

### 14.10.51 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%.

### 14.10.52 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",
    }
)
```



### 14.10.53 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.

### 14.10.54 0.3

2022-12-08

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

### 14.10.55 0.2

2022-12-07

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

### 14.10.56 0.1

2022-12-06

- Initial beta release.

## 14.11 Package

### 14.11.1 `faker_file` package

#### 14.11.1.1 Subpackages

##### 14.11.1.1.1 `faker_file.providers` package

###### 14.11.1.1.1.1 Subpackages

###### 14.11.1.1.1.2 `faker_file.providers.augment_file_from_dir` package

###### 14.11.1.1.1.3 Subpackages

###### 14.11.1.1.1.4 `faker_file.providers.augment_file_from_dir.augmenters` package

###### 14.11.1.1.1.5 Submodules

###### 14.11.1.1.1.6 `faker_file.providers.augment_file_from_dir.augmenters.nlpaug_augmenter` module

###### 14.11.1.1.1.7 Module contents

###### 14.11.1.1.1.8 `faker_file.providers.augment_file_from_dir.extractors` package

###### 14.11.1.1.1.9 Submodules

###### 14.11.1.1.1.10 `faker_file.providers.augment_file_from_dir.extractors.tika_extractor` module

###### 14.11.1.1.1.11 Module contents

###### 14.11.1.1.1.12 Module contents

###### 14.11.1.1.1.13 `faker_file.providers.base` package

###### 14.11.1.1.1.14 Submodules

###### 14.11.1.1.1.15 `faker_file.providers.base.mp3_generator` module

```
class faker_file.providers.base.mp3_generator.BaseMp3Generator(content: str, generator:  
Union[Faker, Generator,  
Provider], **kwargs)
```

Bases: `object`

Base MP3 generator.

**content:** `str`

**generate**(*\*\*kwargs*) → bytes

**generator**: Union[Faker, Generator, Provider]

**handle\_kwargs**(*\*\*kwargs*)

Handle kwargs.

#### 14.11.1.1.16 `faker_file.providers.base.pdf_generator` module

**class** `faker_file.providers.base.pdf_generator.BasePdfGenerator`(*\*\*kwargs*)

Bases: object

Base PDF generator.

**generate**(*content*: str, *data*: Dict[str, Any], *provider*: Union[Faker, Generator, Provider]) → bytes

**handle\_kwargs**(*\*\*kwargs*)

Handle kwargs.

#### 14.11.1.1.17 `faker_file.providers.base.text_augmenter` module

**class** `faker_file.providers.base.text_augmenter.BaseTextAugmenter`(*\*\*kwargs*)

Bases: object

Base text augmenter.

**augment**(*text*: str) → str

**handle\_kwargs**(*\*\*kwargs*)

Handle kwargs.

#### 14.11.1.1.18 `faker_file.providers.base.text_extractor` module

**class** `faker_file.providers.base.text_extractor.BaseTextExtractor`(*\*\*kwargs*)

Bases: object

Base text extractor.

**extract**(*source\_file*: Union[Path, str]) → str

**handle\_kwargs**(*\*\*kwargs*)

Handle kwargs.

**path**: str

#### 14.11.1.1.19 Module contents

#### 14.11.1.1.20 `faker_file.providers.helpers` package

#### 14.11.1.1.21 Submodules

#### 14.11.1.1.22 `faker_file.providers.helpers.inner` module

`faker_file.providers.helpers.inner.create_inner_bin_file`(*storage*: *Optional*[[BaseStorage](#)] = *None*,  
*basename*: *Optional*[*str*] = *None*, *prefix*:  
*Optional*[*str*] = *None*, *generator*:  
*Optional*[*Union*[*Faker*, *Generator*,  
*Provider*]] = *None*, *length*: *int* = 1 \* 1024  
\* 1024, *content*: *Optional*[*bytes*] = *None*,  
*raw*: *bool* = *True*, *\*\*kwargs*) →  
[BytesValue](#)

`faker_file.providers.helpers.inner.create_inner_bin_file`(*storage*: *Optional*[[BaseStorage](#)] = *None*,  
*basename*: *Optional*[*str*] = *None*, *prefix*:  
*Optional*[*str*] = *None*, *generator*:  
*Optional*[*Union*[*Faker*, *Generator*,  
*Provider*]] = *None*, *length*: *int* = 1 \* 1024  
\* 1024, *content*: *Optional*[*bytes*] = *None*,  
*\*\*kwargs*) → [StringValue](#)

Create inner BIN file.

`faker_file.providers.helpers.inner.create_inner_csv_file`(*storage*: *Optional*[[BaseStorage](#)] = *None*,  
*basename*: *Optional*[*str*] = *None*, *prefix*:  
*Optional*[*str*] = *None*, *generator*:  
*Optional*[*Union*[*Faker*, *Generator*,  
*Provider*]] = *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*,  
*format\_func*: *Callable*[[*Union*[*Faker*,  
*Generator*, *Provider*], *str*], *str*] =  
*DEFAULT\_FORMAT\_FUNC*, *raw*: *bool*  
= *True*, *\*\*kwargs*) → [BytesValue](#)

`faker_file.providers.helpers.inner.create_inner_csv_file`(*storage*: *Optional*[[BaseStorage](#)] = *None*,  
*basename*: *Optional*[*str*] = *None*, *prefix*:  
*Optional*[*str*] = *None*, *generator*:  
*Optional*[*Union*[*Faker*, *Generator*,  
*Provider*]] = *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*,  
*format\_func*: *Callable*[[*Union*[*Faker*,  
*Generator*, *Provider*], *str*], *str*] =  
*DEFAULT\_FORMAT\_FUNC*, *\*\*kwargs*)  
→ [StringValue](#)

Create inner CSV file.

```
faker_file.providers.helpers.inner.create_inner_docx_file(storage: Optional[BaseStorage] = None,
                                                         basename: Optional[str] = None, prefix:
                                                         Optional[str] = None, generator:
                                                         Optional[Union[Faker, Generator,
                                                         Provider]] = None, max_nb_chars: int =
                                                         DEFAULT_TEXT_MAX_NB_CHARS,
                                                         wrap_chars_after: Optional[int] =
                                                         None, content: Optional[str] = None,
                                                         format_func: Callable[[Union[Faker,
                                                         Generator, Provider], str], str] =
                                                         DEFAULT_FORMAT_FUNC, raw: bool
                                                         = True, **kwargs) → BytesValue
```

```
faker_file.providers.helpers.inner.create_inner_docx_file(storage: Optional[BaseStorage] = None,
                                                         basename: Optional[str] = None, prefix:
                                                         Optional[str] = None, generator:
                                                         Optional[Union[Faker, Generator,
                                                         Provider]] = None, max_nb_chars: int =
                                                         DEFAULT_TEXT_MAX_NB_CHARS,
                                                         wrap_chars_after: Optional[int] =
                                                         None, content: Optional[str] = None,
                                                         format_func: Callable[[Union[Faker,
                                                         Generator, Provider], str], str] =
                                                         DEFAULT_FORMAT_FUNC,
                                                         **kwargs) → StringValue
```

Create inner DOCX file.

```
faker_file.providers.helpers.inner.create_inner eml_file(storage: Optional[BaseStorage] = None,
                                                         basename: Optional[str] = None, prefix:
                                                         Optional[str] = None, generator:
                                                         Optional[Union[Faker, Generator,
                                                         Provider]] = None, options:
                                                         Optional[Dict[str, Any]] = None,
                                                         max_nb_chars: int =
                                                         DEFAULT_TEXT_MAX_NB_CHARS,
                                                         wrap_chars_after: Optional[int] = None,
                                                         content: Optional[str] = None,
                                                         format_func: Callable[[Union[Faker,
                                                         Generator, Provider], str], str] =
                                                         DEFAULT_FORMAT_FUNC, raw: bool
                                                         = True, **kwargs) → BytesValue
```

```
faker_file.providers.helpers.inner.create_inner_eml_file(storage: Optional[BaseStorage] = None,
                                                         basename: Optional[str] = None, prefix:
                                                         Optional[str] = None, generator:
                                                         Optional[Union[Faker, Generator,
                                                         Provider]] = None, options:
                                                         Optional[Dict[str, Any]] = None,
                                                         max_nb_chars: int =
                                                         DEFAULT_TEXT_MAX_NB_CHARS,
                                                         wrap_chars_after: Optional[int] = None,
                                                         content: Optional[str] = None,
                                                         format_func: Callable[[Union[Faker,
                                                         Generator, Provider], str], str] =
                                                         DEFAULT_FORMAT_FUNC, **kwargs)
                                                         → StringValue
```

Create inner EML file.

```
faker_file.providers.helpers.inner.create_inner_epub_file(storage: Optional[BaseStorage] = None,
                                                           basename: Optional[str] = None, prefix:
                                                           Optional[str] = None, generator:
                                                           Optional[Union[Faker, Generator,
                                                           Provider]] = None, max_nb_chars: int =
                                                           DEFAULT_TEXT_MAX_NB_CHARS,
                                                           wrap_chars_after: Optional[int] =
                                                           None, content: Optional[str] = None,
                                                           title: Optional[str] = None,
                                                           chapter_title: Optional[str] = None,
                                                           format_func: Callable[[Union[Faker,
                                                           Generator, Provider], str], str] =
                                                           DEFAULT_FORMAT_FUNC, raw: bool
                                                           = True, **kwargs) → BytesValue
```

```
faker_file.providers.helpers.inner.create_inner_epub_file(storage: Optional[BaseStorage] = None,
                                                           basename: Optional[str] = None, prefix:
                                                           Optional[str] = None, generator:
                                                           Optional[Union[Faker, Generator,
                                                           Provider]] = None, max_nb_chars: int =
                                                           DEFAULT_TEXT_MAX_NB_CHARS,
                                                           wrap_chars_after: Optional[int] =
                                                           None, content: Optional[str] = None,
                                                           title: Optional[str] = None,
                                                           chapter_title: Optional[str] = None,
                                                           format_func: Callable[[Union[Faker,
                                                           Generator, Provider], str], str] =
                                                           DEFAULT_FORMAT_FUNC,
                                                           **kwargs) → StringValue
```

Create inner EPUB file.

```
faker_file.providers.helpers.inner.create_inner_file_from_path(path: str, storage:
                                                                Optional[BaseStorage] = None,
                                                                basename: Optional[str] = None,
                                                                prefix: Optional[str] = None,
                                                                generator: Optional[Union[Faker,
                                                                Generator, Provider]] = None,
                                                                raw: bool = True, **kwargs) →
                                                                BytesValue
```

```
faker_file.providers.helpers.inner.create_inner_file_from_path(path: str, storage:
    Optional[BaseStorage] = None,
    basename: Optional[str] = None,
    prefix: Optional[str] = None,
    generator: Optional[Union[Faker,
        Generator, Provider]] = None,
    **kwargs) → StringValue
```

Create inner file from path.

```
faker_file.providers.helpers.inner.create_inner_generic_file(content: Union[bytes, str], extension:
    str, storage: Optional[BaseStorage]
    = None, basename: Optional[str] =
    None, prefix: Optional[str] = None,
    generator: Optional[Union[Faker,
        Generator, Provider]] = None,
    format_func:
    Callable[[Union[Faker, Generator,
        Provider], str], str] =
    DEFAULT_FORMAT_FUNC, raw:
    bool = True, **kwargs) →
    BytesValue
```

```
faker_file.providers.helpers.inner.create_inner_generic_file(content: Union[bytes, str], extension:
    str, storage: Optional[BaseStorage]
    = None, basename: Optional[str] =
    None, prefix: Optional[str] = None,
    generator: Optional[Union[Faker,
        Generator, Provider]] = None,
    format_func:
    Callable[[Union[Faker, Generator,
        Provider], str], str] =
    DEFAULT_FORMAT_FUNC,
    **kwargs) → StringValue
```

Create inner generic file.

```
faker_file.providers.helpers.inner.create_inner_ico_file(storage: Optional[BaseStorage] = None,
    basename: Optional[str] = None, prefix:
    Optional[str] = None, generator:
    Optional[Union[Faker, Generator,
        Provider]] = None, max_nb_chars: int =
    DEFAULT_IMAGE_MAX_NB_CHARS,
    wrap_chars_after: Optional[int] = None,
    content: Optional[str] = None,
    format_func: Callable[[Union[Faker,
        Generator, Provider], str], str] =
    DEFAULT_FORMAT_FUNC, raw: bool
    = True, **kwargs) → BytesValue
```

```
faker_file.providers.helpers.inner.create_inner_ico_file(storage: Optional[BaseStorage] = None,
                                                         basename: Optional[str] = None, prefix:
                                                         Optional[str] = None, generator:
                                                         Optional[Union[Faker, Generator,
                                                         Provider]] = None, max_nb_chars: int =
                                                         DEFAULT_IMAGE_MAX_NB_CHARS,
                                                         wrap_chars_after: Optional[int] = None,
                                                         content: Optional[str] = None,
                                                         format_func: Callable[[Union[Faker,
                                                         Generator, Provider], str], str] =
                                                         DEFAULT_FORMAT_FUNC, **kwargs)
                                                         → StringValue
```

Create inner ICO file.

```
faker_file.providers.helpers.inner.create_inner_jpeg_file(storage: Optional[BaseStorage] = None,
                                                           basename: Optional[str] = None, prefix:
                                                           Optional[str] = None, generator:
                                                           Optional[Union[Faker, Generator,
                                                           Provider]] = None, max_nb_chars: int =
                                                           DEFAULT_IMAGE_MAX_NB_CHARS,
                                                           wrap_chars_after: Optional[int] =
                                                           None, content: Optional[str] = None,
                                                           format_func: Callable[[Union[Faker,
                                                           Generator, Provider], str], str] =
                                                           DEFAULT_FORMAT_FUNC, raw: bool
                                                           = True, **kwargs) → BytesValue
```

```
faker_file.providers.helpers.inner.create_inner_jpeg_file(storage: Optional[BaseStorage] = None,
                                                           basename: Optional[str] = None, prefix:
                                                           Optional[str] = None, generator:
                                                           Optional[Union[Faker, Generator,
                                                           Provider]] = None, max_nb_chars: int =
                                                           DEFAULT_IMAGE_MAX_NB_CHARS,
                                                           wrap_chars_after: Optional[int] =
                                                           None, content: Optional[str] = None,
                                                           format_func: Callable[[Union[Faker,
                                                           Generator, Provider], str], str] =
                                                           DEFAULT_FORMAT_FUNC,
                                                           **kwargs) → StringValue
```

Create inner JPEG file.



```
faker_file.providers.helpers.inner.create_inner_mp3_file(storage: Optional[BaseStorage] = None,
                                                         basename: Optional[str] = None, prefix:
                                                         Optional[str] = None, generator:
                                                         Optional[Union[Faker, Generator,
                                                         Provider]] = None, max_nb_chars: int =
                                                         DEFAULT_AUDIO_MAX_NB_CHARS,
                                                         content: Optional[str] = None,
                                                         mp3_generator_cls: Optional[Union[str,
                                                         Type[BaseMp3Generator]]] = None,
                                                         mp3_generator_kwargs:
                                                         Optional[Dict[str, Any]] = None,
                                                         format_func: Callable[[Union[Faker,
                                                         Generator, Provider], str], str] =
                                                         DEFAULT_FORMAT_FUNC, raw: bool
                                                         = True, **kwargs) → BytesValue
```

```
faker_file.providers.helpers.inner.create_inner_mp3_file(storage: Optional[BaseStorage] = None,
                                                         basename: Optional[str] = None, prefix:
                                                         Optional[str] = None, generator:
                                                         Optional[Union[Faker, Generator,
                                                         Provider]] = None, max_nb_chars: int =
                                                         DEFAULT_AUDIO_MAX_NB_CHARS,
                                                         content: Optional[str] = None,
                                                         mp3_generator_cls: Optional[Union[str,
                                                         Type[BaseMp3Generator]]] = None,
                                                         mp3_generator_kwargs:
                                                         Optional[Dict[str, Any]] = None,
                                                         format_func: Callable[[Union[Faker,
                                                         Generator, Provider], str], str] =
                                                         DEFAULT_FORMAT_FUNC, **kwargs)
                                                         → StringValue
```

Create inner ODS file.

```
faker_file.providers.helpers.inner.create_inner_odp_file(storage: Optional[BaseStorage] = None,
                                                         basename: Optional[str] = None, prefix:
                                                         Optional[str] = None, generator:
                                                         Optional[Union[Faker, Generator,
                                                         Provider]] = None, max_nb_chars: int =
                                                         DEFAULT_TEXT_MAX_NB_CHARS,
                                                         wrap_chars_after: Optional[int] = None,
                                                         content: Optional[str] = None,
                                                         format_func: Callable[[Union[Faker,
                                                         Generator, Provider], str], str] =
                                                         DEFAULT_FORMAT_FUNC, raw: bool
                                                         = True, **kwargs) → BytesValue
```

```
faker_file.providers.helpers.inner.create_inner_odp_file(storage: Optional[BaseStorage] = None,
                                                         basename: Optional[str] = None, prefix:
                                                         Optional[str] = None, generator:
                                                         Optional[Union[Faker, Generator,
                                                         Provider]] = None, max_nb_chars: int =
                                                         DEFAULT_TEXT_MAX_NB_CHARS,
                                                         wrap_chars_after: Optional[int] = None,
                                                         content: Optional[str] = None,
                                                         format_func: Callable[[Union[Faker,
                                                         Generator, Provider], str], str] =
                                                         DEFAULT_FORMAT_FUNC, **kwargs)
                                                         → Union[BytesValue, StringValue]
```

Create inner ODP file.

```
faker_file.providers.helpers.inner.create_inner_ods_file(storage: Optional[BaseStorage] = None,
                                                         basename: Optional[str] = None, prefix:
                                                         Optional[str] = None, generator:
                                                         Optional[Union[Faker, Generator,
                                                         Provider]] = None, data_columns:
                                                         Optional[Dict[str, str]] = None,
                                                         num_rows: int = 10, content:
                                                         Optional[str] = None, format_func:
                                                         Callable[[Union[Faker, Generator,
                                                         Provider], str], str] =
                                                         DEFAULT_FORMAT_FUNC, raw: bool
                                                         = True, **kwargs) → BytesValue
```

```
faker_file.providers.helpers.inner.create_inner_ods_file(storage: Optional[BaseStorage] = None,
                                                         basename: Optional[str] = None, prefix:
                                                         Optional[str] = None, generator:
                                                         Optional[Union[Faker, Generator,
                                                         Provider]] = None, data_columns:
                                                         Optional[Dict[str, str]] = None,
                                                         num_rows: int = 10, content:
                                                         Optional[str] = None, format_func:
                                                         Callable[[Union[Faker, Generator,
                                                         Provider], str], str] =
                                                         DEFAULT_FORMAT_FUNC, **kwargs)
                                                         → StringValue
```

Create inner ODS file.

```
faker_file.providers.helpers.inner.create_inner_odt_file(storage: Optional[BaseStorage] = None,
                                                         basename: Optional[str] = None, prefix:
                                                         Optional[str] = None, generator:
                                                         Optional[Union[Faker, Generator,
                                                         Provider]] = None, max_nb_chars: int =
                                                         DEFAULT_TEXT_MAX_NB_CHARS,
                                                         wrap_chars_after: Optional[int] = None,
                                                         content: Optional[str] = None,
                                                         format_func: Callable[[Union[Faker,
                                                         Generator, Provider], str], str] =
                                                         DEFAULT_FORMAT_FUNC, raw: bool
                                                         = True, **kwargs) → BytesValue
```

```
faker_file.providers.helpers.inner.create_inner_odt_file(storage: Optional[BaseStorage] = None,
                                                         basename: Optional[str] = None, prefix:
                                                         Optional[str] = None, generator:
                                                         Optional[Union[Faker, Generator,
                                                         Provider]] = None, max_nb_chars: int =
                                                         DEFAULT_TEXT_MAX_NB_CHARS,
                                                         wrap_chars_after: Optional[int] = None,
                                                         content: Optional[str] = None,
                                                         format_func: Callable[[Union[Faker,
                                                         Generator, Provider], str], str] =
                                                         DEFAULT_FORMAT_FUNC, **kwargs)
                                                         → StringValue
```

Create inner ODT file.

```
faker_file.providers.helpers.inner.create_inner_pdf_file(storage: Optional[BaseStorage] = None,
                                                         basename: Optional[str] = None, prefix:
                                                         Optional[str] = None, generator:
                                                         Optional[Union[Faker, Generator,
                                                         Provider]] = None, max_nb_chars: int =
                                                         DEFAULT_TEXT_MAX_NB_CHARS,
                                                         wrap_chars_after: Optional[int] = None,
                                                         content: Optional[str] = None,
                                                         pdf_generator_cls: Optional[Union[str,
                                                         Type[BasePdfGenerator]]] = None,
                                                         pdf_generator_kwargs: Optional[Dict[str,
                                                         Any]] = None, format_func:
                                                         Callable[[Union[Faker, Generator,
                                                         Provider], str], str] =
                                                         DEFAULT_FORMAT_FUNC, raw: bool
                                                         = True, **kwargs) → BytesValue
```

```
faker_file.providers.helpers.inner.create_inner_pdf_file(storage: Optional[BaseStorage] = None,
                                                         basename: Optional[str] = None, prefix:
                                                         Optional[str] = None, generator:
                                                         Optional[Union[Faker, Generator,
                                                         Provider]] = None, max_nb_chars: int =
                                                         DEFAULT_TEXT_MAX_NB_CHARS,
                                                         wrap_chars_after: Optional[int] = None,
                                                         content: Optional[str] = None,
                                                         pdf_generator_cls: Optional[Union[str,
                                                         Type[BasePdfGenerator]]] = None,
                                                         pdf_generator_kwargs: Optional[Dict[str,
                                                         Any]] = None, format_func:
                                                         Callable[[Union[Faker, Generator,
                                                         Provider], str], str] =
                                                         DEFAULT_FORMAT_FUNC, **kwargs)
                                                         → StringValue
```

Create inner PDF file.

```
faker_file.providers.helpers.inner.create_inner_png_file(storage: Optional[BaseStorage] = None,
                                                         basename: Optional[str] = None, prefix:
                                                         Optional[str] = None, generator:
                                                         Optional[Union[Faker, Generator,
                                                         Provider]] = None, max_nb_chars: int =
                                                         DEFAULT_IMAGE_MAX_NB_CHARS,
                                                         wrap_chars_after: Optional[int] = None,
                                                         content: Optional[str] = None,
                                                         format_func: Callable[[Union[Faker,
                                                         Generator, Provider], str], str] =
                                                         DEFAULT_FORMAT_FUNC, raw: bool
                                                         = True, **kwargs) → BytesValue
```

```
faker_file.providers.helpers.inner.create_inner_png_file(storage: Optional[BaseStorage] = None,
                                                         basename: Optional[str] = None, prefix:
                                                         Optional[str] = None, generator:
                                                         Optional[Union[Faker, Generator,
                                                         Provider]] = None, max_nb_chars: int =
                                                         DEFAULT_IMAGE_MAX_NB_CHARS,
                                                         wrap_chars_after: Optional[int] = None,
                                                         content: Optional[str] = None,
                                                         format_func: Callable[[Union[Faker,
                                                         Generator, Provider], str], str] =
                                                         DEFAULT_FORMAT_FUNC, **kwargs)
                                                         → StringValue
```

Create inner PNG file.

```
faker_file.providers.helpers.inner.create_inner_pptx_file(storage: Optional[BaseStorage] = None,
                                                          basename: Optional[str] = None, prefix:
                                                          Optional[str] = None, generator:
                                                          Optional[Union[Faker, Generator,
                                                          Provider]] = None, max_nb_chars: int =
                                                          DEFAULT_TEXT_MAX_NB_CHARS,
                                                          wrap_chars_after: Optional[int] =
                                                          None, content: Optional[str] = None,
                                                          format_func: Callable[[Union[Faker,
                                                          Generator, Provider], str], str] =
                                                          DEFAULT_FORMAT_FUNC, raw: bool
                                                          = True, **kwargs) → BytesValue
```

```
faker_file.providers.helpers.inner.create_inner_pptx_file(storage: Optional[BaseStorage] = None,
                                                          basename: Optional[str] = None, prefix:
                                                          Optional[str] = None, generator:
                                                          Optional[Union[Faker, Generator,
                                                          Provider]] = None, max_nb_chars: int =
                                                          DEFAULT_TEXT_MAX_NB_CHARS,
                                                          wrap_chars_after: Optional[int] =
                                                          None, content: Optional[str] = None,
                                                          format_func: Callable[[Union[Faker,
                                                          Generator, Provider], str], str] =
                                                          DEFAULT_FORMAT_FUNC,
                                                          **kwargs) → StringValue
```

Create inner PPTX file.

```
faker_file.providers.helpers.inner.create_inner_random_file_from_dir(source_dir_path: str,
                                                                    storage:
                                                                    Optional[BaseStorage] =
                                                                    None, basename:
                                                                    Optional[str] = None,
                                                                    prefix: Optional[str] =
                                                                    None, generator:
                                                                    Optional[Union[Faker,
                                                                    Generator, Provider]] =
                                                                    None, raw: bool = True,
                                                                    **kwargs) → BytesValue
```

```
faker_file.providers.helpers.inner.create_inner_random_file_from_dir(source_dir_path: str,
                                                                    storage:
                                                                    Optional[BaseStorage] =
                                                                    None, basename:
                                                                    Optional[str] = None,
                                                                    prefix: Optional[str] =
                                                                    None, generator:
                                                                    Optional[Union[Faker,
                                                                    Generator, Provider]] =
                                                                    None, **kwargs) →
                                                                    StringValue
```

Create inner random\_file\_from\_dir file.

```
faker_file.providers.helpers.inner.create_inner_rtf_file(storage: Optional[BaseStorage] = None,
                                                         basename: Optional[str] = None, prefix:
                                                         Optional[str] = None, generator:
                                                         Optional[Union[Faker, Generator,
                                                         Provider]] = None, max_nb_chars: int =
                                                         DEFAULT_TEXT_MAX_NB_CHARS,
                                                         wrap_chars_after: Optional[int] = None,
                                                         content: Optional[str] = None,
                                                         format_func: Callable[[Union[Faker,
                                                         Generator, Provider], str], str] =
                                                         DEFAULT_FORMAT_FUNC, raw: bool
                                                         = True, **kwargs) → BytesValue
```

```
faker_file.providers.helpers.inner.create_inner_rtf_file(storage: Optional[BaseStorage] = None,
                                                         basename: Optional[str] = None, prefix:
                                                         Optional[str] = None, generator:
                                                         Optional[Union[Faker, Generator,
                                                         Provider]] = None, max_nb_chars: int =
                                                         DEFAULT_TEXT_MAX_NB_CHARS,
                                                         wrap_chars_after: Optional[int] = None,
                                                         content: Optional[str] = None,
                                                         format_func: Callable[[Union[Faker,
                                                         Generator, Provider], str], str] =
                                                         DEFAULT_FORMAT_FUNC, **kwargs)
                                                         → StringValue
```

Create inner RTF file.

```
faker_file.providers.helpers.inner.create_inner_svg_file(storage: Optional[BaseStorage] = None,
                                                         basename: Optional[str] = None, prefix:
                                                         Optional[str] = None, generator:
                                                         Optional[Union[Faker, Generator,
                                                         Provider]] = None, max_nb_chars: int =
                                                         DEFAULT_IMAGE_MAX_NB_CHARS,
                                                         wrap_chars_after: Optional[int] = None,
                                                         content: Optional[str] = None,
                                                         format_func: Callable[[Union[Faker,
                                                         Generator, Provider], str], str] =
                                                         DEFAULT_FORMAT_FUNC, raw: bool
                                                         = True, **kwargs) → BytesValue
```

```
faker_file.providers.helpers.inner.create_inner_svg_file(storage: Optional[BaseStorage] = None,
                                                         basename: Optional[str] = None, prefix:
                                                         Optional[str] = None, generator:
                                                         Optional[Union[Faker, Generator,
                                                         Provider]] = None, max_nb_chars: int =
                                                         DEFAULT_IMAGE_MAX_NB_CHARS,
                                                         wrap_chars_after: Optional[int] = None,
                                                         content: Optional[str] = None,
                                                         format_func: Callable[[Union[Faker,
                                                         Generator, Provider], str], str] =
                                                         DEFAULT_FORMAT_FUNC, **kwargs)
                                                         → StringValue
```

Create inner SVG file.

```
faker_file.providers.helpers.inner.create_inner_tar_file(storage: Optional[BaseStorage] = None,
                                                         basename: Optional[str] = None, prefix:
                                                         Optional[str] = None, generator:
                                                         Optional[Union[Faker, Generator,
                                                         Provider]] = None, options:
                                                         Optional[Dict[str, Any]] = None,
                                                         compression: Optional[str] = None, raw:
                                                         bool = True, **kwargs) → BytesValue
```

```
faker_file.providers.helpers.inner.create_inner_tar_file(storage: Optional[BaseStorage] = None,
                                                         basename: Optional[str] = None, prefix:
                                                         Optional[str] = None, generator:
                                                         Optional[Union[Faker, Generator,
                                                         Provider]] = None, options:
                                                         Optional[Dict[str, Any]] = None,
                                                         compression: Optional[str] = None,
                                                         **kwargs) → StringValue
```

Create inner TAR file.

```
faker_file.providers.helpers.inner.create_inner_txt_file(storage: Optional[BaseStorage] = None,
                                                         basename: Optional[str] = None, prefix:
                                                         Optional[str] = None, generator:
                                                         Optional[Union[Faker, Generator,
                                                         Provider]] = None, max_nb_chars: int =
                                                         DEFAULT_TEXT_MAX_NB_CHARS,
                                                         wrap_chars_after: Optional[int] = None,
                                                         content: Optional[str] = None,
                                                         format_func: Callable[[Union[Faker,
                                                         Generator, Provider], str], str] =
                                                         DEFAULT_FORMAT_FUNC, raw: bool
                                                         = True, **kwargs) → BytesValue
```

```
faker_file.providers.helpers.inner.create_inner_txt_file(storage: Optional[BaseStorage] = None,
                                                         basename: Optional[str] = None, prefix:
                                                         Optional[str] = None, generator:
                                                         Optional[Union[Faker, Generator,
                                                         Provider]] = None, max_nb_chars: int =
                                                         DEFAULT_TEXT_MAX_NB_CHARS,
                                                         wrap_chars_after: Optional[int] = None,
                                                         content: Optional[str] = None,
                                                         format_func: Callable[[Union[Faker,
                                                         Generator, Provider], str], str] =
                                                         DEFAULT_FORMAT_FUNC, **kwargs)
                                                         → StringValue
```

Create inner TXT file.

```
faker_file.providers.helpers.inner.create_inner_webp_file(storage: Optional[BaseStorage] = None,
                                                           basename: Optional[str] = None, prefix:
                                                           Optional[str] = None, generator:
                                                           Optional[Union[Faker, Generator,
                                                           Provider]] = None, max_nb_chars: int =
                                                           DEFAULT_IMAGE_MAX_NB_CHARS,
                                                           wrap_chars_after: Optional[int] =
                                                           None, content: Optional[str] = None,
                                                           format_func: Callable[[Union[Faker,
                                                           Generator, Provider], str], str] =
                                                           DEFAULT_FORMAT_FUNC, raw: bool
                                                           = True, **kwargs) → BytesValue
```

```
faker_file.providers.helpers.inner.create_inner_webp_file(storage: Optional[BaseStorage] = None,
                                                           basename: Optional[str] = None, prefix:
                                                           Optional[str] = None, generator:
                                                           Optional[Union[Faker, Generator,
                                                           Provider]] = None, max_nb_chars: int =
                                                           DEFAULT_IMAGE_MAX_NB_CHARS,
                                                           wrap_chars_after: Optional[int] =
                                                           None, content: Optional[str] = None,
                                                           format_func: Callable[[Union[Faker,
                                                           Generator, Provider], str], str] =
                                                           DEFAULT_FORMAT_FUNC,
                                                           **kwargs) → StringValue
```

Create inner WEBP file.

```
faker_file.providers.helpers.inner.create_inner_xlsx_file(storage: Optional[BaseStorage] = None,
                                                         basename: Optional[str] = None, prefix:
                                                         Optional[str] = None, generator:
                                                         Optional[Union[Faker, Generator,
                                                         Provider]] = None, data_columns:
                                                         Optional[Dict[str, str]] = None,
                                                         num_rows: int = 10, content:
                                                         Optional[str] = None, format_func:
                                                         Callable[[Union[Faker, Generator,
                                                         Provider], str], str] =
                                                         DEFAULT_FORMAT_FUNC, raw: bool
                                                         = True, **kwargs) → BytesValue
```

```
faker_file.providers.helpers.inner.create_inner_xlsx_file(storage: Optional[BaseStorage] = None,
                                                         basename: Optional[str] = None, prefix:
                                                         Optional[str] = None, generator:
                                                         Optional[Union[Faker, Generator,
                                                         Provider]] = None, data_columns:
                                                         Optional[Dict[str, str]] = None,
                                                         num_rows: int = 10, content:
                                                         Optional[str] = None, format_func:
                                                         Callable[[Union[Faker, Generator,
                                                         Provider], str], str] =
                                                         DEFAULT_FORMAT_FUNC,
                                                         **kwargs) → StringValue
```

Create inner XLSX file.

```
faker_file.providers.helpers.inner.create_inner_xml_file(storage: Optional[BaseStorage] = None,
                                                         basename: Optional[str] = None, prefix:
                                                         Optional[str] = None, generator:
                                                         Optional[Union[Faker, Generator,
                                                         Provider]] = None, root_element: str =
                                                         'root', row_element: str = 'row',
                                                         data_columns: Optional[Dict[str, str]] =
                                                         None, num_rows: int = 10, content:
                                                         Optional[str] = None, encoding:
                                                         Optional[str] = None, format_func:
                                                         Callable[[Union[Faker, Generator,
                                                         Provider], str], str] =
                                                         DEFAULT_FORMAT_FUNC, raw: bool
                                                         = True, **kwargs) → BytesValue
```

```
faker_file.providers.helpers.inner.create_inner_xml_file(storage: Optional[BaseStorage] = None,
                                                         basename: Optional[str] = None, prefix:
                                                         Optional[str] = None, generator:
                                                         Optional[Union[Faker, Generator,
                                                         Provider]] = None, root_element: str =
                                                         'root', row_element: str = 'row',
                                                         data_columns: Optional[Dict[str, str]] =
                                                         None, num_rows: int = 10, content:
                                                         Optional[str] = None, encoding:
                                                         Optional[str] = None, format_func:
                                                         Callable[[Union[Faker, Generator,
                                                         Provider], str], str] =
                                                         DEFAULT_FORMAT_FUNC, **kwargs)
                                                         → StringValue
```



Create inner XML file.

```
faker_file.providers.helpers.inner.create_inner_zip_file(storage: Optional[BaseStorage] = None,
                                                         basename: Optional[str] = None, prefix:
                                                         Optional[str] = None, generator:
                                                         Optional[Union[Faker, Generator,
                                                         Provider]] = None, options:
                                                         Optional[Dict[str, Any]] = None, raw:
                                                         bool = True, **kwargs) → BytesValue
```

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

Create inner ZIP file.

```
faker_file.providers.helpers.inner.fuzzy_choice_create_inner_file(func_choices:
                                                                    List[Tuple[Callable, Dict[str,
                                                                    Any]]], **kwargs) →
                                                                    Union[BytesValue,
                                                                    StringValue]
```

Create inner file from given list of function choices.

#### Parameters

**func\_choices** – List of functions to choose from.

#### Returns

StringValue.

Usage example:

```
from faker import Faker
from faker_file.providers.helpers.inner import (
    create_inner_docx_file, create_inner_epub_file, create_inner_txt_file,
    fuzzy_choice_create_inner_file,
)
from faker_file.storages.filesystem import FileSystemStorage

FAKER = Faker()
STORAGE = FileSystemStorage()

kwargs = {"storage": STORAGE, "generator": FAKER}
file = fuzzy_choice_create_inner_file(
    [
        (create_inner_docx_file, kwargs),
        (create_inner_epub_file, kwargs),
        (create_inner_txt_file, kwargs),
    ]
)
```

You could use it in archives to make a variety of different file types within the archive.

```
from faker import Faker
from faker_file.providers.helpers.inner import (
    create_inner_docx_file, create_inner_epub_file, create_inner_txt_file,
    fuzzy_choice_create_inner_file,
```

```
) from faker_file.providers.zip_file import ZipFileProvider from faker_file.storages.filesystem import
FileSystemStorage

FAKER = Faker() STORAGE = FileSystemStorage()

kwargs = {"storage": STORAGE, "generator": FAKER} file = ZipFileProvider(FAKER).zip_file(
    prefix="zzz_archive_", options={
        "count": 50, "create_inner_file_func": fuzzy_choice_create_inner_file, "create_inner_file_args": {
            "func_choices": [
                (create_inner_docx_file, kwargs), (create_inner_epub_file, kwargs), (create_inner_txt_file, kwargs),
            ],
        }, "directory": "zzz",
    }
)

faker_file.providers.helpers.inner.list_create_inner_file(func_list: List[Tuple[Callable, Dict[str, Any]]], **kwargs) →
List[Union[BytesValue, StringValue]]
```

Generates multiple files based on the provided list of functions and arguments.

#### Parameters

**func\_list** – List of tuples, each containing a function to generate a file and its arguments.

#### Returns

List of generated file names.

Usage example:

```
from faker import Faker from faker_file.providers.helpers.inner import (
    create_inner_docx_file, create_inner_xml_file, list_create_inner_file,
) from faker_file.providers.zip_file import ZipFileProvider from faker_file.storages.filesystem import
FileSystemStorage

FAKER = Faker() STORAGE = FileSystemStorage()

kwargs = {"storage": STORAGE, "generator": FAKER} file = ZipFileProvider(FAKER).zip_file(
    basename="alice-looking-through-the-glass", options={
        "create_inner_file_func": list_create_inner_file, "create_inner_file_args": {
            "func_list": [
                (create_inner_docx_file, {"basename": "doc"}), (create_inner_xml_file,
                {"basename": "doc_metadata"}), (create_inner_xml_file, {"basename":
                "doc_isbn"}),
            ],
        },
    }
)
```

Note, that while all other inner functions return back `Union[BytesValue, StringValue]` value, `list_create_inner_file` returns back a `List[Union[BytesValue, StringValue]]` value.

Notably, all inner functions were designed to support archives (such as ZIP, TAR and EML, but the list may grow in the future). If the inner function passed in the `create_inner_file_func` argument returns a List of `Union[BytesValue, StringValue]` values, the `option` argument is being ignored and generated files are simply limited to what has been passed in the `func_list` list of tuples.

#### 14.11.1.1.1.23 Module contents

#### 14.11.1.1.1.24 `faker_file.providers.mixins` package

#### 14.11.1.1.1.25 Submodules

#### 14.11.1.1.1.26 `faker_file.providers.mixins.image_mixin` module

```
class faker_file.providers.mixins.image_mixin.ImageMixin
    Bases: FileMixin
    Image mixin.
    extension: str
    formats: List[str]
    generator: Union[Faker, Generator, Provider]
    numerify: Callable
    random_element: Callable
```

#### 14.11.1.1.1.27 `faker_file.providers.mixins.tablular_data_mixin` module

```
class faker_file.providers.mixins.tablular_data_mixin.TabularDataMixin
    Bases: FileMixin
    Tabular data mixin.
    extension: str
    formats: List[str]
    generator: Union[Faker, Generator, Provider]
    numerify: Callable
    random_element: Callable
```

#### 14.11.1.1.1.28 Module contents

#### 14.11.1.1.1.29 `faker_file.providers.mp3_file` package

#### 14.11.1.1.1.30 Subpackages

#### 14.11.1.1.1.31 `faker_file.providers.mp3_file.generators` package

#### 14.11.1.1.1.32 Submodules

#### 14.11.1.1.1.33 `faker_file.providers.mp3_file.generators.edge_tts_generator` module

```
class faker_file.providers.mp3_file.generators.edge_tts_generator.EdgeTtsMp3Generator(content:
                                                                 str,
                                                                 gen-
                                                                 era-
                                                                 tor:
                                                                 Union[Faker,
                                                                 Gen-
                                                                 era-
                                                                 tor,
                                                                 Provider],
                                                                 **kwargs)
```

Bases: [\*BaseMp3Generator\*](#)

Edge Text-to-Speech generator.

Usage example:

```
from faker import Faker from faker_file.providers.mp3_file import Mp3FileProvider from
faker_file.providers.mp3_file.generators import edge_tts_generator

FAKER = Faker()

file = Mp3FileProvider(FAKER).mp3_file(
    mp3_generator_cls=edge_tts_generator.EdgeTtsMp3Generator
)

generate(**kwargs) → bytes
    Generate MP3.

handle_kwargs(**kwargs) → None
    Handle kwargs.

voice: str = 'en-GB-SoniaNeural'
```

#### 14.11.1.1.1.34 `faker_file.providers.mp3_file.generators.gtts_generator` module

```
class faker_file.providers.mp3_file.generators.gtts_generator.GttsMp3Generator(content: str,  
generator:  
Union[Faker,  
Generator,  
Provider],  
**kwargs)
```

Bases: `BaseMp3Generator`

Google Text-to-Speech generator.

Usage example:

```
from faker import Faker from faker_file.providers.mp3_file import Mp3FileProvider from  
faker_file.providers.mp3_file.generators.gtts_generator import (  
    GttsMp3Generator,  
)  
FAKER = Faker()  
file = Mp3FileProvider(FAKER).mp3_file(  
    mp3_generator_cls=GttsMp3Generator  
)  
generate(**kwargs) → bytes  
    Generate MP3.  
handle_kwargs(**kwargs) → None  
    Handle kwargs.  
lang: str = 'en'  
tld: str = 'com'
```

#### 14.11.1.1.1.35 Module contents

#### 14.11.1.1.1.36 Module contents

```
class faker_file.providers.mp3_file.Mp3FileProvider(generator: Any)
```

Bases: `BaseProvider`, `FileMixin`

MP3 file provider.

Usage example:

```
from faker import Faker from faker_file.providers.mp3_file import Mp3FileProvider  
FAKER = Faker() file = Mp3FileProvider(FAKER).mp3_file()
```

Usage example with options:

```
file = Mp3FileProvider(FAKER).mp3_file(  
    prefix="zzz", max_nb_chars=500,  
)
```

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

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

file = Mp3FileProvider(FAKER).mp3_file(
    storage=FileSystemStorage(
        root_path=settings.MEDIA_ROOT, rel_path="tmp",
    ), prefix="zzz", max_nb_chars=500,
)
```

Default MP3 generator class is *GttsMp3Generator* which uses Google Text-to-Speech services to generate an MP3 file from given or randomly generated text. It does not require additional services to run and the only dependency here is the *gtts* package. You can however implement your own custom MP3 generator class and pass it to the *mp3\_file* method in *mp3\_generator\_cls* argument instead of the default *GttsMp3Generator*.

Usage with custom MP3 generator class.

```
# Imaginary marytts Python library from marytts import MaryTTS
# Import BaseMp3Generator from faker_file.providers.base.mp3_generator import (
    BaseMp3Generator,
)
# Define custom MP3 generator class MaryTtsMp3Generator(BaseMp3Generator):
    locale: str = "cmu-rms-hsmm" voice: str = "en_US"

    def handle_kwargs(self, **kwargs) -> None:
        # Since it's impossible to unify all TTS systems it's allowed # to pass arbitrary arguments to the BaseMp3Generator # constructor. Each implementation class contains its own # additional tuning arguments. Check the source code of the # implemented MP3 generators as an example. if "locale" in kwargs:
            self.locale = kwargs["locale"]

        if "voice" in kwargs:
            self.voice = kwargs["voice"]

    def generate(self) -> bytes:
        # Your implementation here. Note, that self.content # in this context is the text to make MP3 from. # self.generator would be the Faker or Generator # instance from which you could extract information on # active locale. # What comes below is pseudo implementation. mary_tts = MaryTTS(locale=self.locale, voice=self.voice) return mary_tts.synth_mp3(self.content)

# Generate MP3 file from random text file = Mp3FileProvider(FAKER).mp3_file(
    mp3_generator_cls=MaryTtsMp3Generator,
)

extension: str = 'mp3'

mp3_file(storage: Optional[BaseStorage] = None, basename: Optional[str] = None, prefix: Optional[str] = None, max_nb_chars: int = DEFAULT_AUDIO_MAX_NB_CHARS, content: Optional[str] = None, mp3_generator_cls: Optional[Union[str, Type[BaseMp3Generator]]] = DEFAULT_MP3_GENERATOR, mp3_generator_kwargs: Optional[Dict[str, Any]] = None, format_func: Callable[[Union[Faker, Generator, Provider], str], str] = DEFAULT_FORMAT_FUNC, raw: bool = True, **kwargs) -> BytesValue
```

```
mp3_file(storage: Optional[BaseStorage] = None, basename: Optional[str] = None, prefix: Optional[str] =
None, max_nb_chars: int = DEFAULT_AUDIO_MAX_NB_CHARS, content: Optional[str] =
None, mp3_generator_cls: Optional[Union[str, Type[BaseMp3Generator]]] =
DEFAULT_MP3_GENERATOR, mp3_generator_kwargs: Optional[Dict[str, Any]] = None,
format_func: Callable[[Union[Faker, Generator, Provider], str], str] =
DEFAULT_FORMAT_FUNC, **kwargs) → StringValue
```

Generate a MP3 file with random text.

#### Parameters

- **storage** – Storage. Defaults to *FileSystemStorage*.
- **basename** – File basename (without extension).
- **prefix** – File name prefix.
- **max\_nb\_chars** – Max number of chars for the content.
- **content** – File content. Might contain dynamic elements, which are then replaced by correspondent fixtures.
- **mp3\_generator\_cls** – Mp3 generator class.
- **mp3\_generator\_kwargs** – Mp3 generator kwargs.
- **format\_func** – Callable responsible for formatting template strings.
- **raw** – If set to True, return *BytesValue* (binary content of the file). Otherwise, return *StringValue* (path to the saved file).

#### Returns

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

### 14.11.1.1.1.37 `faker_file.providers.pdf_file` package

### 14.11.1.1.1.38 Subpackages

### 14.11.1.1.1.39 `faker_file.providers.pdf_file.generators` package

### 14.11.1.1.1.40 Submodules

### 14.11.1.1.1.41 `faker_file.providers.pdf_file.generators.pdfkit_generator` module

```
class faker_file.providers.pdf_file.generators.pdfkit_generator.PdfkitPdfGenerator(**kwargs)
```

Bases: *BasePdfGenerator*

Pdfkit PDF generator.

Usage example:

```
from faker import Faker
from faker_file.providers.pdf_file import PdfFileProvider
from faker_file.providers.pdf_file.generators import pdfkit_generator

FAKER = Faker()

file = PdfFileProvider(FAKER).pdf_file(
    pdf_generator_cls=pdfkit_generator.PdfkitPdfGenerator
)
```

Using *DynamicTemplate*:

```
import base64 from faker import Faker from faker_file.base import DynamicTemplate from
faker_file.providers.jpeg_file import JpegFileProvider from faker_file.providers.pdf_file im-
port PdfFileProvider from faker_file.providers.pdf_file.generators import pdfkit_generator

FAKER = Faker()

def create_data_url(image_bytes, image_format):
    encoded_image = base64.b64encode(image_bytes).decode('utf-8')    return
    f"data:image/{image_format};base64,{encoded_image}"

# Add table function def pdf_add_table(
    provider, generator, document, data, counter, **kwargs

):
    rows = kwargs.get("rows", 3) cols = kwargs.get("cols", 4)
    # Begin the HTML table table_html = "<table>"

    for row_num in range(rows):
        table_html += "<tr>"

        for col_num in range(cols):
            text = provider.generator.paragraph() table_html += f"<td>{text}</td>"

            data.setdefault("content_modifiers", {}) data["content_modifiers"].setdefault("add_table",
            {}) data["content_modifiers"]["add_table"].setdefault(
                counter, []
            ) data["content_modifiers"]["add_table"][counter].append(
                text
            )

        table_html += "</tr>"

    # End the HTML table table_html += "</table>"
    document += ("

+ table_html)

# Add picture function def pdf_add_picture(
    provider, generator, document, data, counter, **kwargs

):
    jpeg_file = JpegFileProvider(provider.generator).jpeg_file(
        raw=True
    ) data_url = create_data_url(jpeg_file, "jpg") document += f"<img
src='{data_url}' alt='Inline Image' />" data.setdefault("content_modifiers",
    {}) data["content_modifiers"].setdefault("add_picture",
    {})
    data["content_modifiers"]["add_picture"].setdefault(counter,
    [])
    data["content_modifiers"]["add_picture"][counter].append(
        jpeg_file.data["content"]
    ) data["content"] += ("
```



```

“ + jpeg_file.data[“content”])
    # Add page break def pdf_add_page_break(
        provider, generator, document, data, counter, **kwargs

    ):
        page_break_html = “<div style=’page-break-before: always;’></div>” document += “

“ + page_break_html
    # Add a paragraph def pdf_add_paragraph(
        provider, generator, document, data, counter, **kwargs

    ):
        content      =      provider.generator.text(max_nb_chars=5_000)      paragraph_html      =
        f’<div><p>{content}</p></div>” document += “

“ + paragraph_html
    # Create a file with table, page-break, picture, page-break, paragraph file = Pdf-
    FileProvider(Faker()).pdf_file(
        pdf_generator_cls=pdfkit_generator.PdfkitPdfGenerator, content=DynamicTemplate(
            [
                (pdf_add_table, {}), (pdf_add_page_break, {}), (pdf_add_picture, {}),
                (pdf_add_page_break, {}), (pdf_add_paragraph, {}),
            ]
        )
    )
    # Create a file with text of 100 pages file = PdfFileProvider(Faker()).pdf_file(
        pdf_generator_cls=pdfkit_generator.PdfkitPdfGenerator, content=DynamicTemplate(
            [
                (pdf_add_paragraph, {}), (pdf_add_page_break, {}),
            ] * 100
        )
    )

encoding:   str = ‘utf-8’

generate(content: Union[str, DynamicTemplate], data: Dict[str, Any], provider: Union[Faker, Generator,
    Provider], **kwargs) → bytes
    Generate PDF.

handle_kwargs(**kwargs) → None
    Handle kwargs.

```

#### 14.11.1.1.1.42 `faker_file.providers.pdf_file.generators.reportlab_generator` module

**class** `faker_file.providers.pdf_file.generators.reportlab_generator.ReportlabPdfGenerator(**kwargs)`

Bases: *BasePdfGenerator*

Reportlab PDF generator.

Usage example:

```
from faker import Faker from faker_file.providers.pdf_file import PdfFileProvider from
faker_file.providers.pdf_file.generators import reportlab_generator

FAKER = Faker()

file = PdfFileProvider(FAKER).pdf_file(
    pdf_generator_cls=reportlab_generator.ReportlabPdfGenerator
)
```

Using *DynamicTemplate*:

```
from io import BytesIO

from faker import Faker from faker_file.base import DynamicTemplate from
faker_file.providers.jpeg_file import JpegFileProvider from faker_file.providers.pdf_file import
PdfFileProvider from faker_file.providers.pdf_file.generators import (
    reportlab_generator
) from PIL import Image as PilImage from reportlab.lib.pagesizes import letter, landscape
from reportlab.lib.styles import getSampleStyleSheet from reportlab.lib import colors from
reportlab.pdfbase import pdfmetrics from reportlab.pdfbase.ttfonts import TTFont from report-
lab.platypus import (
    Image, Table, TableStyle, PageBreak, Paragraph,
)

FAKER = Faker()

# Add table function def pdf_add_table(
    provider, generator, story, data, counter, **kwargs
):
    rows = kwargs.get("rows", 3) cols = kwargs.get("cols", 4)
    # Define your table headers headers = [f'Header {i+1}' for i in range(cols)]
    # Generate the rest of the table data table_data = [
        [
            provider.generator.word() for _ in range(cols)
        ] for _ in range(rows)
    ]
    # Add the headers to the table data table_data.insert(0, headers)
    # Create the table object table = Table(table_data)
    # Apply table styles table.setStyle(
        TableStyle(
```

```

        [
            ("BACKGROUND", (0, 0), (-1, 0), colors.grey), ("TEXTCOLOR", (0, 0), (-1, 0), colors.whitesmoke), ("ALIGN", (0, 0), (-1, -1), "CENTER"), ("FONT-NAME", (0, 0), (-1, 0), "Helvetica-Bold"), ("FONTSIZE", (0, 0), (-1, 0), 14), ("BOTTOMPADDING", (0, 0), (-1, 0), 12), ("BACKGROUND", (0, 1), (-1, -1), colors.beige), ("GRID", (0, 0), (-1, -1), 1, colors.black), ("BOX", (0, 0), (-1, -1), 1, colors.black),
        ]
    )
)

# Add the table to the document and build it story.append(table)

# Add picture function def pdf_add_picture(
    provider, generator, story, data, counter, **kwargs
):
    jpeg_file = JpegFileProvider(provider.generator).jpeg_file(
        raw=True
    )

    # Create a BytesIO object and load the image data with BytesIO(jpeg_file) as input_stream:
    pil_image = Pillow.open(input_stream)

    # Resize the image new_width = 400 new_height = 400 pil_image =
    pil_image.resize((new_width, new_height))

    # Create a BytesIO object outside the 'with' statement output_stream = BytesIO()
    pil_image.save(output_stream, format='JPEG') output_stream.seek(0) # Move to the
    start of the stream

    # Now you can use output_stream as your image data img = Image(output_stream)
    img.width = new_width img.height = new_height story.append(img)

# Add page break function def pdf_add_page_break(
    provider, generator, story, data, counter, **kwargs
):
    # Insert a page break story.append(PageBreak())

# Add paragraph function def pdf_add_paragraph(
    provider, generator, story, data, counter, **kwargs
):
    # Insert a paragraph styles = getSampleStyleSheet() style_paragraph = styles["Normal"]
    style_paragraph.fontName = generator.font_name pdfmetrics.registerFont(
        TTFont(generator.font_name, generator.font_path)
    ) content = provider.generator.text(max_nb_chars=5_000) paragraph = Paragraph(content,
    style_paragraph) story.append(paragraph)

```

```
# Create a file with table, page-break, picture, page-break, paragraph file = Pdf-
FileProvider(Faker()).pdf_file(
    pdf_generator_cls=(
        reportlab_generator.ReportlabPdfGenerator
    ), content=DynamicTemplate(
        [
            (pdf_add_table, {}), (pdf_add_page_break, {}), (pdf_add_picture, {}),
            (pdf_add_page_break, {}), (pdf_add_paragraph, {}),
        ]
    )
)

# Create a file with text of 100 pages file = PdfFileProvider(Faker()).pdf_file(
    pdf_generator_cls=(
        reportlab_generator.ReportlabPdfGenerator
    ), content=DynamicTemplate(
        [
            (pdf_add_paragraph, {}), (pdf_add_page_break, {}),
        ] * 100
    )
)

font_name: str = 'Vera'
font_path: str = 'Vera.ttf'

generate(content: Union[str, DynamicTemplate], data: Dict[str, Any], provider: Union[Faker, Generator,
Provider], **kwargs) → bytes
Generate PDF.

Parameters
    • content –
    • data –
    • provider – PdfFileProvider instance.

handle_kwargs(**kwargs) → None
Handle kwargs.
```

#### 14.11.1.1.1.43 Module contents

#### 14.11.1.1.1.44 Module contents

```
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,
)
```

Default PDF generator class is *PdfkitPdfGenerator* which uses *pdfkit* Python package and *wkhtmltopdf* system package for generating PDFs from randomly generated text. The quality of the produced PDFs is very good, but it's less performant than *ReportlabPdfGenerator* (factor 40x), which does not require additional system dependencies to run. To use it, pass *ReportlabPdfGenerator* class in *pdf\_generator\_cls* argument.

```
from faker_file.providers.pdf_file.generators import (
    reportlab_generator,
)

file = PdfFileProvider(None).pdf_file(
    max_nb_chars=1_000, wrap_chars_after=80, pdf_generator_cls=reportlab_generator.ReportlabPdfGenerator,
)
```

**extension:** `str = 'pdf'`

```
pdf_file(storage: Optional[BaseStorage] = None, basename: Optional[str] = None, prefix: Optional[str] =
    None, max_nb_chars: int = DEFAULT_TEXT_MAX_NB_CHARS, wrap_chars_after:
    Optional[int] = None, content: Optional[Union[str, DynamicTemplate]] = None,
    pdf_generator_cls: Optional[Union[str, Type[BasePdfGenerator]]] =
    DEFAULT_PDF_GENERATOR, pdf_generator_kwargs: Optional[Dict[str, Any]] = None,
    format_func: Callable[[Union[Faker, Generator, Provider], str], str] =
    DEFAULT_FORMAT_FUNC, raw: bool = True, **kwargs) → BytesValue
```

```
pdf_file(storage: Optional[BaseStorage] = None, basename: Optional[str] = None, prefix: Optional[str] =
    None, max_nb_chars: int = DEFAULT_TEXT_MAX_NB_CHARS, wrap_chars_after:
    Optional[int] = None, content: Optional[Union[str, DynamicTemplate]] = None,
    pdf_generator_cls: Optional[Union[str, Type[BasePdfGenerator]]] =
    DEFAULT_PDF_GENERATOR, pdf_generator_kwargs: Optional[Dict[str, Any]] = None,
    format_func: Callable[[Union[Faker, Generator, Provider], str], str] =
    DEFAULT_FORMAT_FUNC, **kwargs) → StringValue
```

Generate a PDF file with random text.

#### Parameters

- **storage** – Storage. Defaults to *FileSystemStorage*.

- **basename** – File basename (without extension).
- **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.
- **pdf\_generator\_cls** – PDF generator class.
- **pdf\_generator\_kwargs** – PDF generator kwargs.
- **format\_func** – Callable responsible for formatting template strings.
- **raw** – If set to True, return *BytesValue* (binary content of the file). Otherwise, return *StringValue* (path to the saved file).

#### Returns

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

### 14.11.1.1.1.45 Submodules

#### 14.11.1.1.1.46 `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*, *basename*: *Optional*[*str*] = *None*, *prefix*: *Optional*[*str*] = *None*, *length*: *int* = 1 \* 1024 \* 1024, *content*: *Optional*[*bytes*] = *None*, *raw*: *bool* = *True*, *\*\*kwargs*) → *BytesValue*

**bin\_file**(*storage*: *Optional*[[BaseStorage](#)] = *None*, *basename*: *Optional*[*str*] = *None*, *prefix*: *Optional*[*str*] = *None*, *length*: *int* = 1 \* 1024 \* 1024, *content*: *Optional*[*bytes*] = *None*, *\*\*kwargs*) → *StringValue*

Generate a BIN file with random bytes.

#### Parameters

- **storage** – Storage class. Defaults to *FileSystemStorage*.
- **basename** – File basename (without extension).
- **prefix** – File name prefix.
- **length** –
- **content** – File content. If given, used as is.
- **raw** – If set to *True*, return *BytesValue* (binary content of the file). Otherwise, return *StringValue* (path to the saved file).

#### Returns

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

**extension:** *str* = 'bin'

#### 14.11.1.1.47 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, basename: Optional[str] = None, prefix: Optional[str] =
None, header: Optional[Sequence[str]] = None, data_columns: Tuple[str, ...] = ('{{name}}',
'{{address}}'), num_rows: int = 10, include_row_ids: bool = False, content: Optional[str] = None,
encoding: Optional[str] = None, format_func: Callable[[Union[Faker, Generator, Provider], str],
str] = DEFAULT_FORMAT_FUNC, raw: bool = True, **kwargs) → BytesValue
```

```
csv_file(storage: Optional[BaseStorage] = None, basename: Optional[str] = None, prefix: Optional[str] =
None, header: Optional[Sequence[str]] = None, data_columns: Tuple[str, ...] = ('{{name}}',
'{{address}}'), num_rows: int = 10, include_row_ids: bool = False, content: Optional[str] = None,
encoding: Optional[str] = None, format_func: Callable[[Union[Faker, Generator, Provider], str],
str] = DEFAULT_FORMAT_FUNC, **kwargs) → StringValue
```

Generate a CSV file with random text.

#### Parameters

- **storage** – Storage. Defaults to *FileSystemStorage*.
- **basename** – File basename (without extension).
- **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 `parse()` 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.
- **encoding** – Encoding.
- **format\_func** – Callable responsible for formatting template strings.
- **raw** – If set to `True`, return *BytesValue* (binary content of the file). Otherwise, return *StringValue* (path to the saved file).

#### Returns

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

**extension:** `str = 'csv'`

#### 14.11.1.1.148 `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 FileSystem-
Storage

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,
)
```

Usage example with content modifiers:

```
from io import BytesIO from faker_file.base import DynamicTemplate from
faker_file.providers.jpeg_file import JpegFileProvider

def add_table(provider, document, data, counter, **kwargs):

    table = document.add_table(
        kwargs.get("rows", 3), kwargs.get("cols", 4),
    ) data.setdefault("content_modifiers", {}) data["content_modifiers"].setdefault("add_table",
    {}) data["content_modifiers"]["add_table"].setdefault(counter, [])

    for row in table.rows:

        for cell in row.cells:
            text = provider.generator.paragraph() cell.text = text
            data["content_modifiers"]["add_table"][counter].append(
                text
            ) data["content"] += ("
" + text)

def add_picture(provider, document, data, counter, **kwargs):

    jpeg_file = JpegFileProvider(provider.generator).jpeg_file(
        raw=True
    ) picture = document.add_picture(BytesIO(jpeg_file))

    data.setdefault("content_modifiers", {}) data["content_modifiers"].setdefault("add_picture",
    {}) data["content_modifiers"]["add_picture"].setdefault(counter, [])

    data["content_modifiers"]["add_picture"][counter].append(
        jpeg_file.data["content"]
    ) data["content"] += ("
" + jpeg_file.data["content"])

    file = DocxFileProvider(Faker()).docx_file(
        content=DynamicTemplate([(add_table, {}), (add_picture, {})])
    )
```

```
docx_file(storage: Optional[BaseStorage] = None, basename: Optional[str] = None, prefix: Optional[str] = None, max_nb_chars: int = DEFAULT_TEXT_MAX_NB_CHARS, wrap_chars_after: Optional[int] = None, content: Optional[Union[str, DynamicTemplate]] = None, format_func: Callable[[Union[Faker, Generator, Provider], str], str] = DEFAULT_FORMAT_FUNC, raw: bool = True, **kwargs) → BytesValue
```

```
docx_file(storage: Optional[BaseStorage] = None, basename: Optional[str] = None, prefix: Optional[str] = None, max_nb_chars: int = DEFAULT_TEXT_MAX_NB_CHARS, wrap_chars_after: Optional[int] = None, content: Optional[Union[str, DynamicTemplate]] = None, format_func: Callable[[Union[Faker, Generator, Provider], str], str] = DEFAULT_FORMAT_FUNC, **kwargs) → StringValue
```

Generate a DOCX file with random text.

#### Parameters

- **storage** – Storage. Defaults to *FileSystemStorage*.
- **basename** – File basename (without extension).
- **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 (still being a string), which are then replaced by correspondent fixtures. Can alternatively be a *DynamicTemplate* - list of content modifiers (callables to call after the document instance has been created). Each callable should accept the following arguments: provider, document, data, counter and **\*\*kwargs**.
- **format\_func** – Callable responsible for formatting template strings.
- **raw** – If set to True, return *BytesValue* (binary content of the file). Otherwise, return *StringValue* (path to the saved file).

#### Returns

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

**extension:** `str = 'docx'`

#### 14.11.1.1.1.49 `faker_file.providers.eml_file` module

```
class faker_file.providers.eml_file.EmlFileProvider(generator: Any)
```

Bases: `BaseProvider`, `FileMixin`

EML file provider.

Usage example:

```
from faker import Faker
from faker_file.providers.eml_file import EmlFileProvider

FAKER = Faker()

file = EmlFileProvider(FAKER).eml_file()
```

Usage example with attachments:

```
from faker_file.providers.helpers.inner import create_inner_docx_file
from faker_file.providers.eml_file import EmlFileProvider
```

```

file = EmlFileProvider(FAKER).eml_file(
    prefix="zzz_email_", options={
        "count": 5, "create_inner_file_func": create_inner_docx_file, "create_inner_file_args": {
            "prefix": "zzz_docx_file_", "max_nb_chars": 1_024,
        },
    },
)

```

Usage example of nested EMLs attachments:

```

from faker_file.providers.helpers.inner import create_inner_eml_file

file = EmlFileProvider(FAKER).eml_file(
    options={
        "create_inner_file_func": create_inner_eml_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 EML, check the `file.data["files"]`.

**eml\_file**(*storage*: *Optional*[[BaseStorage](#)] = *None*, *basename*: *Optional*[*str*] = *None*, *prefix*: *Optional*[*str*] = *None*, *options*: *Optional*[*Dict*[*str*, *Any*]] = *None*, *max\_nb\_chars*: *int* = *DEFAULT\_TEXT\_MAX\_NB\_CHARS*, *wrap\_chars\_after*: *Optional*[*int*] = *None*, *content*: *Optional*[*str*] = *None*, *subject*: *Optional*[*str*] = *None*, *format\_func*: *Callable*[[*Union*[*Faker*, *Generator*, *Provider*], *str*], *str*] = *DEFAULT\_FORMAT\_FUNC*, *raw*: *bool* = *True*, *\*\*kwargs*) → [BytesValue](#)

**eml\_file**(*storage*: *Optional*[[BaseStorage](#)] = *None*, *basename*: *Optional*[*str*] = *None*, *prefix*: *Optional*[*str*] = *None*, *options*: *Optional*[*Dict*[*str*, *Any*]] = *None*, *max\_nb\_chars*: *int* = *DEFAULT\_TEXT\_MAX\_NB\_CHARS*, *wrap\_chars\_after*: *Optional*[*int*] = *None*, *content*: *Optional*[*str*] = *None*, *subject*: *Optional*[*str*] = *None*, *format\_func*: *Callable*[[*Union*[*Faker*, *Generator*, *Provider*], *str*], *str*] = *DEFAULT\_FORMAT\_FUNC*, *\*\*kwargs*) → [StringValue](#)

Generate an EML file with random text.

#### Parameters

- **storage** – Storage. Defaults to *FileSystemStorage*.
- **basename** – File basename (without extension).
- **prefix** – File name prefix.
- **options** – Options (non-structured) for complex types, such as ZIP.
- **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.
- **subject** – Email subject. Might contain dynamic elements, which are then replaced by correspondent fixtures.
- **format\_func** – Callable responsible for formatting template strings.
- **raw** – If set to True, return *BytesValue* (binary content of the file). Otherwise, return *StringValue* (path to the saved file).

**Returns**

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

**extension:** `str = 'eml'`

#### 14.11.1.1.1.50 `faker_file.providers.epub_file` module

**class** `faker_file.providers.epub_file.EpubFileProvider(generator: Any)`

Bases: `BaseProvider`, `FileMixin`

EPUB file provider.

Usage example:

```
from faker import Faker
from faker_file.providers.epub_file import EpubFileProvider

file = EpubFileProvider(Faker()).epub_file()
```

Usage example with options:

```
file = EpubFileProvider(Faker()).epub_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 = EpubFileProvider(Faker()).epub_file(
    storage=FileSystemStorage(
        root_path=settings.MEDIA_ROOT, rel_path="tmp",
    ), prefix="zzz", max_nb_chars=100_000, wrap_chars_after=80,
)
```

**epub\_file**(*storage: Optional[BaseStorage] = None, basename: Optional[str] = None, prefix: Optional[str] = None, max\_nb\_chars: int = DEFAULT\_TEXT\_MAX\_NB\_CHARS, wrap\_chars\_after: Optional[int] = None, content: Optional[str] = None, title: Optional[str] = None, chapter\_title: Optional[str] = None, format\_func: Callable[[Union[Faker, Generator, Provider], str], str] = DEFAULT\_FORMAT\_FUNC, raw: bool = True, \*\*kwargs*) → *BytesValue*

**epub\_file**(*storage: Optional[BaseStorage] = None, basename: Optional[str] = None, prefix: Optional[str] = None, max\_nb\_chars: int = DEFAULT\_TEXT\_MAX\_NB\_CHARS, wrap\_chars\_after: Optional[int] = None, content: Optional[str] = None, title: Optional[str] = None, chapter\_title: Optional[str] = None, format\_func: Callable[[Union[Faker, Generator, Provider], str], str] = DEFAULT\_FORMAT\_FUNC, \*\*kwargs*) → *StringValue*

Generate a EPUB file with random text.

**Parameters**

- **storage** – Storage. Defaults to *FileSystemStorage*.
- **basename** – File basename (without extension).
- **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.
- **title** – E-book title. Might contain dynamic elements, which are then replaced by correspondent fixtures.
- **chapter\_title** – Chapter title. Might contain dynamic elements, which are then replaced by correspondent fixtures.
- **format\_func** – Callable responsible for formatting template strings.
- **raw** – If set to True, return *BytesValue* (binary content of the file). Otherwise, return *StringValue* (path to the saved file).

**Returns**

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

**extension:** `str = 'epub'`

#### 14.11.1.1.151 `faker_file.providers.file_from_path` module

`class faker_file.providers.file_from_path.FileFromPathProvider(generator: Any)`

Bases: `BaseProvider`, `FileMixin`

File from given path provider.

Usage example:

```
from faker_file.providers.file_from_path import (
    FileFromPathProvider,
)

file = FileFromPathProvider(None).file_from_path(
    path="/path/to/file.pdf",
)
```

Usage example with options:

```
from faker_file.providers.file_from_path import (
    FileFromPathProvider,
)

file = FileFromPathProvider(None).file_from_path(
    path="/path/to/file.pdf", prefix="zzz",
)
```

**extension:** `str = ''`

**file\_from\_path**(*path: str, storage: Optional[BaseStorage] = None, basename: Optional[str] = None, prefix: Optional[str] = None, raw: bool = True, \*\*kwargs*) → *BytesValue*

**file\_from\_path**(*path: str, storage: Optional[BaseStorage] = None, basename: Optional[str] = None, prefix: Optional[str] = None, \*\*kwargs*) → *StringValue*

File from given path.

#### Parameters

- **path** – Path to source file.
- **storage** – Storage. Defaults to *FileSystemStorage*.
- **basename** – File basename (without extension).
- **prefix** – File name prefix.
- **raw** – If set to True, return *BytesValue* (binary content of the file). Otherwise, return *StringValue* (path to the saved file).

#### Returns

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

### 14.11.1.1.1.52 `faker_file.providers.generic_file` module

**class** `faker_file.providers.generic_file.GenericFileProvider`(*generator: Any*)

Bases: `BaseProvider`, *FileMixin*

Generic file provider.

Usage example:

```
from faker import Faker
from faker_file.providers.generic_file import GenericFileProvider

file = GenericFileProvider(Faker()).generic_file(
    content='<html><body><p>{{text}}</p></body></html>', extension='html',
)
```

Usage example with options:

```
file = GenericFileProvider(Faker()).generic_file(
    content='<html><body><p>{{text}}</p></body></html>', extension='html', pre-
    fix='zzz',
)
```

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

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

file = GenericFileProvider(Faker()).generic_file(
    content='<html><body><p>{{text}}</p></body></html>', extension='html', base-
    name='index', storage=FileSystemStorage(
        root_path=settings.MEDIA_ROOT, rel_path='tmp',
    ),
)
```

Usage example with AWS S3 storage:

```

from faker_file.storages.aws_s3 import AWSS3Storage

file = GenericFileProvider(Faker()).generic_file(
    storage=AWSS3Storage(bucket_name="My-test-bucket"),
    tent="<html><body><p>{ {text} }</p></body></html>", extension="html",
)

extension: str = None

generic_file(content: Union[bytes, str], extension: str, storage: Optional[BaseStorage] = None,
            basename: Optional[str] = None, prefix: Optional[str] = None, format_func:
            Callable[[Union[Faker, Generator, Provider], str], str] = DEFAULT_FORMAT_FUNC, raw:
            bool = True, **kwargs) → BytesValue

generic_file(content: Union[bytes, str], extension: str, storage: Optional[BaseStorage] = None, basename:
            Optional[str] = None, prefix: Optional[str] = None, format_func: Callable[[Union[Faker,
            Generator, Provider], str], str] = DEFAULT_FORMAT_FUNC, **kwargs) → StringValue

Generate a generic file with given content.

```

#### Parameters

- **content** – File content. If given, used as is.
- **extension** – File extension.
- **storage** – Storage class. Defaults to *FileSystemStorage*.
- **basename** – File basename (without extension).
- **prefix** – File name prefix.
- **format\_func** – Callable responsible for formatting template strings.
- **raw** – If set to True, return *BytesValue* (binary content of the file). Otherwise, return *StringValue* (path to the saved file).

#### Returns

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

#### 14.11.1.1.1.53 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, basename: Optional[str] = None, prefix: Optional[str] =
    None, max_nb_chars: int = DEFAULT_IMAGE_MAX_NB_CHARS, wrap_chars_after:
    Optional[int] = None, content: Optional[str] = None, format_func: Callable[[Union[Faker,
    Generator, Provider], str], str] = DEFAULT_FORMAT_FUNC, raw: bool = True, **kwargs) →
    BytesValue

ico_file(storage: Optional[BaseStorage] = None, basename: Optional[str] = None, prefix: Optional[str] =
    None, max_nb_chars: int = DEFAULT_IMAGE_MAX_NB_CHARS, wrap_chars_after:
    Optional[int] = None, content: Optional[str] = None, format_func: Callable[[Union[Faker,
    Generator, Provider], str], str] = DEFAULT_FORMAT_FUNC, **kwargs) → StringValue
```

Generate an ICO file with random text.

#### Parameters

- **storage** – Storage. Defaults to *FileSystemStorage*.
- **basename** – File basename (without extension).
- **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.
- **format\_func** – Callable responsible for formatting template strings.
- **raw** – If set to True, return *BytesValue* (binary content of the file). Otherwise, return *StringValue* (path to the saved file).

#### Returns

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

#### 14.11.1.1.154 `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, basename: Optional[str] = None, prefix: Optional[str]
          = None, max_nb_chars: int = DEFAULT_IMAGE_MAX_NB_CHARS, wrap_chars_after:
          Optional[int] = None, content: Optional[str] = None, format_func: Callable[[Union[Faker,
          Generator, Provider], str], str] = DEFAULT_FORMAT_FUNC, raw: bool = True, **kwargs) →
          BytesValue
```

```
jpeg_file(storage: Optional[BaseStorage] = None, basename: Optional[str] = None, prefix: Optional[str]
          = None, max_nb_chars: int = DEFAULT_IMAGE_MAX_NB_CHARS, wrap_chars_after:
          Optional[int] = None, content: Optional[str] = None, format_func: Callable[[Union[Faker,
          Generator, Provider], str], str] = DEFAULT_FORMAT_FUNC, **kwargs) → StringValue
```

Generate a JPEG file with random text.

#### Parameters

- **storage** – Storage. Defaults to *FileSystemStorage*.
- **basename** – File basename (without extension).
- **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.
- **format\_func** – Callable responsible for formatting template strings.
- **raw** – If set to True, return *BytesValue* (binary content of the file). Otherwise, return *StringValue* (path to the saved file).

#### Returns

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

#### 14.11.1.1.1.55 `faker_file.providers.odp_file` module

**class** `faker_file.providers.odp_file.OdpFileProvider(generator: Any)`

Bases: `BaseProvider`, `FileMixin`

ODP file provider.

Usage example:

```
from faker import Faker
from faker_file.providers.odp_file import OdpFileProvider

FAKER = Faker()

file = OdpFileProvider(FAKER).odp_file()
```

Usage example with options:

```
file = OdpFileProvider(FAKER).odp_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 = OdpFileProvider(FAKER).odp_file(
    storage=FileSystemStorage(
        root_path=settings.MEDIA_ROOT, rel_path="tmp",
    ), prefix="zzz", max_nb_chars=100_000, wrap_chars_after=80,
)
```

**extension:** `str = 'odp'`

**odp\_file**(*storage: Optional[BaseStorage] = None, basename: Optional[str] = None, prefix: Optional[str] = None, max\_nb\_chars: int = DEFAULT\_TEXT\_MAX\_NB\_CHARS, wrap\_chars\_after: Optional[int] = None, content: Optional[str] = None, format\_func: Callable[[Union[Faker, Generator, Provider], str], str] = DEFAULT\_FORMAT\_FUNC, raw: bool = True, \*\*kwargs*) → `BytesValue`

**odp\_file**(*storage: Optional[BaseStorage] = None, basename: Optional[str] = None, prefix: Optional[str] = None, max\_nb\_chars: int = DEFAULT\_TEXT\_MAX\_NB\_CHARS, wrap\_chars\_after: Optional[int] = None, content: Optional[str] = None, format\_func: Callable[[Union[Faker, Generator, Provider], str], str] = DEFAULT\_FORMAT\_FUNC, \*\*kwargs*) → `StringValue`

Generate an ODP file with random text.

##### Parameters

- **storage** – Storage. Defaults to `FileSystemStorage`.
- **basename** – File basename (without extension).
- **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.

- **format\_func** – Callable responsible for formatting template strings.
- **raw** – If set to True, return *BytesValue* (binary content of the file). Otherwise, return *StringValue* (path to the saved file).

#### Returns

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

#### 14.11.1.1.56 faker\_file.providers.ods\_file module

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

Bases: `BaseProvider`, `TabularDataMixin`

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, basename: Optional[str] = None, prefix: Optional[str] = None, data\_columns: Optional[Dict[str, str]] = None, num\_rows: int = 10, content: Optional[str] = None, format\_func: Callable[[Union[Faker, Generator, Provider], str], str] = DEFAULT\_FORMAT\_FUNC, raw: bool = True, \*\*kwargs*) → *BytesValue*

**ods\_file**(*storage: Optional[BaseStorage] = None, basename: Optional[str] = None, prefix: Optional[str] = None, data\_columns: Optional[Dict[str, str]] = None, num\_rows: int = 10, content: Optional[str] = None, format\_func: Callable[[Union[Faker, Generator, Provider], str], str] = DEFAULT\_FORMAT\_FUNC, \*\*kwargs*) → *StringValue*

Generate an ODS file with random text.

**Parameters**

- **storage** – Storage. Defaults to *FileSystemStorage*.
- **basename** – File basename (without extension).
- **prefix** – File name prefix.
- **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.
- **content** – List of dicts with content (JSON-like format). If given, used as is.
- **format\_func** – Callable responsible for formatting template strings.
- **raw** – If set to `True`, return *BytesValue* (binary content of the file). Otherwise, return *StringValue* (path to the saved file).

**Returns**

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

**14.11.1.1.1.57 `faker_file.providers.odt_file` module**

**class** `faker_file.providers.odt_file.OdtFileProvider(generator: Any)`

Bases: `BaseProvider`, `FileMixin`

ODT file provider.

Usage example:

```
from faker import Faker
from faker_file.providers.odt_file import OdtFileProvider

FAKER = Faker()

file = OdtFileProvider(FAKER).odt_file()
```

Usage example with options:

```
file = OdtFileProvider(FAKER).odt_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 = OdtFileProvider(FAKER).odt_file(
    storage=FileSystemStorage(
        root_path=settings.MEDIA_ROOT, rel_path="tmp",
    ), prefix="zzz", max_nb_chars=100_000, wrap_chars_after=80,
)
```

Usage example with content modifiers:

```

from faker_file.base import DynamicTemplate from faker_file.providers.jpeg_file import
JpegFileProvider from odf.draw import Frame, Image from odf.style import (
    Style, TextProperties, TableColumnProperties, TableRowProperties, Table-
    CellProperties, GraphicProperties,
) from odf.table import Table, TableRow, TableCell, TableColumn from odf.text import
P

def add_table(provider, document, data, counter, **kwargs):
    table = Table() rows = kwargs.get("rows", 3) cols = kwargs.get("cols", 4)
    table_col_style = Style(name="TableColumn", family="table-column") ta-
    ble_col_style.addElement(
        TableColumnProperties(columnwidth="2cm")
    ) document.automaticstyles.addElement(table_col_style)

    table_row_style = Style(name="TableRow", family="table-row") ta-
    ble_row_style.addElement(TableRowProperties(rowheight="1cm")) docu-
    ment.automaticstyles.addElement(table_row_style)

    data.setdefault("content_modifiers", {}) data["content_modifiers"].setdefault("add_table",
    {}) data["content_modifiers"]["add_table"].setdefault(counter, [])

    table_cell_style = Style(name="TableCell", family="table-cell") ta-
    ble_cell_style.addElement(
        TableCellProperties(
            padding="0.1cm", border="0.05cm solid #000000"
        )
    ) document.automaticstyles.addElement(table_cell_style)

    # Create table table = Table() for i in range(rows):
        table.addElement(TableColumn(stylename=table_col_style))

    for row in range(cols):
        tr = TableRow(stylename=table_row_style) table.addElement(tr) for col in
        range(4):
            tc = TableCell(stylename=table_cell_style) tr.addElement(tc) text
            = provider.generator.paragraph() p = P(text=text) tc.addElement(p)
            data["content_modifiers"]["add_table"][counter].append(text)
            data["content"] += "

+ text

document.text.addElement(table)

def add_picture(
    provider, document, data, counter, width="10cm", height="5cm", **kwargs,
):
    paragraph = P() document.text.addElement(paragraph) jpeg_file = Jpeg-
    FileProvider(provider.generator).jpeg_file() image_data = jpeg_file.data["content"] im-
    age_frame = Frame(
        width=width, height=height, x="56pt", y="56pt", anchortype="paragraph",

```

```
)      href      =      document.addPicture(jpeg_file.data["filename"])      im-
age_frame.addElement(Image(href=href)) paragraph.addElement(image_frame)

data["content"] += "

+ jpeg_file.data["content"]

data.setdefault("content_modifiers", {}) data["content_modifiers"].setdefault("add_picture",
{}) data["content_modifiers"]["add_picture"].setdefault(counter, [])

data["content_modifiers"]["add_picture"][counter].append(
    jpeg_file.data["content"]
)

file = OdtFileProvider(FAKER).odt_file(
    content=DynamicTemplate([(add_table, {}), (add_picture, {})])
)

extension: str = 'odt'

odt_file(storage: Optional[BaseStorage] = None, basename: Optional[str] = None, prefix: Optional[str] =
None, max_nb_chars: int = DEFAULT_TEXT_MAX_NB_CHARS, wrap_chars_after:
Optional[int] = None, content: Optional[Union[str, DynamicTemplate]] = None, format_func:
Callable[[Union[Faker, Generator, Provider], str], str] = DEFAULT_FORMAT_FUNC, raw: bool
= True, **kwargs) → BytesValue

odt_file(storage: Optional[BaseStorage] = None, basename: Optional[str] = None, prefix: Optional[str] =
None, max_nb_chars: int = DEFAULT_TEXT_MAX_NB_CHARS, wrap_chars_after:
Optional[int] = None, content: Optional[Union[str, DynamicTemplate]] = None, format_func:
Callable[[Union[Faker, Generator, Provider], str], str] = DEFAULT_FORMAT_FUNC,
**kwargs) → StringValue
```

Generate an ODT file with random text.

#### Parameters

- **storage** – Storage. Defaults to *FileSystemStorage*.
- **basename** – File basename (without extension).
- **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.
- **format\_func** – Callable responsible for formatting template strings.
- **raw** – If set to True, return *BytesValue* (binary content of the file). Otherwise, return *StringValue* (path to the saved file).

#### Returns

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

#### 14.11.1.1.1.58 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, basename: Optional[str] = None, prefix: Optional[str] = None, max\_nb\_chars: int = DEFAULT\_IMAGE\_MAX\_NB\_CHARS, wrap\_chars\_after: Optional[int] = None, content: Optional[str] = None, format\_func: Callable[[Union[Faker, Generator, Provider], str], str] = DEFAULT\_FORMAT\_FUNC, raw: bool = True, \*\*kwargs*) → *BytesValue*

**png\_file**(*storage: Optional[BaseStorage] = None, basename: Optional[str] = None, prefix: Optional[str] = None, max\_nb\_chars: int = DEFAULT\_IMAGE\_MAX\_NB\_CHARS, wrap\_chars\_after: Optional[int] = None, content: Optional[str] = None, format\_func: Callable[[Union[Faker, Generator, Provider], str], str] = DEFAULT\_FORMAT\_FUNC, \*\*kwargs*) → *StringValue*

Generate a PNG file with random text.

##### Parameters

- **storage** – Storage. Defaults to *FileSystemStorage*.
- **basename** – File basename (without extension).
- **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.
- **format\_func** – Callable responsible for formatting template strings.

- **raw** – If set to True, return *BytesValue* (binary content of the file). Otherwise, return *StringValue* (path to the saved file).

**Returns**

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

**14.11.1.1.159 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, basename: Optional[str] = None, prefix: Optional[str] = None, max\_nb\_chars: int = DEFAULT\_TEXT\_MAX\_NB\_CHARS, wrap\_chars\_after: Optional[int] = None, content: Optional[str] = None, format\_func: Callable[[Union[Faker, Generator, Provider], str], str] = DEFAULT\_FORMAT\_FUNC, raw: bool = True, \*\*kwargs*) → *BytesValue*

**pptx\_file**(*storage: Optional[BaseStorage] = None, basename: Optional[str] = None, prefix: Optional[str] = None, max\_nb\_chars: int = DEFAULT\_TEXT\_MAX\_NB\_CHARS, wrap\_chars\_after: Optional[int] = None, content: Optional[str] = None, format\_func: Callable[[Union[Faker, Generator, Provider], str], str] = DEFAULT\_FORMAT\_FUNC, \*\*kwargs*) → *StringValue*

Generate a file with random text.

**Parameters**

- **storage** – Storage. Defaults to *FileSystemStorage*.
- **basename** – File basename (without extension).
- **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.
- **format\_func** – Callable responsible for formatting template strings.
- **raw** – If set to True, return *BytesValue* (binary content of the file). Otherwise, return *StringValue* (path to the saved file).

**Returns**

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

**14.11.1.1.60 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, storage: Optional[BaseStorage] = None, basename: Optional[str] = None, prefix: Optional[str] = None, raw: bool = True, \*\*kwargs*) → *BytesValue*

**random\_file\_from\_dir**(*source\_dir\_path: str, storage: Optional[BaseStorage] = None, basename: Optional[str] = None, prefix: Optional[str] = None, \*\*kwargs*) → *StringValue*

Pick a random file from given directory.

**Parameters**

- **source\_dir\_path** – Source files directory.
- **storage** – Storage. Defaults to *FileSystemStorage*.
- **basename** – File basename (without extension).
- **prefix** – File name prefix.

- **raw** – If set to True, return *BytesValue* (binary content of the file). Otherwise, return *StringValue* (path to the saved file).

**Returns**

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

**14.11.1.1.161 faker\_file.providers.rtf\_file module**

**class** `faker_file.providers.rtf_file.RtfFileProvider(generator: Any)`

Bases: `BaseProvider`, `FileMixin`

RTF file provider.

Usage example:

```
from faker_file.providers.rtf_file import RtfFileProvider
file = RtfFileProvider(None).rtf_file()
```

Usage example with options:

```
from faker_file.providers.rtf_file import RtfFileProvider
file = RtfFileProvider(None).rtf_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 = RtfFileProvider(Faker()).rtf_file(
    storage=FileSystemStorage(
        root_path=settings.MEDIA_ROOT, rel_path="tmp",
    ), prefix="zzz", max_nb_chars=100_000, wrap_chars_after=80,
)
```

**extension:** `str = 'rtf'`

**rtf\_file**(*storage: Optional[BaseStorage] = None, basename: Optional[str] = None, prefix: Optional[str] = None, max\_nb\_chars: int = DEFAULT\_TEXT\_MAX\_NB\_CHARS, wrap\_chars\_after: Optional[int] = None, content: Optional[str] = None, format\_func: Callable[[Union[Faker, Generator, Provider], str], str] = DEFAULT\_FORMAT\_FUNC, raw: bool = True, \*\*kwargs*) → *BytesValue*

**rtf\_file**(*storage: Optional[BaseStorage] = None, basename: Optional[str] = None, prefix: Optional[str] = None, max\_nb\_chars: int = DEFAULT\_TEXT\_MAX\_NB\_CHARS, wrap\_chars\_after: Optional[int] = None, content: Optional[str] = None, format\_func: Callable[[Union[Faker, Generator, Provider], str], str] = DEFAULT\_FORMAT\_FUNC, \*\*kwargs*) → *StringValue*

Generate a RTF file with random text.

**Parameters**

- **storage** – Storage. Defaults to *FileSystemStorage*.
- **basename** – File basename (without extension).
- **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.
- **format\_func** – Callable responsible for formatting template strings.
- **raw** – If set to True, return *BytesValue* (binary content of the file). Otherwise, return *StringValue* (path to the saved file).

#### Returns

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

#### 14.11.1.1.62 `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, basename: Optional[str] = None, prefix: Optional[str] = None, max\_nb\_chars: int = DEFAULT\_IMAGE\_MAX\_NB\_CHARS, wrap\_chars\_after: Optional[int] = None, content: Optional[str] = None, format\_func: Callable[[Union[Faker, Generator, Provider], str], str] = DEFAULT\_FORMAT\_FUNC, raw: bool = True, \*\*kwargs*) → *BytesValue*

**svg\_file**(*storage: Optional[BaseStorage] = None, basename: Optional[str] = None, prefix: Optional[str] = None, max\_nb\_chars: int = DEFAULT\_IMAGE\_MAX\_NB\_CHARS, wrap\_chars\_after: Optional[int] = None, content: Optional[str] = None, format\_func: Callable[[Union[Faker, Generator, Provider], str], str] = DEFAULT\_FORMAT\_FUNC, \*\*kwargs*) → *StringValue*

Generate an SVG file with random text.

#### Parameters

- **storage** – Storage. Defaults to *FileSystemStorage*.
- **basename** – File basename (without extension).
- **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.
- **format\_func** – Callable responsible for formatting template strings.
- **raw** – If set to True, return *BytesValue* (binary content of the file). Otherwise, return *StringValue* (path to the saved file).

#### Returns

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

#### 14.11.1.1.1.63 `faker_file.providers.tar_file` module

**class** `faker_file.providers.tar_file.TarFileProvider(generator: Any)`

Bases: `BaseProvider`, `FileMixin`

TAR file provider.

Usage example:

```
from faker import Faker
from faker_file.providers.tar_file import TarFileProvider

FAKER = Faker()

file = TarFileProvider(FAKER).tar_file()
```

Usage example with options:

```
from faker_file.providers.helpers.inner import create_inner_docx_file
from faker_file.providers.tar_file import TarFileProvider

file = TarFileProvider(FAKER).tar_file(
    prefix="ttt_archive_", options={
        "count": 5, "create_inner_file_func": create_inner_docx_file, "create_inner_file_args": {
            "prefix": "ttt_docx_file_", "max_nb_chars": 1_024,
        }, "directory": "ttt",
    },
)
```

Usage example of nested TARs:

```
from faker_file.providers.helpers.inner import create_inner_tar_file

file = TarFileProvider(FAKER).tar_file(
    options={
        "create_inner_file_func": create_inner_tar_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 TAR, check the `file.data["files"]`.

**extension:** `str = 'tar'`

**tar\_file**(*storage: Optional[BaseStorage] = None, basename: Optional[str] = None, prefix: Optional[str] = None, options: Optional[Dict[str, Any]] = None, compression: Optional[str] = None, raw: bool = True, \*\*kwargs*) → *BytesValue*

**tar\_file**(*storage: Optional[BaseStorage] = None, basename: Optional[str] = None, prefix: Optional[str] = None, options: Optional[Dict[str, Any]] = None, compression: Optional[str] = None, \*\*kwargs*) → *StringValue*

Generate a TAR file with random text.

#### Parameters

- **storage** – Storage. Defaults to *FileSystemStorage*.
- **basename** – File basename (without extension).
- **prefix** – File name prefix.
- **options** – Options (non-structured) for complex types, such as ZIP.
- **compression** – Desired compression. Can be *None* or *gz*, *bz2* or *xz*.
- **raw** – If set to *True*, return *BytesValue* (binary content of the file). Otherwise, return *StringValue* (path to the saved file).

#### Returns

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

#### 14.11.1.1.1.64 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 import Faker
from faker_file.providers.txt_file import TxtFileProvider

file = TxtFileProvider(Faker()).txt_file()

```

Usage example with options:

```

file = TxtFileProvider(Faker()).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, basename: Optional[str] = None, prefix: Optional[str] =
    None, max_nb_chars: int = DEFAULT_TEXT_MAX_NB_CHARS, wrap_chars_after:
    Optional[int] = None, content: Optional[str] = None, format_func: Callable[[Union[Faker,
    Generator, Provider], str], str] = DEFAULT_FORMAT_FUNC, raw: bool = True, **kwargs) →
    BytesValue

txt_file(storage: Optional[BaseStorage] = None, basename: Optional[str] = None, prefix: Optional[str] =
    None, max_nb_chars: int = DEFAULT_TEXT_MAX_NB_CHARS, wrap_chars_after:
    Optional[int] = None, content: Optional[str] = None, format_func: Callable[[Union[Faker,
    Generator, Provider], str], str] = DEFAULT_FORMAT_FUNC, **kwargs) → StringValue
```

Generate a TXT file with random text.

#### Parameters

- **storage** – Storage. Defaults to *FileSystemStorage*.
- **basename** – File basename (without extension).
- **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.
- **format\_func** – Callable responsible for formatting template strings.
- **raw** – If set to True, return *BytesValue* (binary content of the file). Otherwise, return *StringValue* (path to the saved file).

#### Returns

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

#### 14.11.1.1.1.65 `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, basename: Optional[str] = None, prefix: Optional[str] = None, max\_nb\_chars: int = DEFAULT\_IMAGE\_MAX\_NB\_CHARS, wrap\_chars\_after: Optional[int] = None, content: Optional[str] = None, format\_func: Callable[[Union[Faker, Generator, Provider], str], str] = DEFAULT\_FORMAT\_FUNC, raw: bool = True, \*\*kwargs*) → *BytesValue*

**webp\_file**(*storage: Optional[BaseStorage] = None, basename: Optional[str] = None, prefix: Optional[str] = None, max\_nb\_chars: int = DEFAULT\_IMAGE\_MAX\_NB\_CHARS, wrap\_chars\_after: Optional[int] = None, content: Optional[str] = None, format\_func: Callable[[Union[Faker, Generator, Provider], str], str] = DEFAULT\_FORMAT\_FUNC, \*\*kwargs*) → *StringValue*

Generate a WEBP file with random text.

#### Parameters

- **storage** – Storage. Defaults to *FileSystemStorage*.
- **basename** – File basename (without extension).
- **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.
- **format\_func** – Callable responsible for formatting template strings.
- **raw** – If set to True, return *BytesValue* (binary content of the file). Otherwise, return *StringValue* (path to the saved file).

#### Returns

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

#### 14.11.1.1.1.66 `faker_file.providers.xlsx_file` module

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

Bases: `BaseProvider`, `TabularDataMixin`

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, basename: Optional[str] = None, prefix: Optional[str] = None, data\_columns: Optional[Dict[str, str]] = None, num\_rows: int = 10, content: Optional[str] = None, format\_func: Callable[[Union[Faker, Generator, Provider], str], str] = DEFAULT\_FORMAT\_FUNC, raw: bool = True, \*\*kwargs*) → *BytesValue*

**xlsx\_file**(*storage: Optional[BaseStorage] = None, basename: Optional[str] = None, prefix: Optional[str] = None, data\_columns: Optional[Dict[str, str]] = None, num\_rows: int = 10, content: Optional[str] = None, format\_func: Callable[[Union[Faker, Generator, Provider], str], str] = DEFAULT\_FORMAT\_FUNC, \*\*kwargs*) → *StringValue*

Generate a XLSX file with random text.

##### Parameters

- **storage** – Storage. Defaults to *FileSystemStorage*.
- **basename** – File basename (without extension).
- **prefix** – File name prefix.



- **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.
- **content** – List of dicts with content (JSON-like format). If given, used as is.
- **format\_func** – Callable responsible for formatting template strings.
- **raw** – If set to `True`, return *BytesValue* (binary content of the file). Otherwise, return *StringValue* (path to the saved file).

**Returns**

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

**14.11.1.1.67 faker\_file.providers.xml\_file module**

**class** `faker_file.providers.xml_file.XmlFileProvider(generator: Any)`

Bases: `BaseProvider`, `FileMixin`

XML file provider.

Usage example:

```
from faker import Faker
from faker_file.providers.xml_file import XmlFileProvider

file = XmlFileProvider(Faker()).xml_file()
```

Usage example with options:

```
from faker_file.providers.xml_file import XmlFileProvider

file = XmlFileProvider(Faker()).xml_file(
    prefix="zzz", num_rows=100, data_columns={
        "name": "{{name}}", "sentence": "{{sentence}}", "address": "{{address}}",
    },
)
```

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

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

file = XmlFileProvider(Faker()).xml_file(
    storage=FileSystemStorage(
        root_path=settings.MEDIA_ROOT, rel_path="tmp",
    ), prefix="zzz", num_rows=100,
)
```

Usage example with template:

```
XML_TEMPLATE = "<books>
```

```
<book>
  <name>{{sentence}}</name>      <description>{{paragraph}}</description>
  <isbn>{{isbn13}}</isbn>

</book> <book>

  <name>{{sentence}}</name>      <description>{{paragraph}}</description>
  <isbn>{{isbn13}}</isbn>

</book> <book>

  <name>{{sentence}}</name>      <description>{{paragraph}}</description>
  <isbn>{{isbn13}}</isbn>

</book>

</books> ""
```

```
file = XmlFileProvider(Faker()).xml_file(content=XML_TEMPLATE)
```

**extension:** `str = 'xml'`

**xml\_file**(storage: *Optional*[[BaseStorage](#)] = None, basename: *Optional*[str] = None, prefix: *Optional*[str] = None, root\_element: str = 'root', row\_element: str = 'row', data\_columns: *Optional*[Dict[str, str]] = None, num\_rows: int = 10, content: *Optional*[str] = None, encoding: *Optional*[str] = None, format\_func: *Callable*[[*Union*[[Faker](#), [Generator](#), [Provider](#)], str], str] = [DEFAULT\\_FORMAT\\_FUNC](#), raw: bool = True, \*\*kwargs) → [BytesValue](#)

**xml\_file**(storage: *Optional*[[BaseStorage](#)] = None, basename: *Optional*[str] = None, prefix: *Optional*[str] = None, root\_element: str = 'root', row\_element: str = 'row', data\_columns: *Optional*[Dict[str, str]] = None, num\_rows: int = 10, content: *Optional*[str] = None, encoding: *Optional*[str] = None, format\_func: *Callable*[[*Union*[[Faker](#), [Generator](#), [Provider](#)], str], str] = [DEFAULT\\_FORMAT\\_FUNC](#), \*\*kwargs) → [StringValue](#)

Generate an XML file with random text.

#### Parameters

- **storage** – Storage. Defaults to *FileSystemStorage*.
- **basename** – File basename (without extension).
- **prefix** – File name prefix.
- **root\_element** – Root XML element.
- **row\_element** – Row XML element.
- **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.
- **content** – File content. Might contain dynamic elements, which are then replaced by correspondent fixtures.
- **encoding** – Encoding.
- **format\_func** – Callable responsible for formatting template strings.
- **raw** – If set to `True`, return *BytesValue* (binary content of the file). Otherwise, return *StringValue* (path to the saved file).

**Returns**

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

**14.11.1.1.1.68 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.helpers.inner import create_inner_docx_file
from faker_file.providers.zip_file import ZipFileProvider

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:

```
from faker_file.providers.helpers.inner import create_inner_zip_file

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, basename: Optional[str] = None, prefix: Optional[str] = None, options: Optional[Dict[str, Any]] = None, raw: bool = True, \*\*kwargs*) → *BytesValue*

**zip\_file**(*storage: Optional[BaseStorage] = None, basename: Optional[str] = None, prefix: Optional[str] = None, options: Optional[Dict[str, Any]] = None, \*\*kwargs*) → Union[BytesValue, StringValue]

Generate a ZIP file with random text.

#### Parameters

- **storage** – Storage. Defaults to *FileSystemStorage*.
- **basename** – File basename (without extension).
- **prefix** – File name prefix.
- **options** – Options (non-structured) for complex types, such as ZIP.
- **raw** – If set to True, return *BytesValue* (binary content of the file). Otherwise, return *StringValue* (path to the saved file).

#### Returns

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

### 14.11.1.1.1.69 Module contents

#### 14.11.1.1.2 faker\_file.storages package

##### 14.11.1.1.2.1 Submodules

##### 14.11.1.1.2.2 faker\_file.storages.aws\_s3 module

**class** `faker_file.storages.aws_s3.AWSS3Storage`(*bucket\_name: str, root\_path: Optional[str] = 'tmp', 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'`

#### 14.11.1.1.2.3 faker\_file.storages.azure\_cloud\_storage module

```
class faker_file.storages.azure_cloud_storage.AzureCloudStorage(bucket_name: str, root_path:
    Optional[str] = 'tmp', 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:** Optional[str] = 'azure'

#### 14.11.1.1.2.4 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**(*extension: str, prefix: Optional[str] = None, basename: Optional[str] = None*) → Any

Generate filename.

**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.

#### 14.11.1.1.2.5 faker\_file.storages.cloud module

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

Bases: [BaseStorage](#)

Base cloud storage.

```
abspath(filename: Pathy) → str
```

Return relative path.

```
abstract authenticate(*args, **kwargs)
```

```
bucket: Pathy
```

```
bucket_name: str
```

```
credentials: Dict[str, str]
```

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

Check if file exists.

```
generate_filename(extension: str, prefix: Optional[str] = None, basename: Optional[str] = None) →
Pathy
```

Generate filename.

```
relpath(filename: Pathy) → str
```

Return relative path.

```
schema: Optional[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, root_path: Optional[str]
= 'tmp', 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") 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'
```

#### 14.11.1.1.2.6 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

Check if file exists.

**generate\_filename**(extension: str, prefix: *Optional[str] = None*, basename: *Optional[str] = None*) → 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.

#### 14.11.1.1.2.7 faker\_file.storages.google\_cloud\_storage module

```
class faker_file.storages.google_cloud_storage.GoogleCloudStorage(bucket_name: str, root_path:
Optional[str] = 'tmp',
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:** Optional[str] = 'gs'

#### 14.11.1.1.2.8 Module contents

#### 14.11.1.1.3 faker\_file.tests package

##### 14.11.1.1.3.1 Submodules

##### 14.11.1.1.3.2 faker\_file.tests.data module

##### 14.11.1.1.3.3 faker\_file.tests.helpers module

##### 14.11.1.1.3.4 faker\_file.tests.test\_augment\_file\_from\_dir\_provider module

##### 14.11.1.1.3.5 faker\_file.tests.test\_base module

**class** `faker_file.tests.test_base.TestStringListTestCase`(*methodName='runTest'*)

Bases: TestCase

Test StringList test case.

**test\_string\_list**() → None

##### 14.11.1.1.3.6 faker\_file.tests.test\_cli module

**class** `faker_file.tests.test_cli.TestCLI`(*methodName='runTest'*)

Bases: TestCase

CLI tests.

**test\_broken\_imports**() → None

Test broken imports.

**test\_cli**

**test\_cli\_error\_no\_provider**() → None

Test CLI, no provider given.

**test\_cli\_generate\_completion**() → None

Test CLI, generate-completion.

**test\_cli\_version**() → None

Test CLI, version.



#### 14.11.1.1.3.7 `faker_file.tests.test_data_integrity` module

#### 14.11.1.1.3.8 `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`

#### 14.11.1.1.3.9 `faker_file.tests.test_providers` module

`class faker_file.tests.test_providers.ProvidersTestCase(methodName='runTest')`

Bases: `TestCase`

Providers test case.

`setUp()`

Hook method for setting up the test fixture before exercising it.

`test_broken_imports`

`test_faker`

`test_faker_retry_failures`

`test_load_class_from_non_existing_path()` → `None`

Test `load_class_from_path` invalid path.

`test_load_class_from_path_class_not_found()` → `None`

Test `load_class_from_path` class not found.

`test_load_class_from_path_no_class_type()` → `None`

Test `load_class_from_path` imported is not class.

`test_mp3_file_generate_not_implemented_exception()`

`test_pdf_file_generate_not_implemented_exception()`

`test_raw_standalone_providers`

`test_raw_standalone_providers_allow_failures`

`test_raw_standalone_providers_retry_failures`

`test_standalone_providers`

`test_standalone_providers_allow_failures`

`test_standalone_providers_retry_failures`

`test_standalone_tar_file`

`test_standalone_tar_file_allow_failures`

`test_standalone_zip_file`

`test_standalone_zip_file_allow_failures`

#### 14.11.1.1.3.10 `faker_file.tests.test_sqlalchemy_integration` module

#### 14.11.1.1.3.11 `faker_file.tests.test_storages` module

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

Bases: `TestCase`

Test storages.

`test_base_storage_exceptions`

`test_cloud_storage_exceptions`

`test_file_system_storage_abspath()` → `None`

Test *FileSystemStorage* *abspath*.

`test_pathy_file_system_storage_abspath()` → `None`

Test *PathyFileSystemStorage* *abspath*.

`test_storage`

`test_storage_generate_filename_exceptions`

`test_storage_initialization_exceptions`

#### 14.11.1.1.3.12 `faker_file.tests.texts` module

#### 14.11.1.1.3.13 Module contents

### 14.11.1.2 Submodules

#### 14.11.1.3 `faker_file.base` module

**class** `faker_file.base.BytesValue`

Bases: `bytes`

**data:** `Dict[str, Any] = {}`

`faker_file.base.DEFAULT_FORMAT_FUNC`(*generator: Union[Faker, Generator, Provider]*, *content: str*) → `str`

**class** `faker_file.base.DynamicTemplate`(*content\_modifiers: List[Tuple[Callable, Dict[str, Any]]]*)

Bases: `object`

Dynamic template.

**class** `faker_file.base.FileMixin`

Bases: `object`

File mixin.

**extension:** `str`

```

formats: List[str]
generator: Union[Faker, Generator, Provider]
numerify: Callable
random_element: Callable

```

```
class faker_file.base.StringList(strings: Optional[List[str]] = None, separator: str = ' ')
```

Bases: object

String list.

Usage example:

```
my_string = StringList(separator="
```

```
“)
```

```
my_string += “grape” my_string += “peaches” print(my_string)
```

```
add_string(value: str) → None
```

```
remove_string(value: str) → None
```

```
class faker_file.base.StringValue
```

Bases: str

```
data: Dict[str, Any] = {}
```

```
faker_file.base.parse_format_func(generator: Union[Faker, Generator, Provider], content: str) → str
```

```
faker_file.base.pystr_format_func(generator: Union[Faker, Generator, Provider], content: str) → str
```

```
faker_file.base.returns_list(func: Callable) → bool
```

Checks if callable returns a list of Union[BytesValue, StringValue].

Returns True if it's a List. Returns False otherwise.

#### 14.11.1.4 faker\_file.cli module

#### 14.11.1.5 faker\_file.constants module

#### 14.11.1.6 faker\_file.helpers module

```
faker_file.helpers.load_class_from_path(full_path: str) → Type
```

Load a class from a given full path string identifier.

##### Parameters

**full\_path** – The full path to the class, e.g. ‘module.submodule.MyClass’.

##### Returns

The loaded class.

##### Raise

If the module cannot be found or the class does not exist in the module, it raises ImportError.

Usage example:

```
my_class = load_class_from_path(“module.submodule.MyClass”) instance = my_class()
```

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

#### 14.11.1.7 Module contents

### 14.12 Indices and tables

- [genindex](#)
- [modindex](#)
- [search](#)

## PYTHON MODULE INDEX

### f

- faker\_file, 156
- faker\_file.base, 154
- faker\_file.cli, 155
- faker\_file.constants, 155
- faker\_file.helpers, 155
- faker\_file.providers, 148
- faker\_file.providers.augment\_file\_from\_dir.augmenters, 90
- faker\_file.providers.base, 92
- faker\_file.providers.base.mp3\_generator, 90
- faker\_file.providers.base.pdf\_generator, 91
- faker\_file.providers.base.text\_augmenter, 91
- faker\_file.providers.base.text\_extractor, 91
- faker\_file.providers.bin\_file, 118
- faker\_file.providers.csv\_file, 119
- faker\_file.providers.docx\_file, 120
- faker\_file.providers.eml\_file, 122
- faker\_file.providers.epub\_file, 124
- faker\_file.providers.file\_from\_path, 125
- faker\_file.providers.generic\_file, 126
- faker\_file.providers.helpers, 107
- faker\_file.providers.helpers.inner, 92
- faker\_file.providers.ico\_file, 127
- faker\_file.providers.jpeg\_file, 128
- faker\_file.providers.mixins, 108
- faker\_file.providers.mixins.image\_mixin, 107
- faker\_file.providers.mixins.tablular\_data\_mixin, 107
- faker\_file.providers.mp3\_file, 109
- faker\_file.providers.mp3\_file.generators, 109
- faker\_file.providers.mp3\_file.generators.edge\_tts\_generator, 108
- faker\_file.providers.mp3\_file.generators.gtts\_generator, 109
- faker\_file.providers.odp\_file, 130
- faker\_file.providers.ods\_file, 131
- faker\_file.providers.odt\_file, 132
- faker\_file.providers.pdf\_file, 116
- faker\_file.providers.pdf\_file.generators, 116
- faker\_file.providers.pdf\_file.generators.pdfkit\_generator, 111
- faker\_file.providers.pdf\_file.generators.reportlab\_generator, 114
- faker\_file.providers.png\_file, 135
- faker\_file.providers.pptx\_file, 136
- faker\_file.providers.random\_file\_from\_dir, 137
- faker\_file.providers.rtf\_file, 138
- faker\_file.providers.svg\_file, 139
- faker\_file.providers.tar\_file, 140
- faker\_file.providers.txt\_file, 141
- faker\_file.providers.webp\_file, 142
- faker\_file.providers.xlsx\_file, 144
- faker\_file.providers.xml\_file, 145
- faker\_file.providers.zip\_file, 147
- faker\_file.storages, 152
- faker\_file.storages.aws\_s3, 148
- faker\_file.storages.azure\_cloud\_storage, 149
- faker\_file.storages.base, 149
- faker\_file.storages.cloud, 150
- faker\_file.storages.filesystem, 151
- faker\_file.storages.google\_cloud\_storage, 151
- faker\_file.tests, 154
- faker\_file.tests.data, 152
- faker\_file.tests.test\_base, 152
- faker\_file.tests.test\_cli, 152
- faker\_file.tests.test\_django\_integration, 153
- faker\_file.tests.test\_providers, 153
- faker\_file.tests.test\_storages, 154
- faker\_file.tests.texts, 154



## INDEX

### A

`abspath()` (*faker\_file.storages.base.BaseStorage* method), 149

`abspath()` (*faker\_file.storages.cloud.CloudStorage* method), 150

`abspath()` (*faker\_file.storages.filesystem.FileSystemStorage* method), 151

`add_string()` (*faker\_file.base.StringList* method), 155

`augment()` (*faker\_file.providers.base.text\_augmenter.BaseTextAugmenter* method), 91

`authenticate()` (*faker\_file.storages.aws\_s3.AWSS3Storage* method), 148

`authenticate()` (*faker\_file.storages.azure\_cloud\_storage.AzureCloudStorage* method), 149

`authenticate()` (*faker\_file.storages.cloud.CloudStorage* method), 150

`authenticate()` (*faker\_file.storages.cloud.PathyFileSystemStorage* method), 150

`authenticate()` (*faker\_file.storages.google\_cloud\_storage.GoogleCloudStorage* method), 151

*AWSS3Storage* (class in *faker\_file.storages.aws\_s3*), 148

*AzureCloudStorage* (class in *faker\_file.storages.azure\_cloud\_storage*), 149

### B

*BaseMp3Generator* (class in *faker\_file.providers.base.mp3\_generator*), 90

*BasePdfGenerator* (class in *faker\_file.providers.base.pdf\_generator*), 91

*BaseStorage* (class in *faker\_file.storages.base*), 149

*BaseTextAugmenter* (class in *faker\_file.providers.base.text\_augmenter*), 91

*BaseTextExtractor* (class in *faker\_file.providers.base.text\_extractor*), 91

`bin_file()` (*faker\_file.providers.bin\_file.BinFileProvider* method), 119

*BinFileProvider* (class in *faker\_file.providers.bin\_file*), 118

`bucket` (*faker\_file.storages.azure\_cloud\_storage.AzureCloudStorage* attribute), 149

`bucket` (*faker\_file.storages.cloud.CloudStorage* attribute), 150

`bucket` (*faker\_file.storages.google\_cloud\_storage.GoogleCloudStorage* attribute), 152

`bucket_name` (*faker\_file.storages.azure\_cloud\_storage.AzureCloudStorage* attribute), 149

`bucket_name` (*faker\_file.storages.cloud.CloudStorage* attribute), 150

`bucket_name` (*faker\_file.storages.google\_cloud\_storage.GoogleCloudStorage* attribute), 152

*BytesValue* (class in *faker\_file.base*), 154

### C

*CloudStorage* (class in *faker\_file.storages.cloud*), 150

`content` (*faker\_file.providers.base.mp3\_generator.BaseMp3Generator* attribute), 90

`create_inner_bin_file()` (in module *faker\_file.providers.helpers.inner*), 92

`create_inner_csv_file()` (in module *faker\_file.providers.helpers.inner*), 92

`create_inner_docx_file()` (in module *faker\_file.providers.helpers.inner*), 93

`create_inner eml_file()` (in module *faker\_file.providers.helpers.inner*), 93

`create_inner_epub_file()` (in module *faker\_file.providers.helpers.inner*), 94

`create_inner_file_from_path()` (in module *faker\_file.providers.helpers.inner*), 94

`create_inner_generic_file()` (in module *faker\_file.providers.helpers.inner*), 95

`create_inner_ico_file()` (in module *faker\_file.providers.helpers.inner*), 95

`create_inner_jpeg_file()` (in module *faker\_file.providers.helpers.inner*), 96

`create_inner_mp3_file()` (in module *faker\_file.providers.helpers.inner*), 96

`create_inner_odp_file()` (in module *faker\_file.providers.helpers.inner*), 97

`create_inner_ods_file()` (in module *faker\_file.providers.helpers.inner*), 97

*faker\_file.providers.helpers.inner*), 98  
 create\_inner\_odt\_file() (in module *faker\_file.providers.helpers.inner*), 98  
 create\_inner\_pdf\_file() (in module *faker\_file.providers.helpers.inner*), 99  
 create\_inner\_png\_file() (in module *faker\_file.providers.helpers.inner*), 99  
 create\_inner\_pptx\_file() (in module *faker\_file.providers.helpers.inner*), 100  
 create\_inner\_random\_file\_from\_dir() (in module *faker\_file.providers.helpers.inner*), 100  
 create\_inner\_rtf\_file() (in module *faker\_file.providers.helpers.inner*), 101  
 create\_inner\_svg\_file() (in module *faker\_file.providers.helpers.inner*), 101  
 create\_inner\_tar\_file() (in module *faker\_file.providers.helpers.inner*), 102  
 create\_inner\_txt\_file() (in module *faker\_file.providers.helpers.inner*), 102  
 create\_inner\_webp\_file() (in module *faker\_file.providers.helpers.inner*), 103  
 create\_inner\_xlsx\_file() (in module *faker\_file.providers.helpers.inner*), 103  
 create\_inner\_xml\_file() (in module *faker\_file.providers.helpers.inner*), 104  
 create\_inner\_zip\_file() (in module *faker\_file.providers.helpers.inner*), 105  
 credentials (*faker\_file.storages.azure\_cloud\_storage.AzureCloudStorage* attribute), 149  
 credentials (*faker\_file.storages.cloud.CloudStorage* attribute), 150  
 credentials (*faker\_file.storages.google\_cloud\_storage.GoogleCloudStorage* attribute), 152  
 csv\_file() (*faker\_file.providers.csv\_file.CsvFileProvider* method), 119  
 CsvFileProvider (class in *faker\_file.providers.csv\_file*), 119

## D

data (*faker\_file.base.BytesValue* attribute), 154  
 data (*faker\_file.base.StringValue* attribute), 155  
 DEFAULT\_FORMAT\_FUNC() (in module *faker\_file.base*), 154  
 DjangoIntegrationTestCase (class in *faker\_file.tests.test\_django\_integration*), 153  
 docx\_file() (*faker\_file.providers.docx\_file.DocxFileProvider* method), 121  
 DocxFileProvider (class in *faker\_file.providers.docx\_file*), 120  
 DynamicTemplate (class in *faker\_file.base*), 154

## E

EdgeTtsMp3Generator (class in

*faker\_file.providers.mp3\_file.generators.edge\_tts\_generator*), 108  
 eml\_file() (*faker\_file.providers.eml\_file.EmlFileProvider* method), 123  
 EmlFileProvider (class in *faker\_file.providers.eml\_file*), 122  
 encoding (*faker\_file.providers.pdf\_file.generators.pdfkit\_generator.PdfkitGenerator* attribute), 113  
 epub\_file() (*faker\_file.providers.epub\_file.EpubFileProvider* method), 124  
 EpubFileProvider (class in *faker\_file.providers.epub\_file*), 124  
 exists() (*faker\_file.storages.base.BaseStorage* method), 149  
 exists() (*faker\_file.storages.cloud.CloudStorage* method), 150  
 exists() (*faker\_file.storages.filesystem.FileSystemStorage* method), 151  
 extension (*faker\_file.base.FileMixin* attribute), 154  
 extension (*faker\_file.providers.bin\_file.BinFileProvider* attribute), 119  
 extension (*faker\_file.providers.csv\_file.CsvFileProvider* attribute), 120  
 extension (*faker\_file.providers.docx\_file.DocxFileProvider* attribute), 122  
 extension (*faker\_file.providers.eml\_file.EmlFileProvider* attribute), 124  
 extension (*faker\_file.providers.epub\_file.EpubFileProvider* attribute), 125  
 extension (*faker\_file.providers.file\_from\_path.FileFromPathProvider* attribute), 125  
 extension (*faker\_file.providers.generic\_file.GenericFileProvider* attribute), 127  
 extension (*faker\_file.providers.ico\_file.IcoFileProvider* attribute), 128  
 extension (*faker\_file.providers.jpeg\_file.JpegFileProvider* attribute), 129  
 extension (*faker\_file.providers.mixins.image\_mixin.ImageMixin* attribute), 107  
 extension (*faker\_file.providers.mixins.tablular\_data\_mixin.TabularDataMixin* attribute), 107  
 extension (*faker\_file.providers.mp3\_file.Mp3FileProvider* attribute), 110  
 extension (*faker\_file.providers.odp\_file.OdpFileProvider* attribute), 130  
 extension (*faker\_file.providers.ods\_file.OdsFileProvider* attribute), 131  
 extension (*faker\_file.providers.odt\_file.OdtFileProvider* attribute), 134  
 extension (*faker\_file.providers.pdf\_file.PdfFileProvider* attribute), 117  
 extension (*faker\_file.providers.png\_file.PngFileProvider* attribute), 135  
 extension (*faker\_file.providers.pptx\_file.PptxFileProvider*



*attribute*), 136  
 extension (*faker\_file.providers.random\_file\_from\_dir.RandomFileFromDirProvider*  
*attribute*), 137  
 extension (*faker\_file.providers.rtf\_file.RtfFileProvider*  
*attribute*), 138  
 extension (*faker\_file.providers.svg\_file.SvgFileProvider*  
*attribute*), 139  
 extension (*faker\_file.providers.tar\_file.TarFileProvider*  
*attribute*), 141  
 extension (*faker\_file.providers.txt\_file.TxtFileProvider*  
*attribute*), 142  
 extension (*faker\_file.providers.webp\_file.WebpFileProvider*  
*attribute*), 143  
 extension (*faker\_file.providers.xlsx\_file.XlsxFileProvider*  
*attribute*), 144  
 extension (*faker\_file.providers.xml\_file.XmlFileProvider*  
*attribute*), 146  
 extension (*faker\_file.providers.zip\_file.ZipFileProvider*  
*attribute*), 147  
 extract() (*faker\_file.providers.base.text\_extractor.BaseTextExtractor*  
*method*), 91

## F

FAKER (*faker\_file.tests.test\_django\_integration.DjangoIntegrationTestCase*  
*attribute*), 153  
 faker\_file  
   module, 156  
 faker\_file.base  
   module, 154  
 faker\_file.cli  
   module, 155  
 faker\_file.constants  
   module, 155  
 faker\_file.helpers  
   module, 155  
 faker\_file.providers  
   module, 148  
 faker\_file.providers.augment\_file\_from\_dir.augmenters  
   module, 90  
 faker\_file.providers.base  
   module, 92  
 faker\_file.providers.base.mp3\_generator  
   module, 90  
 faker\_file.providers.base.pdf\_generator  
   module, 91  
 faker\_file.providers.base.text\_augmenter  
   module, 91  
 faker\_file.providers.base.text\_extractor  
   module, 91  
 faker\_file.providers.bin\_file  
   module, 118  
 faker\_file.providers.csv\_file  
   module, 119  
 faker\_file.providers.docx\_file  
   module, 120  
 faker\_file.providers.eml\_file  
   module, 122  
 faker\_file.providers.epub\_file  
   module, 124  
 faker\_file.providers.file\_from\_path  
   module, 125  
 faker\_file.providers.generic\_file  
   module, 126  
 faker\_file.providers.helpers  
   module, 107  
 faker\_file.providers.helpers.inner  
   module, 92  
 faker\_file.providers.ico\_file  
   module, 127  
 faker\_file.providers.jpeg\_file  
   module, 128  
 faker\_file.providers.mixins  
   module, 108  
 faker\_file.providers.mixins.image\_mixin  
   module, 107  
 faker\_file.providers.mixins.tablular\_data\_mixin  
   module, 107  
 faker\_file.providers.mp3\_file  
   module, 109  
 faker\_file.providers.mp3\_file.generators  
   module, 109  
 faker\_file.providers.mp3\_file.generators.edge\_tts\_generator  
   module, 108  
 faker\_file.providers.mp3\_file.generators.gtts\_generator  
   module, 109  
 faker\_file.providers.odp\_file  
   module, 130  
 faker\_file.providers.ods\_file  
   module, 131  
 faker\_file.providers.odt\_file  
   module, 132  
 faker\_file.providers.pdf\_file  
   module, 116  
 faker\_file.providers.pdf\_file.generators  
   module, 116  
 faker\_file.providers.pdf\_file.generators.pdftkit\_generator  
   module, 111  
 faker\_file.providers.pdf\_file.generators.reportlab\_generator  
   module, 114  
 faker\_file.providers.png\_file  
   module, 135  
 faker\_file.providers.pptx\_file  
   module, 136  
 faker\_file.providers.random\_file\_from\_dir  
   module, 137  
 faker\_file.providers.rtf\_file  
   module, 138  
 faker\_file.providers.svg\_file

module, 139  
 faker\_file.providers.tar\_file  
     module, 140  
 faker\_file.providers.txt\_file  
     module, 141  
 faker\_file.providers.webp\_file  
     module, 142  
 faker\_file.providers.xlsx\_file  
     module, 144  
 faker\_file.providers.xml\_file  
     module, 145  
 faker\_file.providers.zip\_file  
     module, 147  
 faker\_file.storages  
     module, 152  
 faker\_file.storages.aws\_s3  
     module, 148  
 faker\_file.storages.azure\_cloud\_storage  
     module, 149  
 faker\_file.storages.base  
     module, 149  
 faker\_file.storages.cloud  
     module, 150  
 faker\_file.storages.filesystem  
     module, 151  
 faker\_file.storages.google\_cloud\_storage  
     module, 151  
 faker\_file.tests  
     module, 154  
 faker\_file.tests.data  
     module, 152  
 faker\_file.tests.test\_base  
     module, 152  
 faker\_file.tests.test\_cli  
     module, 152  
 faker\_file.tests.test\_django\_integration  
     module, 153  
 faker\_file.tests.test\_providers  
     module, 153  
 faker\_file.tests.test\_storages  
     module, 154  
 faker\_file.tests.texts  
     module, 154  
 file\_from\_path() (faker\_file.providers.file\_from\_path.FileFromPathProvider  
     method), 126  
 FileFromPathProvider (class in  
     faker\_file.providers.file\_from\_path), 125  
 FileMixin (class in faker\_file.base), 154  
 FileSystemStorage (class in  
     faker\_file.storages.filesystem), 151  
 font\_name (faker\_file.providers.pdf\_file.generators.reportlab\_generator.ReportlabPdfGenerator  
     attribute), 116  
 font\_path (faker\_file.providers.pdf\_file.generators.reportlab\_generator.ReportlabPdfGenerator  
     attribute), 116

formats (faker\_file.base.FileMixin attribute), 154  
 formats (faker\_file.providers.mixins.image\_mixin.ImageMixin  
     attribute), 107  
 formats (faker\_file.providers.mixins.tablular\_data\_mixin.TabularDataMixin  
     attribute), 107  
 fuzzy\_choice\_create\_inner\_file() (in module  
     faker\_file.providers.helpers.inner), 105

## G

generate() (faker\_file.providers.base.mp3\_generator.BaseMp3Generator  
     method), 90  
 generate() (faker\_file.providers.base.pdf\_generator.BasePdfGenerator  
     method), 91  
 generate() (faker\_file.providers.mp3\_file.generators.edge\_tts\_generator.EdgeTtsGenerator  
     method), 108  
 generate() (faker\_file.providers.mp3\_file.generators.gtts\_generator.GttsMp3Generator  
     method), 109  
 generate() (faker\_file.providers.pdf\_file.generators.pdfkit\_generator.PdfkitGenerator  
     method), 113  
 generate() (faker\_file.providers.pdf\_file.generators.reportlab\_generator.ReportlabPdfGenerator  
     method), 116  
 generate\_filename()  
     (faker\_file.storages.base.BaseStorage method),  
     149  
 generate\_filename()  
     (faker\_file.storages.cloud.CloudStorage  
     method), 150  
 generate\_filename()  
     (faker\_file.storages.filesystem.FileSystemStorage  
     method), 151  
 generator (faker\_file.base.FileMixin attribute), 155  
 generator (faker\_file.providers.base.mp3\_generator.BaseMp3Generator  
     attribute), 91  
 generator (faker\_file.providers.mixins.image\_mixin.ImageMixin  
     attribute), 107  
 generator (faker\_file.providers.mixins.tablular\_data\_mixin.TabularDataMixin  
     attribute), 107  
 generic\_file() (faker\_file.providers.generic\_file.GenericFileProvider  
     method), 127  
 GenericFileProvider (class in  
     faker\_file.providers.generic\_file), 126  
 GoogleCloudStorage (class in  
     faker\_file.storages.google\_cloud\_storage),  
     151  
 GttsMp3Generator (class in  
     faker\_file.providers.mp3\_file.generators.gtts\_generator),  
     109

## H

handle\_kwargs() (faker\_file.providers.base.mp3\_generator.BaseMp3Generator  
     method), 90  
 handle\_kwargs() (faker\_file.providers.base.pdf\_generator.BasePdfGenerator  
     method), 91

[handle\\_kwargs\(\)](#) (*faker\_file.providers.base.text\_augmenter.BaseTextAugmenter*, *method*), 91  
[handle\\_kwargs\(\)](#) (*faker\_file.providers.base.text\_extractor.BaseTextExtractor*, *method*), 91  
[handle\\_kwargs\(\)](#) (*faker\_file.providers.mp3\_file.generators.edge\_tts\_generator*, *method*), 108  
[handle\\_kwargs\(\)](#) (*faker\_file.providers.mp3\_file.generators.gtts\_generator*, *method*), 109  
[handle\\_kwargs\(\)](#) (*faker\_file.providers.pdf\_file.generators.pdfkit\_generator*, *method*), 113  
[handle\\_kwargs\(\)](#) (*faker\_file.providers.pdf\_file.generators.reportlab\_generator*, *method*), 116  
  
**I**  
[ico\\_file\(\)](#) (*faker\_file.providers.ico\_file.IcoFileProvider*, *method*), 128  
[IcoFileProvider](#) (class in *faker\_file.providers.ico\_file*), 127  
[ImageMixin](#) (class in *faker\_file.providers.mixins.image\_mixin*), 107  
  
**J**  
[jpeg\\_file\(\)](#) (*faker\_file.providers.jpeg\_file.JpegFileProvider*, *method*), 129  
[JpegFileProvider](#) (class in *faker\_file.providers.jpeg\_file*), 128  
  
**L**  
[lang](#) (*faker\_file.providers.mp3\_file.generators.gtts\_generator.GttsMp3Generator*, *attribute*), 109  
[list\\_create\\_inner\\_file\(\)](#) (in module *faker\_file.providers.helpers.inner*), 106  
[load\\_class\\_from\\_path\(\)](#) (in module *faker\_file.helpers*), 155  
  
**M**  
[module](#)  
[faker\\_file](#), 156  
[faker\\_file.base](#), 154  
[faker\\_file.cli](#), 155  
[faker\\_file.constants](#), 155  
[faker\\_file.helpers](#), 155  
[faker\\_file.providers](#), 148  
[faker\\_file.providers.augment\\_file\\_from\\_dir.augmenters](#), 90  
[faker\\_file.providers.base](#), 92  
[faker\\_file.providers.base.mp3\\_generator](#), 90  
[faker\\_file.providers.base.pdf\\_generator](#), 91  
[faker\\_file.providers.base.text\\_augmenter](#), 91  
[faker\\_file.providers.base.text\\_extractor](#), 91  
  
[faker\\_file.providers.bin\\_file](#), 118  
[faker\\_file.providers.csv\\_file](#), 119  
[faker\\_file.providers.docx\\_file](#), 120  
[faker\\_file.providers.eml\\_file](#), 122  
[faker\\_file.providers.epub\\_file](#), 124  
[faker\\_file.providers.file\\_from\\_path](#), 125  
[faker\\_file.providers.generic\\_file](#), 126  
[faker\\_file.providers.helpers](#), 107  
[faker\\_file.providers.helpers.inner](#), 92  
[faker\\_file.providers.ico\\_file](#), 127  
[faker\\_file.providers.jpeg\\_file](#), 128  
[faker\\_file.providers.mixins](#), 108  
[faker\\_file.providers.mixins.image\\_mixin](#), 107  
[faker\\_file.providers.mixins.tablular\\_data\\_mixin](#), 107  
[faker\\_file.providers.mp3\\_file](#), 109  
[faker\\_file.providers.mp3\\_file.generators](#), 109  
[faker\\_file.providers.mp3\\_file.generators.edge\\_tts\\_generator](#), 108  
[faker\\_file.providers.mp3\\_file.generators.gtts\\_generator](#), 109  
[faker\\_file.providers.odp\\_file](#), 130  
[faker\\_file.providers.ods\\_file](#), 131  
[faker\\_file.providers.odt\\_file](#), 132  
[faker\\_file.providers.pdf\\_file](#), 116  
[faker\\_file.providers.pdf\\_file.generators](#), 116  
[faker\\_file.providers.pdf\\_file.generators.pdfkit\\_generator](#), 111  
[faker\\_file.providers.pdf\\_file.generators.reportlab\\_generator](#), 114  
[faker\\_file.providers.png\\_file](#), 135  
[faker\\_file.providers.pptx\\_file](#), 136  
[faker\\_file.providers.random\\_file\\_from\\_dir](#), 137  
[faker\\_file.providers.rtf\\_file](#), 138  
[faker\\_file.providers.svg\\_file](#), 139  
[faker\\_file.providers.tar\\_file](#), 140  
[faker\\_file.providers.txt\\_file](#), 141  
[faker\\_file.providers.webp\\_file](#), 142  
[faker\\_file.providers.xlsx\\_file](#), 144  
[faker\\_file.providers.xml\\_file](#), 145  
[faker\\_file.providers.zip\\_file](#), 147  
[faker\\_file storages](#), 152  
[faker\\_file storages.aws\\_s3](#), 148  
[faker\\_file storages.azure\\_cloud\\_storage](#), 149  
[faker\\_file storages.base](#), 149  
[faker\\_file storages.cloud](#), 150  
[faker\\_file storages.filesystem](#), 151  
[faker\\_file storages.google\\_cloud\\_storage](#), 151

[faker\\_file.tests](#), 154  
[faker\\_file.tests.data](#), 152  
[faker\\_file.tests.test\\_base](#), 152  
[faker\\_file.tests.test\\_cli](#), 152  
[faker\\_file.tests.test\\_django\\_integration](#), 153  
[faker\\_file.tests.test\\_providers](#), 153  
[faker\\_file.tests.test\\_storages](#), 154  
[faker\\_file.tests.texts](#), 154  
[mp3\\_file\(\)](#) ([faker\\_file.providers.mp3\\_file.Mp3FileProvider](#) method), 110  
[Mp3FileProvider](#) (class in [faker\\_file.providers.mp3\\_file](#)), 109

## N

[numerify](#) ([faker\\_file.base.FileMixin](#) attribute), 155  
[numerify](#) ([faker\\_file.providers.mixins.image\\_mixin.ImageMixin](#) attribute), 107  
[numerify](#) ([faker\\_file.providers.mixins.tablular\\_data\\_mixin.TabularDataMixin](#) attribute), 107

## O

[odp\\_file\(\)](#) ([faker\\_file.providers.odp\\_file.OdpFileProvider](#) method), 130  
[OdpFileProvider](#) (class in [faker\\_file.providers.odp\\_file](#)), 130  
[ods\\_file\(\)](#) ([faker\\_file.providers.ods\\_file.OdsFileProvider](#) method), 131  
[OdsFileProvider](#) (class in [faker\\_file.providers.ods\\_file](#)), 131  
[odt\\_file\(\)](#) ([faker\\_file.providers.odt\\_file.OdtFileProvider](#) method), 134  
[OdtFileProvider](#) (class in [faker\\_file.providers.odt\\_file](#)), 132

## P

[parse\\_format\\_func\(\)](#) (in module [faker\\_file.base](#)), 155  
[path](#) ([faker\\_file.providers.base.text\\_extractor.BaseTextExtractor](#) attribute), 91  
[PathyFileSystemStorage](#) (class in [faker\\_file.storages.cloud](#)), 150  
[pdf\\_file\(\)](#) ([faker\\_file.providers.pdf\\_file.PdfFileProvider](#) method), 117  
[PdfFileProvider](#) (class in [faker\\_file.providers.pdf\\_file](#)), 116  
[PdkitPdfGenerator](#) (class in [faker\\_file.providers.pdf\\_file.generators.pdkit\\_generator](#)), 111  
[png\\_file\(\)](#) ([faker\\_file.providers.png\\_file.PngFileProvider](#) method), 135  
[PngFileProvider](#) (class in [faker\\_file.providers.png\\_file](#)), 135  
[pptx\\_file\(\)](#) ([faker\\_file.providers.pptx\\_file.PptxFileProvider](#) method), 136  
[PptxFileProvider](#) (class in [faker\\_file.providers.pptx\\_file](#)), 136  
[ProvidersTestCase](#) (class in [faker\\_file.tests.test\\_providers](#)), 153  
[pystr\\_format\\_func\(\)](#) (in module [faker\\_file.base](#)), 155

## R

[random\\_element](#) ([faker\\_file.base.FileMixin](#) attribute), 155  
[random\\_element](#) ([faker\\_file.providers.mixins.image\\_mixin.ImageMixin](#) attribute), 107  
[random\\_element](#) ([faker\\_file.providers.mixins.tablular\\_data\\_mixin.TabularDataMixin](#) attribute), 107  
[random\\_file\\_from\\_dir\(\)](#) ([faker\\_file.providers.random\\_file\\_from\\_dir.RandomFileFromDirProvider](#) method), 137  
[RandomFileFromDirProvider](#) (class in [faker\\_file.providers.random\\_file\\_from\\_dir](#)), 137  
[relpath\(\)](#) ([faker\\_file.storages.base.BaseStorage](#) method), 149  
[relpath\(\)](#) ([faker\\_file.storages.cloud.CloudStorage](#) method), 150  
[relpath\(\)](#) ([faker\\_file.storages.filesystem.FileSystemStorage](#) method), 151  
[remove\\_string\(\)](#) ([faker\\_file.base.StringList](#) method), 155  
[ReportlabPdfGenerator](#) (class in [faker\\_file.providers.pdf\\_file.generators.reportlab\\_generator](#)), 114  
[returns\\_list\(\)](#) (in module [faker\\_file.base](#)), 155  
[rtf\\_file\(\)](#) ([faker\\_file.providers.rtf\\_file.RtfFileProvider](#) method), 138  
[RtfFileProvider](#) (class in [faker\\_file.providers.rtf\\_file](#)), 138

## S

[schema](#) ([faker\\_file.storages.aws\\_s3.AWSS3Storage](#) attribute), 148  
[schema](#) ([faker\\_file.storages.azure\\_cloud\\_storage.AzureCloudStorage](#) attribute), 149  
[schema](#) ([faker\\_file.storages.cloud.CloudStorage](#) attribute), 150  
[schema](#) ([faker\\_file.storages.cloud.PathyFileSystemStorage](#) attribute), 150  
[schema](#) ([faker\\_file.storages.google\\_cloud\\_storage.GoogleCloudStorage](#) attribute), 152  
[setUp\(\)](#) ([faker\\_file.tests.test\\_providers.ProvidersTestCase](#) method), 153  
[StringList](#) (class in [faker\\_file.base](#)), 155  
[StringValue](#) (class in [faker\\_file.base](#)), 155  
[svg\\_file\(\)](#) ([faker\\_file.providers.svg\\_file.SvgFileProvider](#) method), 139

SvgFileProvider (class in `faker_file.providers.svg_file`), 139

## T

TabularDataMixin (class in `faker_file.providers.mixins.tablular_data_mixin`), 107

tar\_file() (`faker_file.providers.tar_file.TarFileProvider` method), 141

TarFileProvider (class in `faker_file.providers.tar_file`), 140

test\_base\_storage\_exceptions (`faker_file.tests.test_storages.TestStoragesTestCase` attribute), 154

test\_broken\_imports (`faker_file.tests.test_providers.ProvidersTestCase` attribute), 153

test\_broken\_imports() (`faker_file.tests.test_cli.TestCLI` method), 152

test\_cli (`faker_file.tests.test_cli.TestCLI` attribute), 152

test\_cli\_error\_no\_provider() (`faker_file.tests.test_cli.TestCLI` method), 152

test\_cli\_generate\_completion() (`faker_file.tests.test_cli.TestCLI` method), 152

test\_cli\_version() (`faker_file.tests.test_cli.TestCLI` method), 152

test\_cloud\_storage\_exceptions (`faker_file.tests.test_storages.TestStoragesTestCase` attribute), 154

test\_faker (`faker_file.tests.test_providers.ProvidersTestCase` attribute), 153

test\_faker\_retry\_failures (`faker_file.tests.test_providers.ProvidersTestCase` attribute), 153

test\_file (`faker_file.tests.test_django_integration.DjangoIntegrationTestCase` attribute), 153

test\_file\_system\_storage\_abspath() (`faker_file.tests.test_storages.TestStoragesTestCase` method), 154

test\_load\_class\_from\_non\_existing\_path() (`faker_file.tests.test_providers.ProvidersTestCase` method), 153

test\_load\_class\_from\_path\_class\_not\_found() (`faker_file.tests.test_providers.ProvidersTestCase` method), 153

test\_load\_class\_from\_path\_no\_class\_type() (`faker_file.tests.test_providers.ProvidersTestCase` method), 153

test\_mp3\_file\_generate\_not\_implemented\_exception() (`faker_file.tests.test_providers.ProvidersTestCase` method), 153

test\_pathy\_file\_system\_storage\_abspath() (`faker_file.tests.test_storages.TestStoragesTestCase` method), 154

test\_pdf\_file\_generate\_not\_implemented\_exception() (`faker_file.tests.test_providers.ProvidersTestCase` method), 153

test\_raw\_standalone\_providers (`faker_file.tests.test_providers.ProvidersTestCase` attribute), 153

test\_raw\_standalone\_providers\_allow\_failures (`faker_file.tests.test_providers.ProvidersTestCase` attribute), 153

test\_raw\_standalone\_providers\_retry\_failures (`faker_file.tests.test_providers.ProvidersTestCase` attribute), 153

test\_standalone\_providers (`faker_file.tests.test_providers.ProvidersTestCase` attribute), 153

test\_standalone\_providers\_allow\_failures (`faker_file.tests.test_providers.ProvidersTestCase` attribute), 153

test\_standalone\_providers\_retry\_failures (`faker_file.tests.test_providers.ProvidersTestCase` attribute), 153

test\_standalone\_tar\_file (`faker_file.tests.test_providers.ProvidersTestCase` attribute), 153

test\_standalone\_tar\_file\_allow\_failures (`faker_file.tests.test_providers.ProvidersTestCase` attribute), 153

test\_standalone\_zip\_file (`faker_file.tests.test_providers.ProvidersTestCase` attribute), 153

test\_standalone\_zip\_file\_allow\_failures (`faker_file.tests.test_providers.ProvidersTestCase` attribute), 154

test\_storage (`faker_file.tests.test_storages.TestStoragesTestCase` attribute), 154

test\_storage\_generate\_filename\_exceptions (`faker_file.tests.test_storages.TestStoragesTestCase` attribute), 154

test\_storage\_initialization\_exceptions (`faker_file.tests.test_storages.TestStoragesTestCase` attribute), 154

test\_string\_list() (`faker_file.tests.test_base.TestStringListTestCase` method), 152

TestCLI (class in `faker_file.tests.test_cli`), 152

TestStoragesTestCase (class in `faker_file.tests.test_storages`), 154

TestStringListTestCase (class in `faker_file.tests.test_base`), 152

txt\_file() (`faker_file.providers.mp3_file.generators.gttts_generator.GtttsMp3Generator` attribute), 109

txt\_file() (`faker_file.providers.txt_file.TxtFileProvider` method), 109



*method*), [142](#)

`TxtFileProvider` (class in *faker\_file.providers.txt\_file*),  
[141](#)

## V

`voice` (*faker\_file.providers.mp3\_file.generators.edge\_tts\_generator.EdgeTtsMp3Generator*  
attribute), [108](#)

## W

`webp_file()` (*faker\_file.providers.webp\_file.WebpFileProvider*  
*method*), [143](#)

`WebpFileProvider` (class in  
*faker\_file.providers.webp\_file*), [142](#)

`wrap_text()` (in module *faker\_file.helpers*), [155](#)

`write_bytes()` (*faker\_file.storages.base.BaseStorage*  
*method*), [149](#)

`write_bytes()` (*faker\_file.storages.cloud.CloudStorage*  
*method*), [150](#)

`write_bytes()` (*faker\_file.storages.filesystem.FileSystemStorage*  
*method*), [151](#)

`write_text()` (*faker\_file.storages.base.BaseStorage*  
*method*), [149](#)

`write_text()` (*faker\_file.storages.cloud.CloudStorage*  
*method*), [150](#)

`write_text()` (*faker\_file.storages.filesystem.FileSystemStorage*  
*method*), [151](#)

## X

`xlsx_file()` (*faker\_file.providers.xlsx\_file.XlsxFileProvider*  
*method*), [144](#)

`XlsxFileProvider` (class in  
*faker\_file.providers.xlsx\_file*), [144](#)

`xml_file()` (*faker\_file.providers.xml\_file.XmlFileProvider*  
*method*), [146](#)

`XmlFileProvider` (class in  
*faker\_file.providers.xml\_file*), [145](#)

## Z

`zip_file()` (*faker\_file.providers.zip\_file.ZipFileProvider*  
*method*), [147](#)

`ZipFileProvider` (class in *faker\_file.providers.zip\_file*),  
[147](#)