
pytest-factoryboy Documentation

Release 2.5.1

Oleg Pidsadnyi, Anatoly Bubenkov and others

Dec 01, 2022

| | | |
|----------|---|-----------|
| 1 | factory_boy integration with the pytest runner | 1 |
| 1.1 | Install pytest-factoryboy | 1 |
| 1.2 | Concept | 1 |
| 1.3 | Factory Fixture | 1 |
| 1.4 | Model Fixture | 2 |
| 1.5 | Attributes are Fixtures | 3 |
| 1.6 | SubFactory | 3 |
| 1.7 | Related Factory | 3 |
| 1.8 | post-generation | 3 |
| 1.9 | Integration | 3 |
| 1.10 | Fixture partial specialization | 4 |
| 1.11 | Fixture attributes | 5 |
| 1.12 | Generic container classes as models | 5 |
| 2 | Post-generation dependencies | 7 |
| 2.1 | Hooks | 8 |
| 2.2 | License | 8 |
| 3 | Authors | 9 |
| 4 | Changelog | 11 |
| 4.1 | Unreleased | 11 |
| 4.2 | 2.5.1 | 11 |
| 4.3 | 2.5.0 | 11 |
| 4.4 | 2.4.0 | 11 |
| 4.5 | 2.3.1 | 12 |
| 4.6 | 2.3.0 | 12 |
| 4.7 | 2.2.1 | 12 |
| 4.8 | 2.2.0 | 12 |
| 4.9 | 2.1.0 | 13 |
| 4.10 | 2.0.3 | 13 |
| 4.11 | 2.0.2 | 13 |
| 4.12 | 2.0.1 | 13 |
| 4.13 | 1.3.2 | 13 |
| 4.14 | 1.3.1 | 13 |
| 4.15 | 1.3.0 | 14 |
| 4.16 | 1.2.2 | 14 |

| | | |
|------|-------|----|
| 4.17 | 1.2.1 | 14 |
| 4.18 | 1.1.6 | 14 |
| 4.19 | 1.1.5 | 14 |
| 4.20 | 1.1.3 | 14 |
| 4.21 | 1.1.2 | 14 |
| 4.22 | 1.1.1 | 14 |
| 4.23 | 1.1.0 | 14 |
| 4.24 | 1.0.3 | 15 |
| 4.25 | 1.0.2 | 15 |
| 4.26 | 1.0.1 | 15 |
| 4.27 | 1.0.0 | 15 |

factory_boy integration with the pytest runner

pytest-factoryboy makes it easy to combine `factory` approach to the test setup with the dependency injection, heart of the `pytest` fixtures.

1.1 Install pytest-factoryboy

```
pip install pytest-factoryboy
```

1.2 Concept

Library exports a function to register factories as fixtures. Fixtures are contributed to the same module where `register` function is called.

1.3 Factory Fixture

Factory fixtures allow using factories without importing them. The fixture name convention is to use the lowercase-underscore form of the class name.

```
import factory
from pytest_factoryboy import register
```

(continues on next page)

(continued from previous page)

```

class AuthorFactory(factory.Factory):
    class Meta:
        model = Author

register(AuthorFactory) # => author_factory

def test_factory_fixture(author_factory):
    author = author_factory(name="Charles Dickens")
    assert author.name == "Charles Dickens"

```

1.4 Model Fixture

Model fixture implements an instance of a model created by the factory. Name convention is model’s lowercase-underscore class name.

```

import factory
from pytest_factoryboy import register

@register
class AuthorFactory(factory.Factory):
    class Meta:
        model = Author

    name = "Charles Dickens"

def test_model_fixture(author):
    assert author.name == "Charles Dickens"

```

Model fixtures can be registered with specific names. For example, if you address instances of some collection by the name like “first”, “second” or of another parent as “other”:

```

register(AuthorFactory) # author
register(AuthorFactory, "second_author") # second_author

# `register(...)` can be used as a decorator too
@register # book
@register(_name="second_book") # second_book
@register(_name="other_book") # other_book, book of another author
class BookFactory(factory.Factory):
    class Meta:
        model = Book

@pytest.fixture
def other_book__author(second_author):
    """Make the relation of the second_book to another (second) author."""
    return second_author

```

1.5 Attributes are Fixtures

There are fixtures created for factory attributes. Attribute names are prefixed with the model fixture name and double underscore (similar to the convention used by `factory_boy`).

```
@pytest.mark.parametrize("author__name", ["Bill Gates"])
def test_model_fixture(author):
    assert author.name == "Bill Gates"
```

1.6 SubFactory

Sub-factory attribute points to the model fixture of the sub-factory. Attributes of sub-factories are injected as dependencies to the model fixture and can be [overridden](#) via the parametrization.

1.7 Related Factory

Related factory attribute points to the model fixture of the related factory. Attributes of related factories are injected as dependencies to the model fixture and can be [overridden](#) via the parametrization.

1.8 post-generation

Post-generation attribute fixture implements only the extracted value for the post generation function.

1.9 Integration

An example of `factory_boy` and `pytest` integration.

```
# factories/__init__.py

import factory
from faker import Factory as FakerFactory

faker = FakerFactory.create()

class AuthorFactory(factory.django.DjangoModelFactory):
    """Author factory."""

    name = factory.LazyAttribute(lambda x: faker.name())

    class Meta:
        model = 'app.Author'

class BookFactory(factory.django.DjangoModelFactory):
    """Book factory."""

    title = factory.LazyAttribute(lambda x: faker.sentence(nb_words=4))
```

(continues on next page)

(continued from previous page)

```
class Meta:
    model = 'app.Book'

author = factory.SubFactory(AuthorFactory)
```

```
# tests/conftest.py

from pytest_factoryboy import register

from factories import AuthorFactory, BookFactory

register(AuthorFactory)
register(BookFactory)
```

```
# tests/test_models.py

from app.models import Book
from factories import BookFactory

def test_book_factory(book_factory):
    """Factories become fixtures automatically."""
    assert book_factory is BookFactory

def test_book(book):
    """Instances become fixtures automatically."""
    assert isinstance(book, Book)

@pytest.mark.parametrize("book__title", ["PyTest for Dummies"])
@pytest.mark.parametrize("author__name", ["Bill Gates"])
def test_parametrized(book):
    """You can set any factory attribute as a fixture using naming convention."""
    assert book.name == "PyTest for Dummies"
    assert book.author.name == "Bill Gates"
```

1.10 Fixture partial specialization

There is a possibility to pass keyword parameters in order to override factory attribute values during fixture registration. This comes in handy when your test case is requesting a lot of fixture flavors. Too much for the regular pytest parametrization. In this case, you can register fixture flavors in the local test module and specify value deviations inside `register` function calls.

```
register(AuthorFactory, "male_author", gender="M", name="John Doe")
register(AuthorFactory, "female_author", gender="F")

@pytest.fixture
def female_author__name():
    """Override female author name as a separate fixture."""
    return "Jane Doe"
```

(continues on next page)

(continued from previous page)

```
@pytest.mark.parametrize("male_author__age", [42]) # Override even more
def test_partial(male_author, female_author):
    """Test fixture partial specialization."""
    assert male_author.gender == "M"
    assert male_author.name == "John Doe"
    assert male_author.age == 42

    assert female_author.gender == "F"
    assert female_author.name == "Jane Doe"
```

1.11 Fixture attributes

Sometimes it is necessary to pass an instance of another fixture as an attribute value to the factory. It is possible to override the generated attribute fixture where desired values can be requested as fixture dependencies. There is also a lazy wrapper for the fixture that can be used in the parametrization without defining fixtures in a module.

LazyFixture constructor accepts either existing fixture name or callable with dependencies:

```
import pytest
from pytest_factoryboy import register, LazyFixture

@pytest.mark.parametrize("book__author", [LazyFixture("another_author")])
def test_lazy_fixture_name(book, another_author):
    """Test that book author is replaced with another author by fixture name."""
    assert book.author == another_author

@pytest.mark.parametrize("book__author", [LazyFixture(lambda another_author: another_
↪author)])
def test_lazy_fixture_callable(book, another_author):
    """Test that book author is replaced with another author by callable."""
    assert book.author == another_author

# Can also be used in the partial specialization during the registration.
register(BookFactory, "another_book", author=LazyFixture("another_author"))
```

1.12 Generic container classes as models

It's often useful to create factories for dict or other common generic container classes. In that case, you should wrap the container class around `named_model(...)`, so that `pytest-factoryboy` can correctly determine the model name when using it in a `SubFactory` or `RelatedFactory`.

`Pytest-factoryboy` will otherwise raise a warning.

For example:

```
import factory
from pytest_factoryboy import named_model, register
```

(continues on next page)

(continued from previous page)

```
@register
class JSONPayload(factory.Factory):
    class Meta:
        model = named_model("JSONPayload", dict)

        name = "foo"

def test_foo(json_payload):
    assert json_payload.name == "foo"
```

As a bonus, factory is automatically registering the `json_payload` fixture (rather than `dict`), so there is no need to override `@register(_name="json_payload")`.

Post-generation dependencies

Unlike `factory_boy` which binds related objects using an internal container to store results of lazy evaluations, `pytest-factoryboy` relies on the PyTest request.

Circular dependencies between objects can be resolved using post-generation hooks/related factories in combination with passing the `SelfAttribute`, but in the case of PyTest request fixture functions have to return values in order to be cached in the request and to become available to other fixtures.

That's why evaluation of the post-generation declaration in `pytest-factoryboy` is deferred until calling the test function. This solves circular dependency resolution for situations like:

```
o->[ A ]-->[ B ]<--[ C ]-o
|           |
o----- (C depends on A) -----o
```

On the other hand, deferring the evaluation of post-generation declarations evaluation makes their result unavailable during the generation of objects that are not in the circular dependency, but they rely on the post-generation action.

`pytest-factoryboy` is trying to detect cycles and resolve post-generation dependencies automatically.

```
from pytest_factoryboy import register

class Foo(object):
    def __init__(self, value):
        self.value = value

class Bar(object):
    def __init__(self, foo):
        self.foo = foo

@register
class FooFactory(factory.Factory):
    """Foo factory."""
```

(continues on next page)

```

class Meta:
    model = Foo

value = 0

@factory.post_generation
def set1(foo, create, value, **kwargs):
    foo.value = 1

class BarFactory(factory.Factory):
    """Bar factory."""

    foo = factory.SubFactory(FooFactory)

    @classmethod
    def _create(cls, model_class, foo):
        assert foo.value == 1 # Assert that set1 is evaluated before object_
↪generation
        return super(BarFactory, cls)._create(model_class, foo=foo)

    class Meta:
        model = Bar

register(BarFactory, "bar")
"""Forces 'set1' to be evaluated first."""

def test_depends_on_set1(bar):
    """Test that post-generation hooks are done and the value is 2."""
    assert depends_on_1.foo.value == 1

```

2.1 Hooks

pytest-factoryboy exposes several [pytest hooks](#) which might be helpful for e.g. controlling database transaction, for reporting etc:

- `pytest_factoryboy_done(request)` - Called after all factory-based fixtures and their post-generation actions have been evaluated.

2.2 License

This software is licensed under the [MIT license](#).

© 2015 Oleg Pidsadnyi, Anatoly Bubenkov and others

CHAPTER 3

Authors

Oleg Pidsadnyi original idea and implementation

These people have contributed to *pytest-factoryboy*, in alphabetical order:

- Anatoly Bubenkov
- Daniel Duong
- Daniel Hahler
- Hugo van Kemenade
- p13773
- Vasily Kuznetsov

4.1 Unreleased

4.2 2.5.1

- Fix `PytestDeprecationWarning`. #180 #179

4.3 2.5.0

- Using a generic class container like `dict`, `list`, `set`, etc. will raise a warning suggesting you to wrap your model using `named_model(...)`. Doing this will make sure that the fixture name is correctly chosen, otherwise `SubFactory` and `RelatedFactory` aren't able to determine the name of the model. See [Generic Container Classes as models #167](#)
- Fix `Factory._after_postgeneration` being invoked twice. #164 #156
- Stack traces caused by `pytest-factoryboy` are now slimmer. #169
- Check for naming conflicts between factory and model fixture name, and raise a clear error immediately. #86

4.4 2.4.0

- `LazyFixture` is now a `Generic[T]` type.
- Simplify fixture generation (internal change).
- Use poetry (internal change).

4.5 2.3.1

- Fix `AttributeError` when using `LazyFixture` in `register(...)` #159 #158

4.6 2.3.0

- Add support for `factory.PostGenerationMethodCall` #103 #87.

4.7 2.2.1

- `@register()` decorator now refuses kwargs after the initial specialization. This behaviour was mistakenly introduced in version 2.2.0, and it complicates the usage of the `register` function unnecessarily. For example, the following is not allowed anymore:

```
# INVALID
register(
    _name="second_author",
    name="C.S. Lewis",
) (
    AuthorFactory,
    register_user="cs_lewis",
    register_user__password="Aslan1",
)

# VALID
register(
    AuthorFactory,
    _name="second_author",
    name="C.S. Lewis",
    register_user="cs_lewis",
    register_user__password="Aslan1",
)
```

4.8 2.2.0

- Drop support for Python 3.6. We now support only python ≥ 3.7 .
- Improve “debuggability”. Internal `pytest-factoryboy` calls are now visible when using a debugger like `PDB` or `PyCharm`.
- Add type annotations. Now `register` and `LazyFixture` are type annotated.
- Fix `Factory_after_postgeneration` method not getting the evaluated `post_generations` and `RelatedFactory` results correctly in the `result` param.
- Factories can now be registered inside classes (even nested classes) and they won’t pollute the module namespace.
- Allow the `@register` decorator to be called with parameters:


```
@register
@register("other_author")
class AuthorFactory(Factory):
    ...
```

4.9 2.1.0

- Add support for factory_boy \geq 3.2.0
- Drop support for Python 2.7, 3.4, 3.5. We now support only python \geq 3.6.
- Drop support for pytest $<$ 4.6. We now support only pytest \geq 4.6.
- Add missing versions of python (3.9 and 3.10) and pytest (6.x.x) to the CI test matrix.

4.10 2.0.3

- Fix compatibility with pytest 5.

4.11 2.0.2

- Fix warning *use of getfuncargvalue is deprecated, use getfixturevalue* (sliverc)

4.12 2.0.1

Breaking change due to the heavy refactor of both pytest and factory_boy.

- Failing test for using a *attributes* field on the factory (blueyed)
- Minimal pytest version is 3.3.2 (olegpidsadnyi)
- Minimal factory_boy version is 2.10.0 (olegpidsadnyi)

4.13 1.3.2

- use {posargs} in pytest command (blueyed)
- pin factory_boy $<$ 2.9 (blueyed)

4.14 1.3.1

- fix LazyFixture evaluation order (olegpidsadnyi)

4.15 1.3.0

- replace request._fixturedefs by request._fixture_defs (p13773)

4.16 1.2.2

- fix post-generation dependencies (olegpidsadnyi)

4.17 1.2.1

- automatic resolution of the post-generation dependencies (olegpidsadnyi, kvas-it)

4.18 1.1.6

- fixes fixture function module name attribute (olegpidsadnyi)
- fixes _after_postgeneration hook invocation for deferred post-generation declarations (olegpidsadnyi)

4.19 1.1.5

- support factory models to be passed as strings (bubenkoff)

4.20 1.1.3

- circular dependency determination is fixed for the post-generation (olegpidsadnyi)

4.21 1.1.2

- circular dependency determination is fixed for the RelatedFactory attributes (olegpidsadnyi)

4.22 1.1.1

- fix installation issue when django environment is not set (bubenkoff, amaknach)

4.23 1.1.0

- fixture dependencies on deferred post-generation declarations (olegpidsadnyi)

4.24 1.0.3

- `post_generation` extra parameters fixed (olegpidsadnyi)
- fixture partial specialization (olegpidsadnyi)
- fixes readme and example (dduong42)
- lazy fixtures (olegpidsadnyi)
- deferred post-generation evaluation (olegpidsadnyi)
- hooks (olegpidsadnyi)

4.25 1.0.2

- refactoring of the fixture function compilation (olegpidsadnyi)
- related factory fix (olegpidsadnyi)
- `post_generation` fixture dependency fixed (olegpidsadnyi)
- model fixture registration with specific name (olegpidsadnyi)
- README updated (olegpidsadnyi)

4.26 1.0.1

- use `inflection` package to convert camel case to underscore (bubenkoff)

4.27 1.0.0

- initial release (olegpidsadnyi)