Mocki Documentation

Release 1.7.4

Olivier Kozak

July 08, 2014

Contents

Mocki aims to be an easy-to-use but full featured mocking library for Python.

CHAPTER 1

Installation

Here is how to install Mocki using pip :

```
pip install Mocki
```

CHAPTER 2

Starting with Mocki

Here is how to instantiate a new mock with Mocki :

```
>>> import mocki.core
>>>
>>> mock = mocki.core.Mock('myMock')
```

There are basically two things we can do with this mock :

- stub it to do a particular action on a particular call :

```
>>> mock.on_call('myCall').do_return('myValue')
>>>
>>> mock('myCall')
'myValue'
```

- verify whether or not a particular call was invoked on it :

```
>>> mock.verify_call('myCall').invoked_once()
>>>
>>> mock.verify_call('myCall').invoked_exactly(2)
Traceback (most recent call last):
...
AssertionError: Found one matching call invoked from myMock :
> myMock('myCall')
```

CHAPTER 3

Contents

3.1 API reference

class mocki.core.**Fake**(***properties*)

A fake is an object instantiated from its properties.

Fakes are very useful in tests. They allow testers to instantiate objects from their properties in just one line, thus being an elegant way to create data structures on fly.

Here is how to get a new fake :

```
>>> from mocki.core import Fake
>>>
>>> fake = Fake(property='value', otherProperty='otherValue')
```

This new fake takes the properties given on instantiation :

```
>>> fake.property
'value'
>>>
>>> fake.otherProperty
'otherValue'
```

class mocki.core.**Mock**(*name*)

A mock is a callable object that keeps track of any call invocation made from it.

Here is how to get a new mock :

```
>>> from mocki.core import Mock
>>>
>>> mock = Mock('theMock')
```

It must be noted that mocks' members are mocks themselves :

```
>>> mock.theMember
<mocki.core.Mock object at ...>
>>>
>>> mock.theOtherMember
<mocki.core.Mock object at ...>
```

These new mocks take their names from the accessed members' names plus the name of the parent mock as follows :

```
>>> mock.theMember.name
'theMock.theMember'
```

An important thing to note about mocks' members is that any member got from a parent mock under the same name is the same member :

```
>>> mock.theMember is mock.theMember
True
```

on(*matcher*)

Installs a new stub to change the mock's behavior.

Suppose we have the following mock :

```
>>> from mocki.core import Mock
>>>
>>> mock = Mock('theMock')
```

By default, any call invoked from this mock returns nothing :

```
>>> mock('1stCall')
```

This behavior can be changed as follows :

```
>>> mock.on(
...     lambda call_invocation: call_invocation.args == ('2ndCall',)
... ).do(
...     lambda call_invocation: '2ndValue'
... )
```

A custom stub is now installed on the mock, such as any call invocation made from it by passing '2ndCall' will now return '2ndValue' :

```
>>> mock('2ndCall')
'2ndValue'
```

Note that any other call invocation still returns nothing :

```
>>> mock('1stCall')
```

Here are some explanations.

The statement used to install custom stubs is the following : mock.on(matcher).do(action)

The matcher is a filter that describes on which call the stub will be applied. More concretely, it is a function taking a call invocation and returning true or false depending on whether this call invocation is suitable or not with the call we would like to stub. In our example, only call invocations made by passing '2ndCall' will be affected.

The action is the function that will be executed whenever the matcher returns true. In our example, we simply returns '2ndValue'.

It works well, but you may here wonder why we should write such a verbose stubbing statement for such a simple stub. It was to show you the fully customizable form of the stubbing statement, but of course, Mocki is shipped with a set of common matchers and actions, thus the above statement is strictly equivalent to the following one :

```
>>> mock.on_call('2ndCall').do_return('2ndValue')
```

That's much simpler !

Another interesting thing to note about stubs is that they may also be partially overridden. This is done by declaring the more specific stub after the general one to override :

```
>>> mock.on_any_call().do_return('defaultValue')
>>>
>>> mock.on_call('2ndCall').do_return('2ndValue')
```

Now, any call invoked from this mock will return 'defaultValue', except for those made by passing '2nd-Call' which will return '2ndValue' :

```
>>> mock('1stCall')
'defaultValue'
>>>
>>> mock('2ndCall')
'2ndValue'
>>>
>>> mock('3rdCall')
'defaultValue'
```

verify(*matcher*)

Verifies if a given call was invoked as expected from the given mock.

If the given call was invoked as expected from the mock, this function just returns silently, otherwise it raises an exception indicating which call invocations are matching to the given call and which ones are not.

Suppose we made the following call invocations :

```
>>> from mocki.core import Mock
>>>
>>> mock = Mock('theMock')
>>>
>>> mock('1stCall')
>>> mock('2ndCall')
>>> mock('3rdCall')
```

Let's verify if the 2nd call was invoked once :

```
>>> mock.verify(
...     lambda call_invocation: call_invocation.args == ('2ndCall',)
... ).invoked(
...     lambda call_invocations: len(call_invocations) == 1
... )
```

Here are some explanations.

The statement used to do custom verifications is the following : mock.verify(matcher).invoked(expectation)

The matcher is a filter that describes the call. More concretely, it is a function taking a call invocation and returning true or false depending on whether it is suitable or not with the given call, which is used to filter the call invocations made from the mock.

The expectation is an assertion applied to the filtered call invocations that describes what is expected about the given call. It is a function taking the filtered call invocations and returning true or false depending on whether they meet or not the assertion.

It works well, but you may here wonder why we should write such a verbose verification statement for such a simple assertion. It was to show you the fully customizable form of the verification statement, but of course, Mocki is shipped with a set of common matchers and expectations, thus the above statement is strictly equivalent to the following one :

```
>>> mock.verify_call('2ndCall').invoked_once()
```

Now, let's try to verify something wrong :

```
>>> mock.verify_call('2ndCall').invoked_never()
Traceback (most recent call last):
...
AssertionError: Found one matching call invoked from theMock :
  theMock('1stCall')
> theMock('2ndCall')
  theMock('3rdCall')
```

An assertion error is raised along with a message in which matching call invocations are spotted by arrows. Here we found one matching call invocation while none was expected.

Sometimes, it can be useful to verify whether or not a particular call was made in a particular order. This can be done using the in order verification statement.

An important thing to note about this statement is that it can only be used from mocks that are sharing the same parent. This is not a real problem though, since mocks can easily be instantiated from other mocks :

```
>>> new_mock, new_other_mock = mock.theNewMock, mock.theNewOtherMock
```

Suppose we made the following call invocations :

```
>>> new_mock('1stCall')
>>> new_other_mock('2ndCall')
>>> new_mock('3rdCall')
>>> new_other_mock('4thCall')
```

Then, we made the following verification :

```
>>> new_other_mock.verify_call('2ndCall').invoked_once()
```

Now, let's verify if the 1st call was invoked in order :

```
>>> new_mock.verify_call('1stCall').invoked_in_order(new_other_mock)
Traceback (most recent call last):
...
AssertionError: Found one matching call invoked from theMock.theNewMock, but not in order :
  > theMock.theNewMock('1stCall')
X   theMock.theNewOtherMock('2ndCall')
    theMock.theNewOtherMock('4thCall')
```

An assertion error is raised along with a message in which matching call invocations are spotted by arrows while already verified ones are spotted by marks. Here we found one matching call invocation, but not in order : the 2nd call has already been verified, which means that it was expected to be invoked before.

Here are some explanations.

The statement used to do custom in order verifications is the following :
mock.verify(matcher).invoked_in_order(considered_mocks)

The matcher is a filter that describes the call. More concretely, it is a function taking a call invocation and returning true or false depending on whether it is suitable or not with the given call, which is used to filter the call invocations made from the mock.

The considered mocks are the mocks on which the statement applies. If at least one call invocation coming from these mocks was expected to be invoked after, the statement will fail.

If the verified call was invoked in order, this function just returns silently :

```
>>> new_mock.verify_call('3rdCall').invoked_in_order(new_other_mock)
```

verify_no_more_call_invoked()
Verifies if there was no more call invoked from the given mock.

If there was no more call invoked from the given mock, this function just returns silently, otherwise it raises an exception indicating which call invocations were already verified and which ones were not.

Suppose we made the following call invocations :

```
>>> from mocki.core import Mock
>>>
>>> mock = Mock('theMock')
>>>
>>> mock('1stCall')
>>> mock('2ndCall')
>>> mock('3rdCall')
```

Then we made the following verifications :

```
>>> mock.verify_call('1stCall').invoked_once()
>>> mock.verify_call('3rdCall').invoked_once()
```

If we now ask to verify if there was no more call invoked from this mock, an assertion error is raised along with a message in which already verified call invocations are spotted by marks :

```
>>> mock.verify_no_more_call_invoked()
Traceback (most recent call last):
...
AssertionError: Found one call invoked from theMock that was not verified :
X theMock('1stCall')
  theMock('2ndCall')
X theMock('3rdCall')
```

We can see that one call invocation was not verified. So let's verify it :

```
>>> mock.verify_call('2ndCall').invoked_once()
```

Now, this function just returns silently :

```
>>> mock.verify_no_more_call_invoked()
```

It must be noted that any call invoked from mock's members is here taken into account, which explains why we get an assertion error once again when we invoke some calls from them :

```
>>> mock.theMember('4thCall')
>>>
>>> mock.verify_no_more_call_invoked()
Traceback (most recent call last):
...
AssertionError: Found one call invoked from theMock that was not verified :
X theMock('1stCall')
X theMock('2ndCall')
X theMock('3rdCall')
  theMock.theMember('4thCall')
```

class mocki.matchers.**AnyCall**

A matcher that matches any call invocation.

This matcher is usable either from verification or stubbing statements.

Suppose we made the following call invocations :

```
>>> from mocki.core import Mock
>>>
>>> mock = Mock('theMock')
```

```
>>>
>>> mock('1stCall')
>>> mock('2ndCall')
>>> mock('3rdCall')
```

As we can see from the following verification statement, this matcher will match any call invoked from the mock :

```
>>> mock.verify_any_call().invoked_once()
Traceback (most recent call last):
...
AssertionError: Found 3 matching calls invoked from theMock :
> theMock('1stCall')
> theMock('2ndCall')
> theMock('3rdCall')
```

In a stubbing statement, this matcher is used to constantly execute the same action, no matter what arguments were provided to invoke the call.

Here is an example showing how to use it to stub the mock :

```
>>> mock.on_any_call().do_return('value')
```

With this stub installed, 'value' will now be constantly returned on any call invocation made from the mock :

```
>>> mock('1stCall')
'value'
>>>
>>> mock('2ndCall')
'value'
```

class mocki.matchers.**Call**(**args*, ***kwargs*)

A matcher that matches call invocations made with the given arguments.

This matcher is usable either from verification or stubbing statements.

Suppose we made the following call invocations :

```
>>> from mocki.core import Mock
>>>
>>> mock = Mock('theMock')
>>>
>>> mock('1stCall')
>>> mock('2ndCall')
>>> mock('2ndCall')
>>> mock('3rdCall')
```

As we can see from the following verification statement, this matcher will only match the calls invoked from the mock with the given arguments :

```
>>> mock.verify_call('2ndCall').invoked_once()
Traceback (most recent call last):
...
AssertionError: Found 2 matching calls invoked from theMock :
  theMock('1stCall')
> theMock('2ndCall')
> theMock('2ndCall')
  theMock('3rdCall')
```

In the previous verification statement, we used values for arguments. This is the most common situation, but sometimes, it may be useful to describe these arguments with richer assertions. This can be done using callables

:

```
>>> mock.verify_call(lambda value: value != '2ndCall').invoked_once()
Traceback (most recent call last):
...
AssertionError: Found 2 matching calls invoked from theMock :
> theMock('1stCall')
  theMock('2ndCall')
  theMock('2ndCall')
> theMock('3rdCall')
```

We can also use the more expressive Hamcrest version :

```
>>> from hamcrest import equal_to, is_not
>>>
>>> mock.verify_call(is_not(equal_to('2ndCall'))).invoked_once()
Traceback (most recent call last):
...
AssertionError: Found 2 matching calls invoked from theMock :
> theMock('1stCall')
  theMock('2ndCall')
  theMock('2ndCall')
> theMock('3rdCall')
```

In a stubbing statement, this matcher is used to execute different actions on different call invocations.

Here is an example showing how to use it to stub the mock :

```
>>> mock.on_call('1stCall').do_return('value')
>>> mock.on_call('3rdCall').do_return('otherValue')
```

With this stub installed, 'value' will now be returned each time a call invocation is made from the mock by passing '1stCall', while 'otherValue' will be returned for those made by passing '3rdCall' :

```
>>> mock('1stCall')
'value'
>>>
>>> mock('3rdCall')
'otherValue'
```

class mocki.actions.**Return**(*value*)

An action that returns the given value.

Suppose we have the following mock :

```
>>> from mocki.core import Mock
>>>
>>> mock = Mock('theMock')
```

Here is how to stub this mock to return a value :

```
>>> mock.on_any_call().do_return('value')
```

Now, any call invocation made from it will return this value :

```
>>> mock('1stCall')
'value'
```

class mocki.actions.**Raise**(*exception*)

An action that raises the given exception.

Suppose we have the following mock :

```
>>> from mocki.core import Mock
>>>
>>> mock = Mock('theMock')
```

Here is how to stub this mock to raise an exception :

```
>>> mock.on_any_call().do_raise(Exception('error'))
```

Now, any call invocation made from it will raise this exception :

```
>>> mock('1stCall')
Traceback (most recent call last):
...
Exception: error
```

class mocki.actions.**InOrder**(*stub*, **stubs*)

An action that takes a list of actions to execute in order.

The 1st action will be executed on the 1st matching call invocation, the 2nd action on the 2nd one, the 3rd action on the 3rd one, and so on, up to the last action that will be repeatedly executed on any further matching call invocation.

Suppose we have the following mock :

```
>>> from mocki.core import Mock
>>>
>>> mock = Mock('theMock')
```

Here is an example showing how to stub this mock with a list of actions :

```
>>> mock.on_any_call().do_in_order(Return('value'), Return('otherValue'))
```

With this stub installed, the 1st call invocation made from the mock will now return 'value', while any further one will return 'otherValue' :

```
>>> mock('1stCall')
'value'
>>>
>>> mock('2ndCall')
'otherValue'
>>>
>>> mock('3rdCall')
'otherValue'
```

class mocki.expectations.**AtLeast**(*n_times*)

An expectation that returns true when there is at least N matching call invocations.

Suppose we have the following mock :

```
>>> from mocki.core import Mock
>>>
>>> mock = Mock('theMock')
```

With no call invoked from this mock, here is what we get when we ask to verify if it was invoked at least 1 time :

```
>>> mock.verify_any_call().invoked_at_least(1)
Traceback (most recent call last):
...
AssertionError: No call invoked from theMock.
```

With one call invoked :

```
>>> mock('1stCall')
>>>
>>> mock.verify_any_call().invoked_at_least(1)
```

With 2 calls invoked :

```
>>> mock('2ndCall')
>>>
>>> mock.verify_any_call().invoked_at_least(1)
```

class mocki.expectations.**AtLeastOnce**

An expectation that returns true when there is at least one matching call invocation.

Suppose we have the following mock :

```
>>> from mocki.core import Mock
>>>
>>> mock = Mock('theMock')
```

With no call invoked from this mock, here is what we get when we ask to verify if is was invoked at least once :

```
>>> mock.verify_any_call().invoked_at_least_once()
Traceback (most recent call last):
...
AssertionError: No call invoked from theMock.
```

With one call invoked :

```
>>> mock('1stCall')
>>>
>>> mock.verify_any_call().invoked_at_least_once()
```

With 2 calls invoked :

```
>>> mock('2ndCall')
>>>
>>> mock.verify_any_call().invoked_at_least_once()
```

class mocki.expectations.**AtMost**(*n_times*)

An expectation that returns true when there is at most N matching call invocations.

Suppose we have the following mock :

```
>>> from mocki.core import Mock
>>>
>>> mock = Mock('theMock')
```

With no call invoked from this mock, here is what we get when we ask to verify if it was invoked at most 1 time :

```
>>> mock.verify_any_call().invoked_at_most(1)
```

With one call invoked :

```
>>> mock('1stCall')
>>>
>>> mock.verify_any_call().invoked_at_most(1)
```

With 2 calls invoked :

```
>>> mock('2ndCall')
>>>
>>> mock.verify_any_call().invoked_at_most(1)
Traceback (most recent call last):
...
AssertionError: Found 2 matching calls invoked from theMock :
> theMock('1stCall')
> theMock('2ndCall')
```

class mocki.expectations.**AtMostOnce**

An expectation that returns true when there is at most one matching call invocation.

Suppose we have the following mock :

```
>>> from mocki.core import Mock
>>>
>>> mock = Mock('theMock')
```

With no call invoked from this mock, here is what we get when we ask to verify if it was invoked at most once :

```
>>> mock.verify_any_call().invoked_at_most_once()
```

With one call invoked :

```
>>> mock('1stCall')
>>>
>>> mock.verify_any_call().invoked_at_most_once()
```

With 2 calls invoked :

```
>>> mock('2ndCall')
>>>
>>> mock.verify_any_call().invoked_at_most_once()
Traceback (most recent call last):
...
AssertionError: Found 2 matching calls invoked from theMock :
> theMock('1stCall')
> theMock('2ndCall')
```

class mocki.expectations.**Between**(*n_times*, *m_times*)

An expectation that returns true when there is between N and M matching call invocations.

```
>>> from mocki.core import Mock
>>>
>>> mock = Mock('theMock')
```

With no call invoked from this mock, here is what we get when we ask to verify if it was invoked between 1 and 3 times :

```
>>> mock.verify_any_call().invoked_between(1, 3)
Traceback (most recent call last):
...
AssertionError: No call invoked from theMock.
```

With one call invoked :

```
>>> mock('1stCall')
>>>
>>> mock.verify_any_call().invoked_between(1, 3)
```

With 2 calls invoked :

```
>>> mock('2ndCall')
>>>
>>> mock.verify_any_call().invoked_between(1, 3)
```

With 3 calls invoked :

```
>>> mock('3rdCall')
>>>
>>> mock.verify_any_call().invoked_between(1, 3)
```

With 4 calls invoked :

```
>>> mock('4thCall')
>>>
>>> mock.verify_any_call().invoked_between(1, 3)
Traceback (most recent call last):
...
AssertionError: Found 4 matching calls invoked from theMock :
> theMock('1stCall')
> theMock('2ndCall')
> theMock('3rdCall')
> theMock('4thCall')
```

class mocki.expectations.**Exactly**(*n_times*)

An expectation that returns true when there is exactly N matching call invocations.

Suppose we have the following mock :

```
>>> from mocki.core import Mock
>>>
>>> mock = Mock('theMock')
```

With no call invoked from this mock, here is what we get when we ask to verify if it was invoked exactly 1 time :

```
>>> mock.verify_any_call().invoked_exactly(1)
Traceback (most recent call last):
...
AssertionError: No call invoked from theMock.
```

With one call invoked :

```
>>> mock('1stCall')
>>>
>>> mock.verify_any_call().invoked_exactly(1)
```

With 2 calls invoked :

```
>>> mock('2ndCall')
>>>
>>> mock.verify_any_call().invoked_exactly(1)
Traceback (most recent call last):
...
AssertionError: Found 2 matching calls invoked from theMock :
> theMock('1stCall')
> theMock('2ndCall')
```

class mocki.expectations.**Never**

An expectation that returns true when there is no matching call invocation.

Suppose we have the following mock :

```
>>> from mocki.core import Mock
>>>
>>> mock = Mock('theMock')
```

With no call invoked from this mock, here is what we get when we ask to verify if it was never invoked :

```
>>> mock.verify_any_call().invoked_never()
```

With one call invoked :

```
>>> mock('1stCall')
>>>
>>> mock.verify_any_call().invoked_never()
Traceback (most recent call last):
...
AssertionError: Found one matching call invoked from theMock :
> theMock('1stCall')
```

class mocki.expectations.**Once**

An expectation that returns true when there is one matching call invocation.

Suppose we have the following mock :

```
>>> from mocki.core import Mock
>>>
>>> mock = Mock('theMock')
```

With no call invoked from this mock, here is what we get when we ask to verify if it was invoked once :

```
>>> mock.verify_any_call().invoked_once()
Traceback (most recent call last):
...
AssertionError: No call invoked from theMock.
```

With one call invoked :

```
>>> mock('1stCall')
>>>
>>> mock.verify_any_call().invoked_once()
```

With 2 calls invoked :

```
>>> mock('2ndCall')
>>>
>>> mock.verify_any_call().invoked_once()
Traceback (most recent call last):
...
AssertionError: Found 2 matching calls invoked from theMock :
> theMock('1stCall')
> theMock('2ndCall')
```