pymox Documentation

Release UNRELEASED

Ivan Neto

Jan 31, 2024

CONTENTS

1 New Elegant Way **3**

2 Classic Way **5**

3 Jurassic Way **7**
3.1 Why . . . 7

4 Basics **13**
4.1 Getting Started . . . 13
4.2 Install . . . 19

5 Guides **21**
5.1 Tutorial [0] . . . 21
5.2 Tutorial [1] . . . 33

6 *Pymox* in Practice **41**
6.1 Recipes . . . 41

7 API Reference **57**
7.1 Reference . . . 57

8 Project Links **59**

9 Indices and tables **61**

Python mocking on steroids

Release **UNRELEASED** (What's new?)

If you'd like more information on why Pymox, check out our *summary*.

Otherwise, let's dive right in!

CHAPTER

ONE

NEW ELEGANT WAY

```
# conftest.py
pytest_plugins = ("mox.testing.pytest_mox",)


# test.py
from mox import expect, stubout


class TestOs:
     def test_getcwd(self):
        with stubout(os, 'getcwd') as m_getcwd, expect:
            m_getcwd.to_be.called_with().and_return('/mox/path')

        assert os.getcwd() == '/mox/path'
        mox.verify(m_getcwd)
```

CHAPTER

TWO

CLASSIC WAY

```
import mox
import os

class TestOs:
    def test_getcwd(self):
        m = mox.Mox()

        m.stubout(os, 'getcwd')
        # calls
        os.getcwd().returns('/mox/path')

        m.replay_all()
        assert os.getcwd() == '/mox/path'
        m.verify_all()


if __name__ == '__main__':
    import unittest
    unittest.main()
```

CHAPTER

THREE

JURASSIC WAY

```
import mox
import os


class TestOs(mox.MoxTestBase):
    def test_getcwd(self):
        self.mox.StubOutWithMock(os, 'getcwd')
        # calls
        os.getcwd().AndReturn('/mox/path')

        self.mox.ReplayAll()
        self.assertEqual(os.getcwd(), '/mox/path')
        self.mox.VerifyAll()


if __name__ == '__main__':
    import unittest
    unittest.main()
```

3.1 Why

Why Pymox?

3.1.1 Comparing Mox to existing Python mock libraries

An engineer has a couple of options when picking a mock object library to use for testing. In this document I will do my best to outline the differences between Mox and other open source alternatives, and why you may or may not prefer to use Mox.

3.1.2 Other options

Believe it or not, similar code does exist.

- The Python Mock Module (http://python-mock.sourceforge.net/): This was the first mock module I used in Python, and arguably the reason why I wrote Mox.
- pMock (http://pmock.sourceforge.net/): Inspired by jMock, the other side of Java inspired coin.
- Python Mocker (http://labix.org/mocker): The most similar to Mox.

3.1.3 Simple use-case comparison

For now, let's just describe the basic use-case for a mock object. Let's pretend that we have a class that generates random greeting in a very complex, expensive, way.

```
class Greeter(object):
    def get(self): # expensive code here return generated_greeting
```

Mox

```
mock_greeter = mox.create.any()
mock_greeter.get().and_return("hello")
mox.replay(mock_greeter)
assert == "hello", mock_greeter.get()
mox.verify(mock_greeter)
```

Python Mock Module

Python Mock Module uses a dictionary to define the expected behavior, and verify methods were called. Using strings for method names seems very fragile to me. It would also complicate any auto-magic refactoring you might do.

```
mock_greeter = mock.Mock( {"get" : "hello" })
assertEquals("hello", mock_greeter.get())
mock.mockCheckCall(self, 0, "get")
```

pMock

pMock is extremely verbose, and probably very flexible. Because it lacks a replay state, any calls to the mock that weren't recorded before hand might silently be recorded as void method calls.

```
mock_greeter = pmock.Mock()
mock_greeter.expects(pmock.once()).get().will(pmock.return_value("hello"))
assert "hello" == mock_greeter.get()
mock_greeter.verify()
```

Mocker

Mocker is very similar to Mox, but with Mocker the return value is set on the mocker module rather than the mock itself, which seems awkward to me.

```
mock_greeter = mocker.mock()
mock_greeter.get()
mocker.result("hello")
mocker.replay()
assert "hello" == mock_greeter.get()
mocker.verify()
```

3.1.4 Mocking real objects, and caring

Some of the libraries listed above don't (seem to) offer support for a mock enforcing the interface of the mocked-class. This kind of enforcement can be very helpful.

Mox

Mox's MockObject enforces the interface of the supplied object.

```
mock_greeter = mox.MockObject(Greeter)
mock_greeter.get().and_return("hello")
mox.replay(mock_greeter)
assert "hello" == mock_greeter.get()
mox.verify(mock_greeter)
```

Python Mock Module

```
mock_greeter = mock.Mock({"get", "hello"}, Greeter)
assert "hello" == mock_greeter.get()
mock.mockCheckCall(self, 0, "get")
```

pMock

Does not seem to support this.

Mocker

Hurray, Mocker does this.

```
mock_greeter = mocker.mock(Greeter)
mock_greeter.get()
mocker.result("hello")
mocker.replay()
assert "hello" == mock_greeter.get()
mocker.verify()
```

3.1.5 Ordering / unordering of calls

Sometimes it is necessary for calls to be ordered (opening datacase connection before issuing a query), or un-ordered (iterating over a dictionary is non-deterministic). Handling this is done differently in the different libraries.

Mox

Mox imposes ordering by default, because determinism is a good thing (tm) in tests. Methods are expected to be called in the order they are recorded in. In the case that you need to iterate over a dictionary, or do other operations with undefined order, you may use unordered groups

3.1.6 Iterating over dictionary keys

```
mock_int_to_str.convert(1).any_order().and_return("one")
mock_int_to_str.convert(2).any_order().and_return("two")
```

Python Mock Module

The Python Mock Module doesn't impose any ordering on calls, they're just dictionary lookups. The only ordering is done through mockCheckCall or mockSetExpectation. I'm not sure if a lack of ordering is even possible, or specifying different return values based on parameters. Ugh.

```
TODO?
```

pMock

By default, the calls are unordered, but order can be defined by labeling calls

```
mock_db.expects(pmock.once()).open_connection().id("open")
mock_db.expects(pmock.once()).query().id("query").after("open")
```

Mocker

Like the other libraries, order is not enforced. Instead of using method names, it uses block notation, which seems pretty neat.

```
with mocker.order():
    mock_db.open_connection()
    mock_db.query()
```

3.1.7 Raising exceptions

Often times you'll want to test that your code not only works properly, but fails elegantly. In these cases, you would like your mock to return some unexpected value (easy to do), or raise an exception.

Mox

```
mock_greeter.get().and_raise(Exception("no greetings left"))
mox.replay(mock_greeter)
with pytest.raises(Exception, match="no greetings left"):
    mock_greeter.get()
mox.verify(mock_greeter)
```

Python Mock Module

Once again, strings are used for method names, and verification is done by hand.

```
mock_greeter.mockSetExpectation('get', expectException(Exception))
assertRaises(Exception, mock_greeter.get)
mock.mockCheckCall(self, 0, "get")
```

pMock

```
mock_greeter.expects(pmock.once()).get().will(pmock.raise_exception(Exception("no␣
↪greetings left")))
with pytest.raises(Exception, match="no greetings left"):
    mock_greeter.get()
mock_greeter.verify()
```

Mocker

```
mock_greeter.get()
mocker.throw(Exception("no greetings left")
mocker.replay()
with pytest.raises(Exception, match="no greetings left"):
    mock_greeter.get()
mocker.verify()
```

CHAPTER

FOUR

BASICS

The first chapters shows the basics of *Pymox* and the features you'll most likely to use.

4.1 Getting Started

4.1.1 What

Pymox is mocking on steroids. It's a powerful mock object framework for Python, providing many tools to help with your unit tests, so you can write them in an easy, quick and intuitive way.

4.1.2 Why

Why `Pymox`? Python already has batteries included. It has its own `mock` library, widely used in Python applications.

So, why `pytest`, given we have Python's `unittest`? Why `arrow` or `pendulum` given we have Python's `datetime`? Why `X`, given we have Python's `Y`?

You got it !

Coming Soon

- Async support
- Decorator
- Ignore args
- ~~String imports~~

4.1.3 How

Install

```
pip install pymox
```

4.1.4 Cool Stuff

New Elegant Way

```
# conftest.py
pytest_plugins = ("mox.testing.pytest_mox",)


# test.py
from mox import expect, stubout


class TestOs:
     def test_getcwd(self):
        with stubout(os, 'getcwd') as m_getcwd, expect:
            m_getcwd.to_be.called_with().and_return('/mox/path')

        assert os.getcwd() == '/mox/path'
        mox.verify(m_getcwd)
```

If you want to be less verbose:

```
class TestOs:
     def test_getcwd(self):
        with stubout(os, 'getcwd') as m_getcwd:
            m_getcwd().returns('/mox/path')

        assert os.getcwd() == '/mox/path'
        mox.verify(m_getcwd)
```

Anything you put inside the context manager is a call expectation, so to not expect any call you can:

```
class TestOs:
     def test_getcwd(self):
        with stubout(os, 'getcwd') as m_getcwd:
            pass

        # will raise a UnexpectedMethodCallError
        assert os.getcwd() == '/mox/path'
        mox.verify(m_getcwd)
```

Dict Access

```
class TestDict:
    def test_dict_access(self):
        config = {'env': 'dev', 'reload': True}

        # doing in another way using create, but you can do with stubout too
        mock_config = mox.create(config)

        mock_config['env'].returns('prod')
        mock_config['reload'].returns(False)
```

(continues on next page)

(continued from previous page)

```
        mox.replay(mock_config)
        assert mock_config['env'] == 'prod'
        assert mock_config['reload'] is False
        mox.verify(mock_config)
```

Comparators

```
class Client:
    def get(self, url, params):
        return requests.get(url, params)


class Service:
    def get_contacts(self):
        url = 'https://my.reallylong.service/api/v1/contacts/'
        params = {'added': '7days', 'order_by': '-created'}
        return Client().get(url, params)


class TestSevice:
    def test_get_contacts_comparators_str_and_key_value(self):
        with stubout(Client, 'get') as m_get:
            url = mox.str_contains('/api/v1/contacts')
            params = mox.contains_key_value('added', '7days')
            m_get(url, params).returns({})

        service = Service()
        assert service.get_contacts() == {}
        mox.verify(m_get)

    def test_get_contacts_comparators_and_func_in_is_a(self):
        with stubout(Client, 'get') as m_get:
            url = mox.func(lambda v: str.startswith(v, 'https://my.reallylong.service/'))
            params = mox.and_(
                mox.is_a(dict),
                mox.in_('added'),
            )
            m_get(url, params).returns({})

        service = Service()
        assert service.get_contacts() == {}
        mox.verify(m_get)

    def test_get_contacts_comparators_ignore_arg_not(self):
        with stubout(Client, 'get') as m_get:
            url = mox.ignore_arg()
            params = mox.not_(mox.is_(None))
            m_get(url, params).returns({})
```

(continues on next page)

(continued from previous page)

```
        service = Service()
        assert service.get_contacts() == {}
        mox.verify(m_get)
```

Other comparators: `contains_attribute_value`, `in_`, `is_`, `is_almost`, `or_`, `same_elements_as`, `regex`

And Raises

```
class TestOs:
    def test_getcwd(self):
        with stubout(os, 'getcwd') as m_getcwd:
            # .and_raise(..) also works
            os.getcwd().raises(Exception('error'))

        with pytest.raises(Exception, match='error'):
            os.getcwd()
        mox.verify(m_getcwd)
```

Multiple Times

```
class TestOs:
    def test_getcwd(self):
        with stubout(os, 'getcwd') as m_getcwd:
            m_getcwd().returns('/mox/path')
            # the second call will return a different value
            m_getcwd().returns('/mox/another/path')
            # the three subsequent calls will return "/"
            # if no argument is passed, multiple_times doesn't limit the number of calls
            m_getcwd().multiple_times(3).returns('/')

        assert os.getcwd() == '/mox/path'
        assert os.getcwd() == '/mox/another/path'
        mox.verify(m_getcwd)
```

Any order

If you stub out multiple, the order os calls is enforced, unless you use `any_order`

```
class TestOs:
    def test_getcwd(self):
        with stubout.many([os, 'getcwd'], [os, 'cpu_count']) as (m_getcwd, m_cpu_count):
            m_getcwd().returns('/mox/path')
            m_cpu_count().returns('10')

        # will raise a UnexpectedMethodCallError
        assert os.cpu_count() == '10'
        assert os.getcwd() == '/mox/path'
```

(continues on next page)

(continued from previous page)

```
        mox.verify(m_getcwd, m_cpu_count)

    def test_getcwd_anyorder(self):
        with stubout.many([os, 'getcwd'], [os, 'cpu_count']) as (m_getcwd, m_cpu_count):
            m_getcwd().any_order().returns('/mox/path')
            m_cpu_count().any_order().returns('10')

        assert os.cpu_count() == '10'
        assert os.getcwd() == '/mox/path'
        mox.verify(m_getcwd, m_cpu_count)
```

Remember/Value

The Remember and Value are comparators, but they deserve their own section. They can be useful to retrieve some values from deeper levels of your codebase, and bring to the test for comparison. Let's see an example:

```
class Handler:
    def modify(self, d):
        # any integer key less than 5 is removed from the dict
        keys_to_remove = [key for key in d if isinstance(key, int) and key < 5]
        for key in keys_to_remove:
            del d[key]
        return d

    def send(self, d):
        return d


class Manager:
    def __init__(self, handlers=None):
        self.handlers = handlers or []

    def process(self, d):
        for handler in self.handlers:
            modified = handler.modify(d)
            handler.send(modified)


class TestManager(mox.MoxTestBase):
    def test_manager_process(self):
        mydict = {1: "apple", 4: "banana", 6: {2: 3, 4: {1: "orange", 7: 8}}, 8: 3}
        myvalue = mox.value()

        with mox.stubout(Handler, 'send') as mock_send:
            # so we use remember in the send call, and its value then the function is
            # called will go to `myvalue`
            mock_send(mox.remember(myvalue))

        Manager([Handler()]).process(mydict)
        mox.verify(mock_send)
```

(continues on next page)

(continued from previous page)

```
        # now we can compare myvalue with what we think its value must be
        assert myvalue == {6: {2: 3, 4: {1: 'orange', 7: 8}}, 8: 3}
```

Classic Way

```
import mox
import os

class TestOs:
    def test_getcwd(self):
        m = mox.Mox()

        m.stubout(os, 'getcwd')
        # calls
        os.getcwd().returns('/mox/path')

        m.replay_all()
        assert os.getcwd() == '/mox/path'
        m.verify_all()


if __name__ == '__main__':
    import unittest
    unittest.main()
```

Jurassic Way

```
import mox
import os


class TestOs(mox.MoxTestBase):
    def test_getcwd(self):
        self.mox.StubOutWithMock(os, 'getcwd')
        # calls
        os.getcwd().AndReturn('/mox/path')

        self.mox.ReplayAll()
        self.assertEqual(os.getcwd(), '/mox/path')
        self.mox.VerifyAll()


if __name__ == '__main__':
    import unittest
    unittest.main()
```

4.2 Install

```
pip install pymox
```

CHAPTER

FIVE

GUIDES

The following chapters give a broad explanation about *Pymox* features and how to use them.

5.1 Tutorial [0]

5.1.1 Introduction

Mox is a mock object framework for Python.

Mox is based on EasyMock, a Java mock object framework.

Mox will make mock objects for you, so you don't have to create your own! It mocks the public/protected interfaces of Python objects. You set up your mock object's expected behavior using a domain-specific language (DSL), which makes your tests easy to understand and refactor!

5.1.2 Core concepts

When you create a mock object, it is in record mode. You record the behavior you expect by calling the expected methods on the mock object. Once you have recorded the expected behavior, you switch the mock into replay mode. Then you start your actual test code. After your testing is done (usually along with other assertions) you verify that all recorded (expected) interactions occurred.

Mox is strict. That means that it expects the methods to be called in the order they are recorded, on a per-mock level. This is helpful to ensure that a database connection is not closed before it is used, etc.

Workflow overview

Create mock (in record mode) Set up expectations Put mock into replay mode Run test Verify expected interactions with the mock occurred

5.1.3 Basic Usage

```
import unittest import mox

class DaoUnitTest(unittest.TestCase):

    def setUp(self):
        # Create an instance of Mox
        self.person_mocker = mox.Mox()

    def testUsingMox(self):
        # Create a mock PersonDao
        dao = self.person_mocker.CreateMock(PersonDao)

        # Return a value when this method is called
        dao.InsertPerson(test_person).AndReturn(test_primary_key)

        # Void method
        dao.UpdatePerson(test_person)

        # Raise an exception when this is called
        dao.DeletePerson(unknown_person).AndRaise(UnknownPersonError('id not found'))

        # Put all mocks created by mox into replay mode
        self.person_mocker.ReplayAll()

        # Run the test
        ret_pk = dao.InsertPerson(test_person)
        dao.UpdatePerson(test_person)
        self.assertRaises(UnknownPersonError, dao, unknown_person)

        # Verify all mocks were used as expected, and tests ran properly
        self.person_mocker.VerifyAll()
        self.assertEquals(test_primary_key, ret_pk)
```

Or to create a single mock object directly without the Mox() factory:

```
dao = mox.MockObject(PersonDao)
dao.InsertPerson(test_person).AndReturn(test_primary_key)
dao.UpdatePerson(test_person)
dao.DeletePerson(unknown_person).AndRaise(UnknownPersonError('id not found'))
mox.Replay(dao)
...
mox.Verify(dao)
```

Optionally, you can make your test case be a subclass of mox.MoxTestBase; this will automatically create a mock object factory in self.mox, and will automatically verify all mock objects and unset stubs at the end of each test.

5.1.4 Unexpected Behavior

Occasionally, snakes on a plane happen: some code uses your mock object in a way you weren't expecting it to. Fortunately for everyone involved, Mox will let you know exactly what unexpected event happened.

Unexpected method call

```
dao = self.person_mocker.CreateMock(PersonDao)
dao.InsertPerson(test_person).AndReturn(test_primary_key)
self.person_mocker.ReplayAll()
ret_pk = dao.InsertPerson(other_person)
self.person_mocker.VerifyAll()
raises Unexpected method call: InsertPerson(other_person) -> None.
Expecting: InsertPerson(test_person) -> test_primary_key
```

Unknown method call

```
dao = self.person_mocker.CreateMock(PersonDao)
dao.InsertPersonZ(test_person).AndReturn(test_primary_key)
self.person_mocker.ReplayAll()
raises Method called is not a member of the object: dao.InsertPersonZ(test_person)
```

Expected method call (but it didn't happen)

```
dao = self.person_mocker.CreateMock(PersonDao)
dao.InsertPerson(test_person).AndReturn(test_primary_key)
self.person_mocker.ReplayAll()
self.person_mocker.VerifyAll()
raises Verify: Expected methods never called: 0. InsertPerson(test_person) -> test_
↪primary_key
```

Threading issues

Mock objects created by Mox are not thread-safe. If you are replaying mocks in a multi-threaded environment, please guard the mocks via mutex.

Specifically, the code to validate that the current call matches the recorded call can result in a race condition.

Hopefully soon there will be an option to make the mocks thread safe!

5.1.5 Advanced Usage

Believe it or not, there are other features as well!

In Any Order

Unfortunately, there are some things that are non-deterministic, such as iterating over the keys of a dictionary. For these cases, you'll want to group your un-ordered method calls together. This creates a group of method calls that are unordered with respect to each other, but ordered with respect to other expectations. For example:

```
dao.OpenConnection()
dao.Call(1).InAnyOrder().AndReturn('one')
dao.Call(2).InAnyOrder().AndReturn('two')
dao.Call(3).InAnyOrder().AndReturn('three')
dao.CloseConnection()
mox.Replay(mock)
```

The Call methods can occur in any order, but they must all occur after OpenConnection and before CloseConnection.

It is also possible to have two consecutive groups of InAnyOrder. In order to differentiate between the two groups, you would give names to one or both of the groups.

```
dao.OpenConnection()
dao.Foo(1).InAnyOrder('foo').AndReturn('one')
dao.Foo(2).InAnyOrder('foo').AndReturn('two')
dao.Foo(3).InAnyOrder('foo').AndReturn('three')
dao.Bar('one').InAnyOrder('foo').AndReturn(1)
dao.Bar('two').InAnyOrder('bar').AndReturn(2)
dao.Bar('three').InAnyOrder('baz').AndReturn(3)
dao.CloseConnection()
mox.Replay(mock)
```

The Foo calls can still occur in any order, but they must all occur before the unordered Bar calls occur.

Stub Out

Often, the class you're testing has one method that delegates to a lot of other complex methods. The delegation logic can be complicated, so you only want to test that, without having to record expectations for all of the work done by the submethods. For example:

:

```
class MyRequestHandler(object):

    def HandleRequest(self, request):
        if request.IsExternal():
            self.Authenticate(request)
            self.Authorize(request)
            self.Process(request)
        else:
            self.ProcessInternal(request)
```

Here, Authenticate, Authorize and Process are all expensive, and have tons of logic in them. You don't really want or need to test what they do; you just need to test that they're called. But the MyRequestHandler isn't a mock object here: it's the actual object you're testing. So what do you do...?! Use StubOutWithMock!

```
handler = MyRequestHandler()
m = mox.Mox()
m.StubOutWithMock(handler, "Authenticate")
m.StubOutWithMock(handler, "Authorize")
m.StubOutWithMock(handler, "Process")
handler.Authenticate(IsA(Request))
handler.Authorize(IsA(Request))
handler.Process(IsA(Request))
m.ReplayAll()

handler.HandleRequest(request)

m.UnsetStubs()
m.VerifyAll()
```

Note: If UnsetStubs() was called after Verify() and Verify() raises an exception because it fails then the rest of your tests may end up in a strange state. You should either call it before Verify() or – even better – call it in tearDown() which gets executed regardless of whether Verify() fails or succeeds. (If you use mox.MoxTestBase, this is taken care of for you.)

Comparators

If you aren't able to pass a argument which is equal (according to __eq__) to the expected argument when you're recording mock behavior, you probably want to use a Comparator.

- IsA(class) – Check if the parameter is an instance of the given class dao.InsertUser(IsA(Person))
- StrContains(string) - Check if the parameter contains the given substring dao.RunSql(StrContains('WHERE id=%d' % expected_id))
- Regex(pattern [, flags]) - Check if the parameter matches the given regular expression dao.RunSql(Regex(r'WHERE.*s+id=%d' % expected_id, flags=re.IGNORECASE))
- In(value) - Check if the parameter (list, tuple, or dict) contains the given value dao.BulkInsert(In(test_person))
- ContainsKeyValue(key, value) - Check if the parameter contains the given key/value pair dao.BulkInsert(ContainsKeyValue(test_id, test_person))
- Func(callable) - Validate the parameter with the given callable. This can be used for more complex checking. The callable must take 1 argument and return a bool. dao.InsertAuditRecord(Func(IsValidAudit))
- IsAlmost(value [, places]) - Check if the parameter is equal to a given value up to a certain number of decimal places. Useful for floating point numbers. dao.AddInterestToAccount(IsAlmost(0.05))
- SameElementsAs(sequence) - Check if the sequence returned has the same elements as the given sequence. Useful for lists that may be generated with non-deterministic order. dao.ProcessUsers(SameElementsAs([person1, person2]))
- IgnoreArg() - Ignore an argument. Check first and third arguments; but ignore 2nd argument. dao.UpdateUser(3, IgnoreArg(), 'admin')
- And() and Or(): combine comparators. These both take a variable number of comparators. dao.BulkInsert(And(In(test_person), IsA(list)))

You can write your own comparators. It's easy!

MockAnything

Some classes do not provide public interfaces; for example, they might use __getattribute__ to dynamically create their interface. For these classes, you can use a MockAnything. It does not enforce any interface, so any call your heart desires is valid. It works in the same record-replay-verify paradigm. Don't use this unless you absolutely have to! You can create a MockAnything with the CreateMockAnything method of your Mox instance like so: m = mox.Mox() mock = m.CreateMockAnything() mock.AnyMethod()

You may also create a MockAnything instance directly, but then you must call mox.Replay() and mox.Verify() on it, instead of using the Mox factory methods.

```
mock = mox.MockAnything() mock.AnyMethod() mox.Replay(mock)

mock.AnyMethod()

mox.Verify(mock)
```

Attributes

Some classes automatically create attributes on creation. If you stub out a class, then these attributes will not be created. You have to define these attributes in your MockObject on your mock setup.

```
m = mox.Mox()

fake_axis = m.CreateMock(MyAxis)

fake_chart = m.CreateMock(MyChart) fake_chart.axis = fake_axis
```

Mock a class

You may have code that doesn't use dependency injection, and just creates objects directly. You may also want to mock those objects. Thankfully this is possible with Mox.

For example, to stub out the foo.bar module which contains the Baz class that your code creates directly:

Mock out the class using Mox. self.mox.StubOutClassWithMocks(foo.bar.Baz) # Record that the creation of Baz should return a mock baz. mock_baz = foo.bar.Baz()

Side Effects

Sometimes the behavior of the code you are testing is dependent on some side effect of the object you are mocking. Some examples of when this is the case are when real object might treat some object as an "out" or "in/out" parameter or the real object is meant to change some shared resource that modifies the behavior of your testing unit. It is possible to simulate these side effects by using WithSideEffects.

```
# This function will be passed to WithSideEffects; when
# GetWaitingMessages is called on the mock, this function will be
# called with the same arguments as GetWaitingMessages.

def add_messages(message_list):
    message_list += ['message 1', 'message 2']
    message_appender = mox.MockObject(PendingMessages)
```

(continues on next page)

(continued from previous page)

```
    message_appender.GetWaitingMessages(
        ['message 0']).WithSideEffects(add_messages).AndReturn(2)

    mox.Replay(message_appender)
    messages = ['message 0']
    new_messages = message_appender.GetWaitingMessages(messages)
    mox.Verify(message_appender)

    assertEquals(['message 0', 'message 1', 'message 2'], messages)
```

Callbacks

Mocking a callback should be pretty straight forward.

```
m = mox.Mox()
mock_callback = m.CreateMockAnything() # MockAnything is callable
test_object.SetCallback(mock_callback)
mock_callback(42) # Expect this to be called.
m.ReplayAll()
test_object.DoStuff() # Which in turn calls mock_callback... m.VerifyAll()
```

Misc

I've seen code that likes to access class variables through instances, so I've added support for this.

```
print 'this is silly, but it happens:', mock_obj.MY_CLASS_VARIABLE
```

There is support for comparing mock objects. This could be helpful for testing that your mock got injected into the proper places:

```
dao.set_db(mock_db)
self.assertEquals(mock_db, dao._MyDAO__db)
```

I've also seen code that likes to verify if an object is false, for example:

```
def myMethod(self, foo, bar=None):
    if not bar:
        # use internal default
```

To deal with this, you can make your mock expect __nonzero__, so you can safely inject your mock into this object. Hurray!

5.1.6 Examples

Basic Example

Let's say you have this class, and you'd like to test it:

```
class PersonManager(object):

    def init(self, person_dao): self._dao = person_dao

    def CreatePerson(self, person, user): """Create a Person"""

    if user != 'stevepm':
        raise Exception('no way, jose')

    try:
        self._dao.InsertPerson(person)
    except PersistenceException, e:
        raise BusinessException('error talking to db', e)
```

And you have the class PersonManager depends on:

```
class PersonDao(object):

    def init(self, db): self._db = db

    def InsertPerson(self, person):
        self._db.Execute('INSERT INTO Person(name) VALUES ("%s")' % person)
```

So now you can write the test:

```
class PersonManagerTest(unittest.TestCase):

    def setUp(self):
        self.mox = mox.Mox()
        self.dao = self.mox.CreateMock(PersonDao)
        self.manager = PersonManager(self.dao)

    def testCreatePersonWithAccess(self):
        self.dao.InsertPerson(test_person)
        self.mox.ReplayAll()
        self.manager.CreatePerson(test_person, 'stevepm')
        self.mox.VerifyAll()

    def testCreatePersonWithDbException(self):
        self.dao.InsertPerson(test_person).AndRaise(
            PersistenceException('Snakes!'))
        self.mox.ReplayAll()
        self.assertRaises(
            BusinessException, self.manager.CreatePerson, test_person, 'stevepm')
        self.mox.VerifyAll()
```

Pretty cool, huh?

Extending The Basic Example

Now let's say you want to have your DAO return the new primary key for the person, and your manager class would like to verify that the primary key is greater than some number. Who knows, it's a toy example! :) You would change your code as follows:

```
def CreatePerson(self, person, user):
    """Creates a Person."""
    if user != 'stevepm':
        raise Exception('no way, jose')

    try:
        primary_key = self._dao.InsertPerson(person)
    except PersistenceException e:
        raise BusinessException('error talking to db', e)

    if primary_key < MIN_PRIMARY_KEY_VALUE:
        self._dao.DeletePerson(primary_key)
        raise BusinessException('primary key too small')

def InsertPerson(self, person):
    return db.Execute('INSERT INTO Person(name) VALUES ("%s")' % person)

def DeletePerson(self, person_id):
    db.Execute('DELETE FROM Person WHERE ...' % person_id)
```

Now you can modify your test:

```
def testCreatePersonWithAccess(self):
    self.dao.InsertPerson(test_person).AndReturn(HUGE_PRIMARY_KEY)
    self.mox.ReplayAll()
    self.manager.CreatePerson(test_person, 'stevepm')
    self.mox.VerifyAll()
```

And add the new test:

```
def testCreatePersonWithSmallPrimaryKey(self):
self.dao.InsertPerson(test_person).AndReturn(TINY_PRIMARY_KEY)
self.dao.DeletePerson(TINY_PRIMARY_KEY)
self.mox.ReplayAll()
self.assertRaises(
    BuisnessException, self.manager.CreatePerson, test_person, 'stevepm')
self.mox.VerifyAll()
```

Complicating Things Even More...

Ugh, now let's say it is up to your manager to pass some audit trail object to the DAO, which the DAO handles appropriately. Let's not worry about the impl, since we're really just dealing with public interfaces. The new DAO interface is:

```
def InsertPerson(self, person, audit_trail_obj):
```

And the manager now looks like this:

```
def CreatePerson(self, person, user):
    """Create a Person"""
```

(continues on next page)

(continued from previous page)

```
if user != 'stevepm':
    raise Exception('no way, jose')

    audit_record = AuditRecord(user)

    try:
        primary_key = self._dao.InsertPerson(person, audit_record)
    except PersistenceException e:
        raise BusinessException('error talking to db', e)

    if primary_key < MIN_PRIMARY_KEY_VALUE:
        self._dao.DeletePerson(primary_key)
        raise BusinessException('primary key too small')
```

Oh now, how do we setup our expected call to dao.InsertPerson now that a parameter is out of our control?! Have no fear, Mox is here! There are Comparators that can be used to check the equivalency of method parameters. You can even mix and match then with real parameters, as you'll see below.

def testCreatePersonWithAccess(self): self.dao.InsertPerson(test_person, IsA(AuditRecord)).AndReturn(HUGE_PRIMARY_KEY) self.mox.ReplayAll() self.manager.CreatePerson(test_person, 'stevepm') self.mox.VerifyAll()

There are all kinds of other comparators for simple parameter checking. If you have complex logic to check the value, you can even use a callable to verify it.

```
def testCreatePersonWithAccess(self):
    self.dao.InsertPerson(
        test_person, Func(ValidAuditRecord)).AndReturn(HUGE_PRIMARY_KEY)
    self.mox.ReplayAll()
    self.manager.CreatePerson(test_person, 'stevepm')
    self.mox.VerifyAll()

def ValidAuditRecord(audit_record):
    return (audit_record.user() == 'stevepm' and audit_record.type() == 'insert')
```

Introduction

Here we provide some common Mox recipes that people have found useful. This section's constantly under development; if you find a better way of implementing a recipe below, please let us know in the comments.

Set up your Mox test classes in a sane way

Note: many other recipes assume you've done this.

```
def setUp(self):
    self.mox = mox.Mox()

def tearDown(self):
    self.mox.UnsetStubs()
```

Stub out a method called from a constructor in the same class

```
TODO: Write a public example here.
```

Stub out a static method in the class under test

```
def testFoo(self):

    orig_method = module.class.StaticMethod

    static_stub = staticmethod(lambda *args, **kwargs: None)
    module.class.StaticMethod = static_stub

    self.mox.ReplayAll()

    ...

    self.mox.VerifyAll()

    module.class.StaticMethod = orig_method
```

Mock a module-level function in a different module

```
def testFoo(self):

self.mox.StubOutWithMock(module_to_mock, 'FunctionToMock')
module_to_mock.FunctionToMock().AndReturn(foo)

self.mox.ReplayAll()
...
self.mox.VerifyAll()
```

Stub out a class in a different module

```
TODO: Write a public example here.
```

Mock a method in the class under test.

TODO: Investigate this further. Maybe stubbing out call would help?

```
def testFoo(self):

    # Note the difference: we instantiate the class *before* Replaying.
    foo_instance = module_under_test.ClassUnderTest()
    self.mox.StubOutWithMock(foo_instance, 'MethodToStub')
    foo_instance.MethodToStub().AndReturn('foo')

    ...

    self.mox.ReplayAll()
    ...
    self.mox.VerifyAll()
```

Mock a generator in the class under test

```
def testFoo(self):

    ...

    foo_instance = module_under_test.ClassUnderTest()
```

(continues on next page)

(continued from previous page)

```
    self.mox.StubOutWithMock(foo_instance, 'GeneratorToStub')

    mygen = (x for x in [1, 2, 3])
    foo_instance.MethodToStub(mox.IsA(object)).AndReturn(mygen)

    ...

    self.mox.ReplayAll()

    ...

    self.mox.VerifyAll()
    ```
```

Mocking datetime.datetime.now

```
import datetime import mox

m = mox.Mox()

Stub out the datatime.datetime class.

m.StubOutWithMock(datetime, 'datetime')

Record a call to 'now', and have it return the value '1234'

datetime.datetime.now().AndReturn(1234)

Set the mocks to replay mode

m.ReplayAll()

This will return '1234'

datetime.datetime.now()

Verify the time was actually checked.

m.VerifyAll()
```

Return datetime.datetime to its default (non-mock) state.

```
m.UnsetStubs()
```

Alternatively, rewrite your code so that you can mock out datetime.now without Mox:

```
def FunctionBeingTested(now=datetime.datetime.now): DoSomethingWith(now())
```

in test code

```
def MyNow(): return 1234 FunctionBeingTested(now=MyNow)
```
```

5.2 Tutorial [1]

5.2.1 Basics

As said before, with Pymox you should set expectations and then enter in replay mode. Here is a basic example:

```
class Duck:
    def quack(self, times=1):
        return ['quack'] * times

    def walk(self):
        return ['walking']

    def walk_and_quack(self, times=1):
        return self.walk() + self.quack(times=times)
```

Here is a Duck class. Let's play with our and Pymox!

```
import mox


class TestDuck:

    def test_quack(self):
        m = mox.Mox()
        m_duck = m.CreateMock(Duck)

        # expects quack to be called with `times=1`
        m_duck.quack(times=1).returns(['new quack'])

        m.replay_all()
        assert m_duck.quack(times=1) == ['new quack']
        m.verify_all()
```

Let's change the test a little bit:

```
    # [...]
    def test_quack_2(self):
        m = mox.Mox()
        m_duck = m.CreateMock(Duck)

        # expects quack to be called with `times=1`
        m_duck.quack(times=1).returns(['new quack'])

        m.replay_all()
        assert m_duck.walk() == ['walking']
        assert m_duck.quack(times=1) == ['new quack']
        m.verify_all()
```

The test above will fail with the following error:

```
    E           mox.mox.UnexpectedMethodCallError: Unexpected method call.  unexpected:-␣
↪ expected:+
```

(continues on next page)

(continued from previous page)

```
E           - Duck.walk() -> None
E           + Duck.quack(times=1) -> ['new quack']
```

Since you expected quack to be called and walk was called instead. You can add an expectation for walk:

```
def test_quack_3(self):
    m = mox.Mox()
    m_duck = m.CreateMock(Duck)

    # expects quack to be called with `times=1`
    m_duck.quack(times=1).returns(['new quack'])
    m_duck.walk().returns(['pretending to be walking'])

    m.replay_all()
    assert m_duck.quack(times=1) == ['new quack']
    assert m_duck.walk() == ['pretending to be walking']
    m.verify_all()
```

You can also stub out quack method only and mox won't care about the other methods:

```
def test_quack_4(self):
    m = mox.Mox()
    duck = Duck()

    m.stubout(duck, 'quack')
    """
    You can also do with the class:
    m.stubout(Duck, 'quack')
    """

    # expects quack to be called with `times=1`
    duck.quack(times=1).returns(['new quack'])

    m.replay_all()
    assert duck.quack(times=1) == ['new quack']
    assert duck.walk() == ['walking']
    m.verify_all()
```

The order matters, so if you do:

```
def test_quack_5(self):
    m = mox.Mox()
    m_duck = m.CreateMock(Duck)

    # expects quack to be called with `times=1`
    m_duck.quack(times=1).returns(['new quack'])
    m_duck.walk().returns(['pretending to be walking'])

    m.replay_all()
    assert m_duck.walk() == ['pretending to be walking']
    assert m_duck.quack(times=1) == ['new quack']
    m.verify_all()
```

It fails with:

```
    E           mox.mox.UnexpectedMethodCallError: Unexpected method call.  unexpected:-␣
↪ expected:+
    E           - Duck.walk() -> None
    E           + Duck.quack(times=1) -> ['new quack']
```

To fix that you can use `any_order()`:

```
    def test_quack_6(self):
        m = mox.Mox()
        m_duck = m.CreateMock(Duck)

        # expects quack to be called with `times=1`
        m_duck.quack(times=1).any_order().returns(['new quack'])
        m_duck.walk().any_order().returns(['pretending to be walking'])

        m.replay_all()
        assert m_duck.walk() == ['pretending to be walking']
        assert m_duck.quack(times=1) == ['new quack']
```

5.2.2 Comparators

You can use comparators when you are unsure of the arguments of a method call.

```
    def test_quack_7(self):
        m = mox.Mox()
        duck = Duck()

        m.stubout(Duck, 'quack')

        def validate_arg(arg):
         if arg in [1, 2, 3]:
          return True
         return False

        duck.quack(times=mox.is_a(int)).returns(['new quack'])
        duck.quack(times=mox.not_(mox.is_(4))).returns(['new quack'])
        duck.quack(times=mox.func(validate_arg)).returns(['new quack'])
        duck.quack(times=mox.or_(mox.Is(1), mox.is_(2), mox.is_(3))).returns(['new quack
↪'])

        duck.quack(times=mox.ignore_arg()).returns(['new quack'])
        duck.quack(times=mox.is_almost(1.00003, places=4)).returns(['new quack'])

        m.replay_all()
        assert duck.quack(times=random.choice([1, 2, 3])) == ['new quack']
        assert duck.quack(times=random.choice([1, 2, 3])) == mox.in_('new quack')
        assert duck.quack(times=random.choice([1, 2, 3]))[0] == mox.str_contains('quack')
        assert duck.quack(times=random.choice([1, 2, 3])) == mox.same_elements_as({'new␣
↪quack'})
```

(continues on next page)

(continued from previous page)

```
        assert duck.quack(times=random.choice([1, 2, 3])) == ['new quack']
        assert duck.quack(times=1) == ['new quack']
        m.verify_all()
```

All the assertions for the test above should pass. There are other cool comparators, like: and, `contains_attribute_value`, `contains_key_value`.

For more comparators, see: https://pymox.readthedocs.io/en/latest/reference.html#comparators

5.2.3 Remember

It's possible to also remember a value that might be changed in your code. See the test below:

```
    def test_quack_8(self):

        class StopQuackingDuck:

            def _do_quack(self, choices=None):
                return choices

            def quack(self, choices=[], less=False):
                if less:
                    choices.pop()
                self._do_quack(choices=choices)

        m = mox.Mox()
        duck = StopQuackingDuck()

        m.stubout(StopQuackingDuck, '_do_quack')

        choices_1 = mox.value()
        choices_2 = mox.value()
        duck._do_quack(choices=mox.remember(choices_1))
        duck._do_quack(choices=mox.remember(choices_2))
        duck._do_quack(choices=mox.remember(choices_2))
        duck._do_quack(choices=mox.remember(choices_2))

        all_choices = ['quack', 'new quack', 'newest quack']

        m.replay_all()
        duck.quack(all_choices, less=False)
        assert choices_1 == ['quack', 'new quack', 'newest quack']

        duck.quack(all_choices, less=True)
        assert choices_2 == ['quack', 'new quack']

        duck.quack(all_choices, less=True)
        assert choices_2 == ['quack']

        duck.quack(all_choices, less=True)
        assert choices_2 == []
        m.verify_all()
```

5.2.4 Other

You can also make a method return a different value the second time it's called:

```
    def test_walk_and_quack_0(self):
        m = mox.Mox()
        duck = Duck()

        m.stubout(Duck, 'quack')

        duck.quack(times=1).returns(['new quack'])
        duck.quack(times=1).returns(['newest quack'])

        m.replay_all()
        assert duck.walk_and_quack() == ['walking', 'new quack']
```

But since we didn't use m.verify_all(), it didn't require the second call to happen. Let's add the verify and see what happens:

```
    def test_walk_and_quack_1(self):
        m = mox.Mox()
        duck = Duck()

        m.stubout(Duck, 'quack')

        duck.quack(times=1).returns(['new quack'])
        duck.quack(times=1).returns(['newest quack'])

        m.replay_all()
        assert duck.walk_and_quack() == ['walking', 'new quack']
        m.verify_all()
```

It fails with:

```
    E           mox.mox.ExpectedMethodCallsError: Verify: Expected methods never called:
    E             0.  Duck.quack.__call__(times=1) -> ['newest quack']
```

Let's fix it by adding a second call:

```
    def test_walk_and_quack_2(self):
        m = mox.Mox()
        duck = Duck()

        m.stubout(Duck, 'quack')

        duck.quack(times=1).returns(['new quack'])
        duck.quack(times=1).returns(['newest quack'])

        m.replay_all()
        assert duck.walk_and_quack() == ['walking', 'new quack']
        assert duck.walk_and_quack() == ['walking', 'new quack']
        m.verify_all()
```

Now you get the following error, since in the second time it returns ['newest quack'].

```
    E       AssertionError: assert ['walking', 'newest quack'] == ['walking', 'new quack
↪']
    E         At index 1 diff: 'newest quack' != 'new quack'
    E         Full diff:
    E         - ['walking', 'new quack']
    E         + ['walking', 'newest quack']
    E         ?                 +++
```

Let's fix it:

```
    def test_walk_and_quack_3(self):
        m = mox.Mox()
        duck = Duck()

        m.stubout(Duck, 'quack')

        duck.quack(times=1).returns(['new quack'])
        duck.quack(times=1).returns(['newest quack'])

        m.replay_all()
        assert duck.walk_and_quack() == ['walking', 'new quack']
        assert duck.walk_and_quack() == ['walking', 'newest quack']
        m.verify_all()
```

Let's now see how we can mock and assert calls in the context of a loop:

```
    def test_walk_and_quack_4(self):
        m = mox.Mox()
        duck = Duck()

        m.stubout(Duck, 'quack')

        duck.quack(times=1).returns(['new quack'])

        m.replay_all()
        assert duck.walk() == ['walking']
        for _ in range(3):
            assert duck.walk_and_quack() == ['walking', 'new quack']
        m.verify_all()
```

If you run the test above, you get the following:

```
    E           mox.mox.UnexpectedMethodCallError: Unexpected method call Duck.quack.__
↪call__(times=1) -> None
```

Let's fix by using the `multiple_times` group.

```
    def test_walk_and_quack_5(self):
        m = mox.Mox()
        duck = Duck()

        m.stubout(Duck, 'quack')

```

(continues on next page)

(continued from previous page)

```
        duck.quack(times=1).multiple_times().returns(['new quack'])

        m.replay_all()
        assert duck.walk() == ['walking']
        for _ in range(3):
            assert duck.walk_and_quack() == ['walking', 'new quack']
        m.verify_all()
```

If you know exactly how many calls are made, you can add an argument: `.multiple_times(3)`.

CHAPTER

SIX

PYMOX IN PRACTICE

The following chapters deal with considerations of using *Pymox* in the real world.

6.1 Recipes

6.1.1 Example 1

From project python-keystoneclient, under directory tests, in source file utils.py.

```
def setUp(self):
        super(TestCase, self).setUp()
        self.mox = mox.Mox()
        self._original_time = time.time
        time.time = lambda: 1234
        httplib2.Http.request = self.mox.CreateMockAnything()
        self.client = client.Client(username=self.TEST_USER,
                                    token=self.TEST_TOKEN,
                                    project_id=self.TEST_TENANT,
                                    auth_url=self.TEST_URL,
                                    endpoint=self.TEST_URL)
```

6.1.2 Example 2

From project hyou-master, under directory test, in source file util_test.py.

```
def setUp(self):
    self.mox = mox.Mox()
    self.enumerator = self.mox.CreateMockAnything()
    self.constructor = self.mox.CreateMockAnything()
    self.dict = hyou.util.LazyOrderedDictionary(
        enumerator=self.enumerator, constructor=self.constructor)
```

6.1.3 Example 3

From project hyou-master, under directory test, in source file spreadsheet_test.py.

```
def setUp(self):
    self.mox = mox.Mox()
    self.mox.StubOutClassWithMocks(hyou.client, 'Worksheet')
    self.client = self.mox.CreateMock(
        gdata.spreadsheets.client.SpreadsheetsClient)
    self.drive = self.mox.CreateMockAnything()
    entry = FakeSpreadsheetFeed('Cinamon')
    self.spreadsheet = hyou.client.Spreadsheet(
        None, self.client, self.drive, 'cinamon', entry)
```

6.1.4 Example 4

From project hyou-master, under directory test, in source file worksheet_test.py.

```
def setUp(self):
    self.mox = mox.Mox()
    self.client = self.mox.CreateMock(
        gdata.spreadsheets.client.SpreadsheetsClient)
    self.worksheet = hyou.client.Worksheet(
        FakeSpreadsheet('cinamon'),
        self.client,
        's1',
        FakeWorksheetFeed('Sheet1', '2', '5'))
```

6.1.5 Example 5

From project akanda-rug-master, under directory akanda/rug/test/unit/openvswitch, in source file test_ovs_lib.py.

```
def setUp(self):
        super(OVS_Lib_Test, self).setUp()
        self.BR_NAME = "br-int"
        self.TO = "--timeout=2"

        self.mox = mox.Mox()
        self.root_helper = 'sudo'
        self.br = ovs_lib.OVSBridge(self.BR_NAME, self.root_helper)
        self.mox.StubOutWithMock(utils, "execute")
        self.addCleanup(self.mox.UnsetStubs)
```

6.1.6 Example 6

From project interstellar-master, under directory gsutil/third_party/gcs-oauth2-boto-plugin/gcs_oauth2_boto_plugin, in source file test_oauth2_client.py.

```
def setUp(self):
    self.mox = mox.Mox()
    self.mox.StubOutClassWithMocks(httplib2, 'Http')
    self.mock_http = httplib2.Http()
```

6.1.7 Example 7

From project django-lean, under directory django_lean/experiments/tests, in source file test_tags.py.

```
def setUp(self):
        self.experiment = Experiment(name="test_experiment")
    self.experiment.save()
    self.experiment.state = Experiment.ENABLED_STATE
    self.experiment.save()

    self.other_experiment = Experiment(name="other_test_experiment")
    self.other_experiment.save()
    self.other_experiment.state = Experiment.ENABLED_STATE
    self.other_experiment.save()
    self.mox = mox.Mox()
```

6.1.8 Example 8

From project django-lean, under directory django_lean/experiments/tests, in source file test_daily_report.py.

```
def testZeroParticipantExperiment(self):
    mocker = mox.Mox()
    engagement_calculator = mocker.CreateMockAnything()
    mocker.ReplayAll()

    report_date = date.today()
    EngagementReportGenerator(engagement_score_calculator=engagement_calculator).
↪generate_daily_report_for_experiment(
        self.other_experiment, report_date)

    experiment_report = DailyEngagementReport.objects.get(
        experiment=self.other_experiment, date=report_date)

    mocker.VerifyAll()

    self.assertEquals(None, experiment_report.test_score)
    self.assertEquals(None, experiment_report.control_score)
    self.assertEquals(0, experiment_report.test_group_size)
    self.assertEquals(0, experiment_report.control_group_size)
```

6.1.9 Example 9

From project django-lean, under directory django_lean/lean_retention/tests, in source file test_reports.py.

```
def setUp(self):
        self.mox = mox.Mox()
        self.user = User.objects.create_user('user', 'user@example.com', 'user')
        self.activity, _ = DailyActivity.objects.stamp(user=self.user,
                                                       site=get_current_site(),
                                                       medium='Default')
        self.activity.days = 29
        self.activity.save()
```

6.1.10 Example 10

From project protobuf-objc, under directory python/google/protobuf/internal, in source file decoder_test.py.

```
def setUp(self):
    self.mox = mox.Mox()
    self.mock_stream = self.mox.CreateMock(input_stream.InputStream)
    self.mock_message = self.mox.CreateMock(message.Message)
```

6.1.11 Example 11

From project protobuf-objc, under directory python/google/protobuf/internal, in source file encoder_test.py.

```
def setUp(self):
    self.mox = mox.Mox()
    self.encoder = encoder.Encoder()
    self.mock_stream = self.mox.CreateMock(output_stream.OutputStream)
    self.mock_message = self.mox.CreateMock(message.Message)
    self.encoder._stream = self.mock_stream
```

6.1.12 Example 12

From project Godel, under directory src/Godel/tests, in source file test_rule_engine.py.

```
def __init__(self):
        self.tag_stack = []
        self.state_stack = []
        self.stack = []
        self.hypergraph = None
        self.Groundings = {}
        self.Types = {}

        self.mox = mox.Mox()
        self.hypergraph = self.mox.CreateMockAnything()
        self.hypergraph.AnyMethod()
```

6.1.13 Example 13

From project cc, under directory nova, in source file test.py.

```
def setUp(self):
        super(TrialTestCase, self).setUp()

        # emulate some of the mox stuff, we can't use the metaclass
        # because it screws with our generators
        self.mox = mox.Mox()
        self.stubs = stubout.StubOutForTesting()
        self.flag_overrides = {}
```

6.1.14 Example 14

From project imagr-master, under directory Imagr/gmacpyutil, in source file timer_test.py.

```
def setUp(self):
    self.mox = mox.Mox()
    self.mox.StubOutWithMock(timer.gmacpyutil, 'SetPlistKey')
    self.mox.StubOutWithMock(timer.gmacpyutil, 'GetPlistKey')

    self.timeplist = '/tmp/blah/myapp.plist'
    self.interval = datetime.timedelta(hours=23)
    self.tf = timer.TimeFile(self.timeplist)
```

6.1.15 Example 15

From project imagr-master, under directory Imagr/gmacpyutil, in source file ds_test.py.

```
def setUp(self):
    self.mox = mox.Mox()
    self.mox.StubOutWithMock(ds.gmacpyutil, 'RunProcess')
    if os.uname()[0] == 'Linux':
      self.InitMockFoundation()
    elif os.uname()[0] == 'Darwin':
      self.StubFoundation()
```

6.1.16 Example 16

From project google-apputils-master, under directory tests, in source file file_util_test.py.

```
def setUp(self):
    self.mox = mox.Mox()
    self.sample_contents = 'Contents of the file'
    self.file_path = '/path/to/some/file'
    self.fd = 'a file descriptor'
```

6.1.17 Example 17

From project Android-Development, under directory tools/scripts, in source file divide_and_compress_test.py.

```
def setUp(self):
    """Prepare the test.

    Construct some mock objects for use with the tests.
    """
    self.my_mox = mox.Mox()
    file1 = BagOfParts()
    file1.filename = 'file1.txt'
    file1.contents = 'This is a test file'
    file2 = BagOfParts()
    file2.filename = 'file2.txt'
    file2.contents = ('akdjfk;djsf;kljdslkfjslkdfjlsfjkdvn;kn;2389rtu4i'
                      'tn;ghf8:89H*hp748FJw80fu9WJFpwf39pujens;fihkhjfk'
                      'sdjfljkgsc n;iself')
    self.files = {'file1': file1, 'file2': file2}
```

6.1.18 Example 18

From project nova, under directory nova/tests/xenapi, in source file test_vm_utils.py.

```
def test_lookup_call(self):
        mock = mox.Mox()
        mock.StubOutWithMock(vm_utils, 'lookup')

        vm_utils.lookup('session', 'somename').AndReturn('ignored')

        mock.ReplayAll()
        vm_utils.vm_ref_or_raise('session', 'somename')
        mock.VerifyAll()
```

6.1.19 Example 19

From project nova, under directory nova/tests/virt/xenapi/imageupload, in source file test_glance.py.

```
def setUp(self):
        super(TestGlanceStore, self).setUp()
        self.store = glance.GlanceStore()
        self.mox = mox.Mox()
```

6.1.20 Example 20

From project nova, under directory nova, in source file test.py.

```
def setUp(self):
        super(MoxStubout, self).setUp()
        # emulate some of the mox stuff, we can't use the metaclass
        # because it screws with our generators
        self.mox = mox.Mox()
        self.stubs = stubout.StubOutForTesting()
        self.addCleanup(self.mox.UnsetStubs)
        self.addCleanup(self.stubs.UnsetAll)
        self.addCleanup(self.stubs.SmartUnsetAll)
        self.addCleanup(self.mox.VerifyAll)
```

6.1.21 Example 21

From project nappingcat, under directory tests/gittests, in source file utils.py.

```
def test_uses_getlogin(self):
        settings = {
            'host':'host-%d' % random.randint(1,100),
        }
        _mox = mox.Mox()
        _mox.StubOutWithMock(os, 'getlogin')
        random_user = 'rand-%d' % random.randint(1,100)
        os.getlogin().AndReturn(random_user)
        _mox.ReplayAll()
        results = utils.get_clone_base_url(settings)
        self.assertEqual('%s@%s' % (random_user, settings['host']), results)
        _mox.UnsetStubs()
```

6.1.22 Example 22

From project appengine-python3-master, under directory google/appengine/tools/devappserver2/admin, in source file datastore_viewer_test.py.

```
def setUp(self):
    self.app_id = 'myapp'
    os.environ['APPLICATION_ID'] = self.app_id
    api_server.test_setup_stubs(app_id=self.app_id)

    self.mox = mox.Mox()
    self.mox.StubOutWithMock(admin_request_handler.AdminRequestHandler,
                             'render')
```

6.1.23 Example 23

From project appengine-python3-master, under directory google/appengine/tools/devappserver2/admin, in source file taskqueue_queues_handler_test.py.

```
def setUp(self):
    self.mox = mox.Mox()
    self.mox.StubOutWithMock(taskqueue_utils.QueueInfo, 'get')
    self.mox.StubOutWithMock(admin_request_handler.AdminRequestHandler,
                             'render')
```

6.1.24 Example 24

From project habitat, under directory habitat/tests/test_utils, in source file test_startup.py.

```
def setup(self):
        self.mocker = mox.Mox()
        self.config = copy.deepcopy(_logging_config)

        self.old_handlers = logging.root.handlers
        # nose creates its own handler
        logging.root.handlers = []

        # manual cleanup needed for check_file's tests
        self.temp_dir = None
        self.temp_files = []
```

6.1.25 Example 25

From project habitat, under directory habitat/tests/test_utils, in source file test_immortal_changes.py.

```
def setup(self):
        self.m = mox.Mox()

        self.consumer = immortal_changes.Consumer(None,
            backend='habitat.tests.test_utils.'
                    'test_immortal_changes.DummyConsumer')
        assert isinstance(self.consumer._consumer, DummyConsumer)
        self.m.StubOutWithMock(self.consumer._consumer, "wait")

        assert immortal_changes.time == time
        immortal_changes.time = DummyTimeModule()
        self.m.StubOutWithMock(immortal_changes.time, "sleep")

        self.m.StubOutWithMock(immortal_changes.logger, "exception")

        # for brevity.
        self.backend = self.consumer._consumer.wait
        self.sleep = immortal_changes.time.sleep
        self.exc = immortal_changes.logger.exception
        self.cb = self.m.CreateMockAnything()
```

6.1.26 Example 26

From project habitat, under directory habitat/tests, in source file test_parser_daemon.py.

```
def setup(self):
        self.m = mox.Mox()

        self.config = {
            "couch_uri": "http://localhost:5984", "couch_db": "test"}

        self.m.StubOutWithMock(parser_daemon, 'couchdbkit')
        self.m.StubOutWithMock(parser_daemon, 'immortal_changes')
        self.m.StubOutWithMock(parser_daemon, 'parser')
        self.mock_server = self.m.CreateMock(couchdbkit.Server)
        self.mock_db = self.m.CreateMock(couchdbkit.Database)
        parser_daemon.couchdbkit.Server("http://localhost:5984")\
                .AndReturn(self.mock_server)
        self.mock_server.__getitem__("test").AndReturn(self.mock_db)
        self.mock_db.info().AndReturn({"update_seq": 191238})
        parser_daemon.parser.Parser(self.config)

        self.m.ReplayAll()
        self.daemon = parser_daemon.ParserDaemon(self.config)
        self.m.VerifyAll()
        self.m.ResetAll()
```

6.1.27 Example 27

From project habitat, under directory habitat/tests/test_parser, in source file test_parser.py.

```
def setup(self):
        self.m = mox.Mox()
        self.mock_module = self.m.CreateMock(parser.ParserModule)

        class MockModule(parser.ParserModule):
            def __new__(cls, parser):
                return self.mock_module

        base_path = os.path.split(os.path.abspath(__file__))[0]
        cert_path = os.path.join(base_path, 'certs')
        self.parser_config = {"parser": {"modules": [
            {"name": "Mock", "class": MockModule}],
            "certs_dir": cert_path}, "loadables": [],
            "couch_uri": "http://localhost:5984", "couch_db": "test"}

        self.m.StubOutWithMock(parser, 'couchdbkit')
        self.mock_server = self.m.CreateMock(couchdbkit.Server)
        self.mock_db = self.m.CreateMock(couchdbkit.Database)
        parser.couchdbkit.Server("http://localhost:5984")\
                .AndReturn(self.mock_server)
        self.mock_server.__getitem__("test").AndReturn(self.mock_db)
```

(continues on next page)

(continued from previous page)

```
        self.m.ReplayAll()
        self.parser = parser.Parser(self.parser_config)
        self.m.VerifyAll()
        self.m.ResetAll()
```

6.1.28 Example 28

From project WMCore, under directory test/python/WMCore_t/Alerts_t/ZMQ_t/Sinks_t, in source file EmailSink_t.py.

```
def setUp(self):
        self.config = ConfigSection("email")
        self.config.fromAddr = "some@local.com"
        self.config.toAddr = ["some1@local.com", "some2@local.com"]
        self.config.smtpServer = "smtp.gov"
        self.config.smtpUser = None
        self.config.smtpPass = None

        # now we want to mock smtp emailing stuff - via pymox - no actual
        # email sending to happen
        self.mox = mox.Mox()
        self.smtpReal = EmailSinkMod.smtplib
        EmailSinkMod.smtplib = self.mox.CreateMock(EmailSinkMod.smtplib)
        self.smtp = self.mox.CreateMockAnything()
```

6.1.29 Example 29

From project WMCore, under directory test/python/WMCore_t/Storage_t/Plugins_t, in source file SRMV2Impl_t.py.

```
def setUp(self):
        self.my_mox = mox.Mox()
        self.my_mox.StubOutWithMock(moduleWeAreTesting.os.path, 'getsize')
        self.my_mox.StubOutWithMock(moduleWeAreTesting,'runCommand')
        self.my_mox.StubOutWithMock(moduleWeAreTesting,'tempfile')
        self.popenMocker = self.my_mox.CreateMock(popenMockHelper)
        self.popenBackup = moduleWeAreTesting.Popen


        self.temporaryFiles = []
        self.rules          = []
```

6.1.30 Example 30

From project panfletario, under directory r2/r2/tests/unit, in source file test_link.py.

```
def test_make_permalink(self):
        m  = mox.Mox()
        subreddit = m.CreateMock(self.models.Subreddit)
        #subreddit.name.AndReturn('stuff')
        m.ReplayAll()

        pylons.c.default_sr = True #False
        pylons.c.cname = False
        link = self.models.Link(name = 'Link Name', url = 'self', title = 'A link title',
↪ sr_id = 1)
        link._commit()
        permalink = link.make_permalink(subreddit)

        m.VerifyAll()
        assert permalink == '/lw/%s/a_link_title/' % link._id36


    # def test_make_permalink_slow(self):
    #
    #
    #     link = self.models.Link(name = 'Link Name', url = 'self', sr_id = 1)
    #     m = mox.Mox()
    #     mock_subreddit = mox.MockObject(self.models.Subreddit)
    #
    #     m.StubOutWithMock(link, 'subreddit_slow', use_mock_anything=True)
    #     link.subreddit_slow().AndReturn(mock_subreddit)
    #
    #     m.ReplayAll()
    #
    #     permalink = link.make_permalink_slow()
    #
    #     m.UnsetStubs()
    #     m.VerifyAll()
```

6.1.31 Example 31

From project hyou-master, under directory test, in source file collection_test.py.

```
def setUp(self):
    self.mox = mox.Mox()
    self.mox.StubOutClassWithMocks(hyou.client, 'Spreadsheet')
    self.client = self.mox.CreateMock(
        gdata.spreadsheets.client.SpreadsheetsClient)
    self.drive = self.mox.CreateMockAnything()
    self.collection = hyou.client.Collection(self.client, self.drive)
```

6.1.32 Example 32

From project python-typepad-api, under directory tests, in source file test_tpobject.py.

```
def test_responseless(self):
        request = {
            'uri': mox.Func(self.saver('uri')),
            'method': 'POST',
            'headers': mox.Func(self.saver('headers')),
            'body': mox.Func(self.saver('body')),
        }
        response = {
            'status': 204,  # no content
        }

        http = typepad.TypePadClient()
        typepad.client = http
        http.add_credentials(
            OAuthConsumer('consumertoken', 'consumersecret'),
            OAuthToken('tokentoken', 'tokensecret'),
            domain='api.typepad.com',
        )

        mock = mox.Mox()
        mock.StubOutWithMock(http, 'request')
        http.request(**request).AndReturn((httplib2.Response(response), ''))
        mock.ReplayAll()

        class Moose(typepad.TypePadObject):

            class Snert(typepad.TypePadObject):
                volume = typepad.fields.Field()
            snert = typepad.fields.ActionEndpoint(api_name='snert', post_type=Snert)

        moose = Moose()
        moose._location = 'https://api.typepad.com/meese/7.json'

        ret = moose.snert(volume=10)
        self.assert_(ret is None)

        mock.VerifyAll()

        self.assert_(self.uri)
        self.assertEquals(self.uri, 'https://api.typepad.com/meese/7/snert.json')
        self.assert_(self.headers)
        self.assert_(self.body)

        self.assert_(utils.json_equals({
            'volume': 10
        }, self.body))
```

6.1.33 Example 33

From project horizon, under directory horizon/horizon, in source file test.py.

```
def setUp(self):
        self.mox = mox.Mox()

        def fake_conn_request(*args, **kwargs):
            raise Exception("An external URI request tried to escape through "
                        "an httplib2 client. Args: %s, kwargs: %s"
                        % (args, kwargs))
    self._real_conn_request = httplib2.Http._conn_request
    httplib2.Http._conn_request = fake_conn_request

    self._real_horizon_context_processor = context_processors.horizon
    context_processors.horizon = lambda request: self.TEST_CONTEXT

    self._real_get_user_from_request = users.get_user_from_request
    self.setActiveUser(token=self.TEST_TOKEN,
                       username=self.TEST_USER,
                       tenant_id=self.TEST_TENANT,
                       service_catalog=self.TEST_SERVICE_CATALOG)
    self.request = http.HttpRequest()
    middleware.HorizonMiddleware().process_request(self.request)
```

6.1.34 Example 34

From project nova, under directory nova/tests, in source file test_hypervapi.py.

```
def __init__(self, test_case_name):
        self._mox = mox.Mox()
        super(HyperVAPITestCase, self).__init__(test_case_name)
```

6.1.35 Example 35

From project nappingcat, under directory tests/gittests, in source file handlers.py.

```
def setUp(self):
        self.test_dir = os.path.expanduser('~/.kittygittests')
        self.mox = mox.Mox()
```

6.1.36 Example 36

From project nappingcat, under directory tests/gittests, in source file operations.py.

```
def setUp(self):
        self.mox = mox.Mox()
        self.cleanup_dirs = []
```

6.1.37 Example 37

From project nappingcat, under directory tests, in source file config.py.

```
def setUp(self):
        self.mox = mox.Mox()
        self.filename = 'tests/.%d.conf' % random.randint(1,100)
```

6.1.38 Example 38

From project nappingcat, under directory tests, in source file serve.py.

```
def setUp(self):
        self.mox = mox.Mox()
```

6.1.39 Example 39

From project appengine-python3-master, under directory google/appengine/tools/devappserver2/admin, in source file taskqueue_utils_test.py.

```
def setUp(self):
    self.mox = mox.Mox()
    self.mox.StubOutWithMock(apiproxy_stub_map, 'MakeSyncCall')
```

6.1.40 Example 40

From project appengine-python3-master, under directory google/appengine/tools/devappserver2/admin, in source file xmpp_request_handler_test.py.

```
def setUp(self):
    self.mox = mox.Mox()
```

6.1.41 Example 41

From project habitat, under directory habitat/tests/test_loadable_manager, in source file test_loadable_manager.py.

```
def setup(self):
        self.mocker = mox.Mox()
        self.mocker.StubOutWithMock(loadable_manager, "dynamicloader")
```

6.1.42 Example 42

From project analyzer-master, under directory tests/unit, in source file broken_tick_feeder.py.

```
def setUp(self):
        self.mock = mox.Mox()
```

6.1.43 Example 43

From project django-lean, under directory django_lean/lean_analytics, in source file tests.py.

```
def setUp(self):
            self.mox = mox.Mox()
            self.analytics = KissMetrics()
```

6.1.44 Example 44

From project cc, under directory vendor/pymox, in source file stubout_test.py.

```
def setUp(self):
    self.mox = mox.Mox()
    self.sample_function_backup = stubout_testee.SampleFunction
```

CHAPTER

SEVEN

API REFERENCE

7.1 Reference

7.1.1 Mox

mox.Mox

7.1.2 MockAnything

mox.MockAnything

7.1.3 MockObject

mox.MockObject

7.1.4 MethodSignatureChecker

mox.MethodCallChecker

7.1.5 MockMethod

mox.MockMethod

7.1.6 Comparators

mox.Comparator

mox.IsA

mox.IsAlmost

mox.StrContains

mox.Regex

mox.In

mox.Not

mox.ContainsKeyValue

mox.ContainsAttributeValue

mox.SameElementsAs

mox.And

mox.Or

mox.Func

mox.IgnoreArg

7.1.7 Method Groups

mox.MethodGroup

mox.UnorderedGroup

mox.MultipleTimesGroup

7.1.8 Testing

mox.MoxMetaTestBase

mox.MoxTestBase

7.1.9 Functions

mox.Replay

mox.Verify

mox.Reset

7.1.10 Exceptions

mox.Error

mox.ExpectedMethodCallsError

mox.UnexpectedMethodCallError

mox.UnknownMethodCallError

mox.PrivateAttributeError

mox.ExpectedMockCreationError

mox.UnexpectedMockCreationError

CHAPTER

EIGHT

PROJECT LINKS

- Stack Overflow
- PyPI
- GitHub
- Documentation
- Changelog

CHAPTER

NINE

INDICES AND TABLES

- genindex
- modindex