Altered States Documentation

Release 0

Jacob Oscarson <jacob@plexical.com>

Oct 20, 2020

Contents

1 API **3**

1.1 state . . . 3

1.2 alter/restore . . . 3

2 More **7**

Index **9**

Altered States is a way to simplify monkey patching and make it more accessible. It was written with test fixture setup in mind but can be used for anything that needs a reversible and temporary drastic state change (switching between authenticated users, I/O redirection, probably more).

CHAPTER 1

API

There are two ways to manipulate your world.

1.1 state

For quick state changes, use the `state()` function by way of a context manager (*with* statement):

```
>>> from altered import state
>>> class Anon(object): pass
>>> o = Anon()
>>> o.foo = 'foo'
>>> with state(o, foo='bar'):
...     print(o.foo)
bar
```

or using the same function as a a *decorator*:

```
>>> from altered import state
>>> struct = {'a': 1}
>>> @state(struct, a=3)
... def fn():
...     return struct['a']
>>> fn()
3
```

This example also shows how `state()` can be applied to *dict* as well as objects.

1.2 alter/restore

(This feature is available from version '0.8.5').

If you need the state to be in effect for a bit longer, use the two-step procedure by calling `alter()`. It returns another function that will perform the restoration at a later time:

```
>>> from altered import alter, E
>>> o = E(foo='foo')
>>> restore = alter(o, foo='bar')
>>> print(o.foo)
bar
>>> restore()
>>> print(o.foo)
foo
```

It also takes *dict* -like objects in the same way that `state()` does.

Contents:

1.2.1 Examples

Here are some examples to get you started on usage of Altered States:

I/O redirection

```
>>> import sys
>>> from altered import state, py23compat
>>> buf = py23compat.strio()
>>> with state(sys, stdout=buf):
...     print('foo')
>>> buf.getvalue()
'foo\n'
```

Faking an import

```
>>> import sys
>>> from altered import state, Expando
>>> with state(sys.modules, fakey=Expando(foo='bar') ):
...     import fakey
...     print(fakey.foo)
bar
```

In-place patching

Module scope

```
>>> @state(globals(), injected='foo')
... def fn():
...     return injected
>>> fn()
'foo'
```

Local scope

```
>>> from altered import state, E
>>> with state(vars(), injected='foo'):
...     print(injected)
foo
```

Deny the existance of a module

```
>>> import sys
>>> from altered import state
>>> with state(sys.modules, shutil=None):
...     import shutil # doctest: +SKIP
Traceback (most recent call last):
    ...
ModuleNotFoundError: import of 'shutil' halted; None in sys.modules
>>> import shutil
```

Nested structure

```
>>> from altered import state, Expando
>>> ctx = Expando()
>>> idx = 0
>>> users = [Expando(name='Foo', get_token=lambda: 'xyz')]
>>> @state(ctx, users=users)
... def token(idx):
...     return ctx.users[idx].get_token()
>>> token(0)
'xyz'
```

1.2.2 Expando objects

Altered States also contains an optional feature called *Expando* objects. It's a simple object that can be used to create replacement structures easily. It's basically an empty object that you can add any extra attributes to, with a conceptual implementation along the lines of:

```
class Expando(object):
    def __init__(self, *args, **kw):
        self.__dict__.update(kw)
```

Full source is marginally more complex, see here. So if you need an object with another object embedded that has a method you can create that with:

```
>>> from altered import Expando
>>> faked_ctx = Expando(user=Expando(get_name=lambda: 'Foo Bar'))
>>> faked_ctx.user.get_name()
'Foo Bar'
```

Using and *Expando* object with Altered States can look like this:

```
>>> from altered import Expando, state
>>> obj = Expando(a=1)
>>> @state(obj, a=3)
... def fn():
...     return obj.a
>>> fn()
3
```

Expando classes are aliased to the name *E* if you're seeking maximum terseness.

1.2.3 The state function

state(*original*[, *change1=change1*, *changeN=changeN*])

A *ContextDecorator* that takes *original* and applies the changes sent as parameters and modifies *original* with these parameters. Upon completion it will restore *original* to the state it was before being called. Parameters can also have the marker value of `forget` to temporary remove this name when `state()` is in effect.

1.2.4 The alter function

alter(*original*[, *change1=change1*, *changeN=changeN*])

Modified *original* and applies the changes sent as parameters. Parameters can also have the marker value of `forget` to temporary remove this name while the changes are in effect.

Returns a new function that will reverse the effect of itself.

1.2.5 forget

class forget

A marker class that is sent as the value of a parameter in a `state()` call to show that this symbol should be taken out of the original object while the changed state are in effect.

CHAPTER 2

More

- search

Index

A

`alter()` (*built-in function*), 6

F

`forget` (*built-in class*), 6

S

`state()` (*built-in function*), 6