tf.test.StubOutForTesting

Contents

Class StubOutForTesting
Methods
__init__
CleanUp

Class **StubOutForTesting**

Defined in `tensorflow/python/platform/googletest.py`.

Support class for stubbing methods out for unit testing.

Sample Usage:

You want os.path.exists() to always return true during testing.

stubs = StubOutForTesting() stubs.Set(os.path, 'exists', lambda x: 1) ... stubs.CleanUp()

The above changes os.path.exists into a lambda that returns 1. Once the ... part of the code finishes, the CleanUp() looks up the old value of os.path.exists and restores it.

Methods

__init__

```
__init__()
```

CleanUp

```
CleanUp()
```

Undoes all SmartSet() & Set() calls, restoring original definitions.

Set

```
Set(
    parent,
    child_name,
    new_child
)
```

In parent, replace child_name's old definition with new_child.

The parent could be a module when the child is a function at module scope. Or the parent could be a class when a class' method is being replaced. The named child is set to new_child, while the prior definition is saved away for later, when UnsetAll() is called.

This method supports the case where child_name is a staticmethod or a classmethod of parent.

Args:

* `parent`: The context in which the attribute child_name is to be changed.
* `child_name`: The name of the attribute to change.
* `new_child`: The new value of the attribute.

`SmartSet`

```
SmartSet(
    obj,
    attr_name,
    new_attr
)
```

Replace obj.attr_name with new_attr.

This method is smart and works at the module, class, and instance level while preserving proper inheritance. It will not stub out C types however unless that has been explicitly allowed by the type.

This method supports the case where attr_name is a staticmethod or a classmethod of obj.

Notes: - If obj is an instance, then it is its class that will actually be stubbed. Note that the method Set() does not do that: if obj is an instance, it (and not its class) will be stubbed. - The stubbing is using the builtin getattr and setattr. So, the **get** and **set** will be called when stubbing (TODO: A better idea would probably be to manipulate obj.**dict** instead of getattr() and setattr()).

Args:

* `obj`: The object whose attributes we want to modify.
* `attr_name`: The name of the attribute to modify.
* `new_attr`: The new value for the attribute.

Raises:

* `AttributeError`: If the attribute cannot be found.

`SmartUnsetAll`

```
SmartUnsetAll()
```

Reverses SmartSet() calls, restoring things to original definitions.

This method is automatically called when the StubOutForTesting() object is deleted; there is no need to call it explicitly.

It is okay to call SmartUnsetAll() repeatedly, as later calls have no effect if no SmartSet() calls have been made.

`UnsetAll`

```
UnsetAll()
```

Reverses Set() calls, restoring things to their original definitions.

This method is automatically called when the StubOutForTesting() object is deleted; there is no need to call it explicitly.

It is okay to call UnsetAll() repeatedly, as later calls have no effect if no Set() calls have been made.

`__del__`

```
__del__()
```

Do not rely on the destructor to undo your stubs.

You cannot guarantee exactly when the destructor will get called without relying on implementation details of a Python VM that may change.

`__enter__`

```
__enter__()
```

`__exit__`

```
__exit__(
    unused_exc_type,
    unused_exc_value,
    unused_tb
)
```