depocs Documentation

Release stable

February 12, 2015

Contents

`Scoped` is a mixin class that creates a thread-local stack for each of its subclasses. Instances of the subclass can be pushed and popped on this stack, and the instance at the top of the stack is always available as a property of the class. `Scoped` classes are typically used to make parameters implicitly available within a (dynamic) scope, without having to pass them around as function arguments. `Scoped` helps you do this in a safe and convenient way, and provides very informative error messages when you do something wrong.

CHAPTER 1

Examples

A `Scoped` class can be used to pass around contextual data:

```
class Session(Scoped):
    def __init__(self, user):
        self.user = user

def print_current_user():
    print(Session.current.user)

with Session(some_guy):
    print_current_user()
```

It can also be used to provide dependencies:

```
class Clock(Scoped):
    def __init__(self, now=None):
        self._now = now

    def now(self):
        return self._now or datetime.now()

Clock.default = Clock()

def print_time()
    print(Clock.current.now())

print_time() # Prints the real time

def test_print_fake_time(self):
    with Clock(datetime(2000, 1, 1)):
        print_time() # Prints a fake time
```

CHAPTER 2

Opening and Closing

`Scoped` objects are best used as context managers (i.e. using the `with` statement), but for situations where this isn't possible, you can also open and close them "manually":

```
class Transaction(Scoped):
    ...

transaction = Transaction().open()
try:
    ...
finally:
    transaction.close()
```

Obviously, you will need to do whatever is necessary to ensure that every call to `open` is matched by a call to `close`.

CHAPTER 3

Options

The behavior of `Scoped` subclasses can be customized by declaring options in a nested class named `ScopedOptions`. Except where noted, options are automatically inherited by subclasses that do not override them:

```
class Thingy(Scoped):
    class ScopedOptions:

        # If True, instances will share the stack of their parent class.
        # If False, this class will have its own stack independent of any
        # ancestors. The default is to inherit the stack, unless subclassing
        # Scoped directly. This option is NOT inherited by subclasses.
        inherit_stack = False

        # Maximum number of scopes that can be nested on this stack.
        # This cannot be overridden if inheriting the parent stack.
        max_nesting = 16

        # If True, instances can be re-opened after being closed.
        # If False, instances can only be opened and closed once, and will
        # raise a LifecycleError on any attempt to reopen them.
        allow_reuse = False
```

CHAPTER 4

Default Instance

An instance of a Scoped subclass can be assigned to the `default` property of the class. This instance will be the value of the `current` property when the stack is empty i.e. when no other instances are open. The default instance itself is not opened by virtue of being the default. Opening it will push it onto the stack like any other instance.

CHAPTER 5

Errors

`Scoped` has three inner exception classes that it will raise for various error conditions: `Scoped.Error` is the base class for the other two, which are `Scoped.Missing` and `Scoped.Lifecycle`.

`Scoped.Missing` is raised when an attempt is made to access a scoped object that is not available, i.e. when accessing `Scoped.current` with an empty stack and no default instance.

`Scoped.Lifecycle` is raised on any attempt to open or close a scoped object at the wrong time e.g. opening an object that is already open, closing an object that is not at the top of the stack, and various other cases.

Both of these exceptions are automatically subclassed along with their containing class. Each subclass of `Scoped` gets its own exception classes that inherit from the base exceptions. This allows you to easily handle errors from particular scoped classes without worrying about catching unrelated errors from other scoped classes.