sgmock Documentation

Release 0.1

Western X

February 25, 2016

Contents

This Python package is a mock Shotgun server for use in unit testing, and a series of tools for building test fixtures. It emulates the experience of using shotgun_api3 with a Shotgun server near version 4.0. It is fairly incomplete, but is enough for testing the majority of our own tools which tend to focus heavily on consuming data from Shotgun instead of manipulating it.

This mockup does not perfectly emulate the Shotgun experience. In general, it makes the assumption that your usage of the Shotgun API is correct, and that you are testing your manipulation of the data that comes out of it. For example:

- schemas are not enforced (nor do they exist at all);
- there are no default values, and unspecified fields will not be retrievable later;
- entities do not require a `project`.

Also, the mockup starts with absolutely no entities. That includes things that you would normally have by default, e.g. pipeline steps.

If you ask the mockup to do something that it doesn't understand then we try to fail quickly and fail really hard. Usually with a *`ShotgunError`* or `NotImplementedError`.

Good luck, and happy testing!

CHAPTER 1

API Reference

1.1 sgmock.shotgun

Mock replacement for `shotgun_api3` module.

class `sgmock.shotgun.`**Shotgun**(*base_url=None*, **args*, ***kwargs*)
A mock Shotgun server, replicating the `shotgun_api3` interface.

The constructor is a dummy and eats all arguments.

batch(*requests*)
Perform a series of requests in one request.

This mock does not have transactions, so a failed request will leave the data store in a partially mutated state.

Parameters **requests** (*list*) – A series of `dicts` representing requests.

Returns `list` of results from calling methods individually.

create(*entity_type*, *data*, *return_fields=None*, *_log=True*, *_generate_events=True*)
Store an entity of the given type and data; return the new entity.

Parameters

- **entity_type** (*str*) – The type of the entity.
- **data** (*dict*) – The fields for the new entity.
- **return_fields** (*list*) – Which fields to return from the server in addition to those explicitly stored; only good for `updated_at` in this mock version.

delete(*entity_type*, *entity_id*, *_generate_events=True*)
Delete a single entity.

This mock does not have retired entities, so once it is deleted an entity is gone.

Parameters

- **entity_type** (*str*) – The type of the entity to delete.
- **entity_id** (*int*) – The id of the entity to delete.

Return bool `True` if the entity did exist.

find(*entity_type*, *filters*, *fields=None*, *order=None*, *filter_operator=None*, *limit=500*, *retired_only=False*, *page=1*)
Find and return entities satifying a list of filters.

We currently support deep-linked fields in the return fields, but not in filters.

Parameters

- **entity_type** (*str*) – The type of entities to find.
- **filters** (*list*) – A list of filters
- **fields** (*list*) – Which fields to return.
- **order** – Ignored.
- **filter_operator** – Ignored.
- **limite** – Ignored.
- **retired_only** – Ignored.
- **page** – Ignored.

Returns list of ‘‘dict‘‘s.

find_one(*entity_type*, *filters*, *fields=None*, *order=None*, *filter_operator=None*, *retired_only=False*)
Find and return a single entity.

This is the same as calling *find()* and only returning the first result.

Returns dict or None.

exception sgmock.shotgun.**ShotgunError**
Exception for all server logic.

exception sgmock.shotgun.**Fault**
Exception for all remote API logic; unused in this mock.

1.2 sgmock.fixture

1.3 sgmock.unittest

class sgmock.unittest.**TestCase**(*methodName='runTest'*)
Extensions to unittest.TestCase which understand entities, and backport many methods from Python2.7.

assertSameEntity(*a*, *b*, *msg=None*)
Assert that the two given entities refer to the same object.

E.g.: Both are dicts and their types and ids match.

assertNotSameEntity(*a*, *b*, *msg=None*)
Assert that the two given entities do not refer to the same object.

E.g.: Both are dicts and their types and ids do not match.

assertIs(*expr1*, *expr2*, *msg=None*)
Just like self.assertTrue(a is b), but with a nicer default message.

assertIsNot(*expr1*, *expr2*, *msg=None*)
Just like self.assertTrue(a is not b), but with a nicer default message.

assertIsNone(*obj*, *msg=None*)
Same as self.assertTrue(obj is None), with a nicer default message.

assertIsNotNone(*obj*, *msg=None*)
Included for symmetry with assertIsNone.

assertIn(*member*, *container*, *msg=None*)
Just like self.assertTrue(a in b), but with a nicer default message.

assertNotIn(*member*, *container*, *msg=None*)
Just like self.assertTrue(a not in b), but with a nicer default message.

assertIsInstance(*obj*, *cls*, *msg=None*)
Same as self.assertTrue(isinstance(obj, cls)), with a nicer default message.

assertNotIsInstance(*obj*, *cls*, *msg=None*)
Included for symmetry with assertIsInstance.

CHAPTER 2

Indices and tables

- genindex
- modindex
- search

S

A

B

C

D

F

S

T