# sgsession Documentation

***Release 0.1***

**Western X**

October 05, 2015

Contents

This Python package is a wrapper around shotgun_api3 for Shotgun which provides a local data cache and some additional intelligence on top of bare entities.

This has been crafted to mimick the normal interface, and graft extra features on top. Howveer, this is **not** a drop-in replacement for the normal API. While any individual entity in isolation should behave in the same way as with the normal API, the entities are linked behind the scenes and so complex behaviour is likely to break.

CHAPTER 1

# Contents

## 1.1 Overview

### 1.1.1 Getting Started

All you must do to start using `sgsession` is to construct a *Session* with an existing Shotgun instance:

```
>>> from shotgun_api3 import Shotgun
>>> from sgsession import Session
>>> session = Session(Shotgun(*shotgun_args))
```

From then on you can use the `session` as you would have used the Shotgun instance itself.

### 1.1.2 The Entity

The primary representation of Shotgun entities is the *Entity* class, which extends the familiar dictionary with more Shotgun specific methods and a link back to the session for fetching more fields, parents, etc..

### 1.1.3 Instance Sharing

The same *Entity* instance will always be returned for the same conceptual instance. E.g.:

```
>>> a = session.find_one('Task', [('code', 'is', 'AA_001')])
>>> b = session.find_one('Task', [('code', 'is', 'AA_001')])
>>> a is b
True
```

### 1.1.4 Caching, Merging, and Backrefs

Entities will be cached for the lifetime of their *Session*, and any new information about them will be merged in as it is encountered.

For example, fields fetched in subsequent queries to the server will be availible on earlier found entities:

```
>>> a = session.find_one('Task', [('code', 'is', 'AA_001')])
>>> 'sg_status_list' in a
False
>>> b = session.find_one('Task', [('code', 'is', 'AA_001')], ['sg_status_list'])
```

```
>>> a['sg_status_list']
'fin'
```

Deep-linked fields will also be merged into the main scope of the linked entities for easy referral:

```
>>> x = session.find_one('Task', [], ['entity.Shot.code'])
>>> x['entity']['code']
'AA_001'
```

Links to other entities will automatically populate backrefs on the remote side of the link, allowing for entities to easily find there they have been linked from:

```
>>> task = session.find_one('Task', [], ['entity'])
>>> shot = task['entity']
>>> task in shot.backrefs[('Task', 'entity')]
True
```

### 1.1.5 Important Fields

Several fields will always be queried for whenever you operate on entities. These include `Shot.code`, `Task.step`, and `project` on many types. When availible, deep-linked fields will also be fetched, including `Task.entity.Shot.code`, so even single simple queries will return lots of related data:

```
>>> x = session.find_one('Task', [])
>>> x.pprint()
Task:456 at 0x101577a20; {
    entity = Shot:234 at 0x101541a80; {
            code = 'AA_001'
            name = 'AA_001'
            project = Project:123 at 0x101561f30; {
                    name = 'Demo Project'
            }
    }
    project = Project:123 at 0x101561f30; ...
    sg_status_list = 'fin'
    step = Step:345 at 0x10155b5e0; {
            code = 'Matchmove'
            entity_type = 'Shot'
            name = 'Matchmove'
            short_name = 'Matchmove'
    }
}
```

### 1.1.6 Brace Expansion

During a `find` or `find_one`, return fields can be specified with brace expansions to allow for a more compact representation of complex links:

```
>>> session.find('Task', [], ['entity.{Asset,Shot}.{name,code}'])
```

### 1.1.7 Efficient Heirarchies

Ever have a list of tasks that you need to know the full heirarchy for all the way up to the project? With any number of tasks, you can get all of the important fields for the full heirarchy in no more than 3 requests:

```
>>> tasks = session.find('Task', some_filters)
>>> all_entities = session.fetch_heirarchy(tasks)
```

`all_entities` is a list of every entity above those tasks, and every entity has been linked and backreffed to each other.

## 1.2 sgsession.Session

The Session is a wrapper around a Shotgun instance, proxying requests to the server and applying additional logic on top of it. The Session instance is designed to be used for a single task and then discarded, since it makes the assumption that entity relationships do not change.

While not fully documented below, this object will proxy all attributes to the underlying Shotgun instance, so you can treat this as you would a Shotgun instance.

**class** sgsession.session.**Session**(*shotgun=None*, *schema=None*, **args*, ***kwargs*)

Shotgun wrapper.

**Parameters** **shotgun** – A Shotgun instance to wrap, or the name to be passed to `shotgun_api3_registry.connect()` in order to construct one.

If passed a name, the remaining args and kwargs will also be passed to the api registry connector.

If passed a descendant of `shotgun_api3.Shotgun` (or one is constructed via the registry), it will be wrapped in a *ShotgunPool* so that it becomes thread-safe. Any other objects (e.g. mock servers) are used unmodified.

### 1.2.1 Entity Control

Session.**merge**(*data*, *over=None*, *created_at=None*, *resolve=True*)

Import data containing raw entities into the session.

This will effectively return a copy of any nested structure of lists, tuples, and dicts, while converting any dicts which look like entities into an *Entity*. The returned structure is a copy of the original.

**Parameters**

- **data** (*dict*) – The raw fields to convert into an *Entity*.
- **over** (*bool*) – Control for merge behaviour with existing data. `True` results in the new data taking precedence, and `False` the old data. The default of `None` will automatically decide based on the `updated_at` field.

**Returns** The *Entity*. This will not be a new instance if the entity was already in the session, but it will have all the newly merged data in it.

Session.**get**(**args*, ***kwargs*)

Get one entity by type and ID.

**Parameters**

- **type** (*str*) – The entity type to lookup.
- **id** (*int*) – The entity ID to lookup. Accepts `list` or `tuple` of IDs, and returns the same.
- **fetch** (*bool*) – Request this entity from the server if not cached?

Session.**filter_exists**(**args*, ***kwargs*)

Return the subset of given entities which exist (non-retired).

**Parameters**

- **entities** (*list*) – An iterable of entities to check.
- **check** (*bool*) – Should the server be consulted if we don't already know?
- **force** (*bool*) – Should we always check the server?

**Returns set** The entities which exist, or aren't sure about.

This will handle multiple entity-types in multiple requests.

### 1.2.2 Fetching Fields

Session.**fetch**(**args*, ***kwargs*)

Fetch the named fields on the given entities.

**Parameters**

- **to_fetch** (*list*) – Entities to fetch fields for.
- **fields** (*list*) – The names of fields to fetch on those entities.
- **force** (*bool*) – Perform a request even if we already have this data?

This will safely handle multiple entitiy types at the same time, and by default will only make requests of the server if some of the data does not already exist.

---

**Note:** This does not assert that all "important" fields exist. See *fetch_core()*.

---

Session.**fetch_core**(**args*, ***kwargs*)

Assert all "important" fields exist, and fetch them if they do not.

**Parameters** **to_fetch** (*list*) – The entities to get the core fields on.

**This will populate all important fields, and important fields on linked** entities.

Session.**fetch_backrefs**(**args*, ***kwargs*)

Fetch requested backrefs on the given entities.

**Parameters**

- **to_fetch** (*list*) – Entities to get backrefs on.
- **backref_type** (*str*) – The entity type to look for backrefs on.
- **field** (*str*) – The name of the field to look for backrefs in.

```
    # Find all tasks which refer to this shot.
    >>> session.fetch_backrefs([shot], 'Task', 'entity')
```

Session.**fetch_heirarchy**(**args*, ***kwargs*)

Populate the parents as far up as we can go, and return all involved.

With (new-ish) arbitrarily-deep-links on Shotgun, this method could be made quite a bit more effiecient, since it should be able to request the entire heirarchy for any given type at once.

See *parent_fields*.

### 1.2.3 Importance Controls

These class attributes control which fields are considered "important", which types are potentially linked to by various fields, and which types are considered the parent of other types.

`Session.`**`important_fields_for_all`** **= ['updated_at']**

`Session.`**`important_fields`** **= {'Project': ['name'], 'Step': ['code', 'short_name', 'entity_type'], 'Task': ['step', 'content']**

`Session.`**`important_links`** **= {'PublishEvent': {'project': ['Project'], 'sg_link': ['Task']}, 'Task': {'project': ['Project'], 's**

`Session.`**`parent_fields`** **= {'Task': 'entity', 'Shot': 'sg_sequence', 'Sequence': 'project', 'Project': None, 'Version': 'entit**

### 1.2.4 Wrapped Methods

`Session.`**`create`**(**args*, ***kwargs*)
Create an entity of the given type and data.

**Returns** The new *Entity*.

See the Shotgun docs for more.

`Session.`**`find`**(**args*, ***kwargs*)
Find entities.

**Returns** `list` of found *Entity*.

See the Shotgun docs for more.

`Session.`**`find_one`**(**args*, ***kwargs*)
Find one entity.

**Returns** *Entity* or `None`.

See the Shotgun docs for more.

`Session.`**`update`**(**args*, ***kwargs*)
Update the given entity with the given fields.

---

**Todo**

Add this to the Entity.

---

See the Shotgun docs for more.

`Session.`**`delete`**(**args*, ***kwargs*)
Delete one entity.

> **Warning:** This session will **not** forget about the deleted entity, and all links from other entities will remain intact.

See the Shotgun docs for more.

`Session.`**`batch`**(**args*, ***kwargs*)
Perform a series of requests in a transaction.

See the Shotgun docs for more.

## 1.3 sgsession.Entity

**class** sgsession.entity.**Entity**(*type_*, *id_*, *session*)
A Shotgun entity.

This behaves much like the dict the Shotgun API normally returns does, but understands the links bettween entities in its associated session.

### 1.3.1 Entity Management

Entity.**minimal**
The minimal representation of this entity; a dict with type and id.

Entity.**as_dict**()
Return the entity and all linked entities as pure dict.

The first reference to an entity will have all availible fields, and any subsequent ones will be the minimal representation. This is the ideal format for serialization and remerging into a session.

Entity.**pprint**(*backrefs=None*, *depth=0*)
Print this entity, all links and optional backrefs.

Entity.**exists**(**args*, ***kwargs*)
Determine if this entity still exists (non-retired) on the server.

**Parameters**

- **check** (*bool*) – Check against the server if we don't already know.
- **force** (*bool*) – Always recheck with the server, even if we already know.

**Returns bool** True/False if it is known to exist or not, and None if we do not know.

See *Session.filter_exists()* for the bulk version.

### 1.3.2 Retrieving Data

Entity.**get**(**args*, ***kwargs*)
Get field value(s) if they exist, otherwise a default.

**Parameters**

- **fields** – A str field name or collection of str field names.
- **default** – Default value to return when field does not exist.

If passed a single field name as a str, return the coresponding value. If passed field names as a list or tuple, return a tuple of coresponding values.

Entity.**fetch**(**args*, ***kwargs*)
Get field value(s), automatically fetching them from the server.

**Parameters**

- **fields** – A str field name or collection of str field names.
- **default** – Default value to return when field does not exist.
- **force** (*bool*) – Force an update from the server, otherwise only query they server if fields have been requested that we do not already have.

If passed a single field name as a `str`, return the coresponding value. If passed field names as a list or tuple, return a tuple of coresponding values.

See *Session.fetch()* for the bulk version.

`Entity.`**`fetch_core`**`(*args, **kwargs)`
Assert that all "important" fields exist on this Entity.

See *Session.fetch_core()* for the bulk version.

`Entity.`**`fetch_backrefs`**`(*args, **kwargs)`
Fetch all backrefs to this Entity from the given type and field.

See *Session.fetch_backrefs()* for the bulk version.

`Entity.`**`fetch_heirarchy`**`(*args, **kwargs)`
Fetch the full upward heirarchy (toward the Project) from the server.

See *Session.fetch_heirarchy()* for the bulk version.

### 1.3.3 Heirarchy

`Entity.`**`parent`**`(*args, **kwargs)`
Get the parent of this Entity, automatically fetching from the server.

`Entity.`**`project`**`(*args, **kwargs)`
Get the project of this Entity, automatically fetching from the server.

Depending on what part of the heirarchy is already loaded, many more entities will have their Project fetched by this single call.

## 1.4 sgsession.ShotgunPool

A wrapper around `shotgun_api3` to allow for parallel requests.

The standard Shotgun API uses a connection object that serialized requests. Therefore, efficient usage in a multi-threaded environment is tricker than it could be. Ergo, this module was concieved.

*ShotgunPool* is a connection pool that creates fresh Shotgun instances when needed, and recycles old ones after use. It proxies attributes and methods to the managed instances. An actual Shotgun instance should never leak out of this object, so even passing around bound methods from this object should be safe.

E.g.:

```
>>> # Construct and wrap a Shotgun instance.
>>> shotgun = Shotgun(...)
>>> shotgun = ShotgunPool(shotgun)
>>>
>>> # Use it like normal, except in parallel.
>>> shotgun.find('Task', ...)
```

**class** `sgsession.pool.`**`ShotgunPool`**`(prototype, *args, **kwargs)`
Shotgun connection pool.

**Parameters** **`prototype`** – Shotgun instance to use as a prototype, OR the `base_url` to be used to construct a prototype.

If passed a `base_url`, the remaining args and kwargs will be passed to the Shotgun constructor for creation of a prototype.

The `config` object of the prototype will be shared amoung all Shotgun instances created; changing the settings on one Shotgun instance (or this object) will affect all other instances.

## S