sgfs Documentation

Release 0.1

Western X

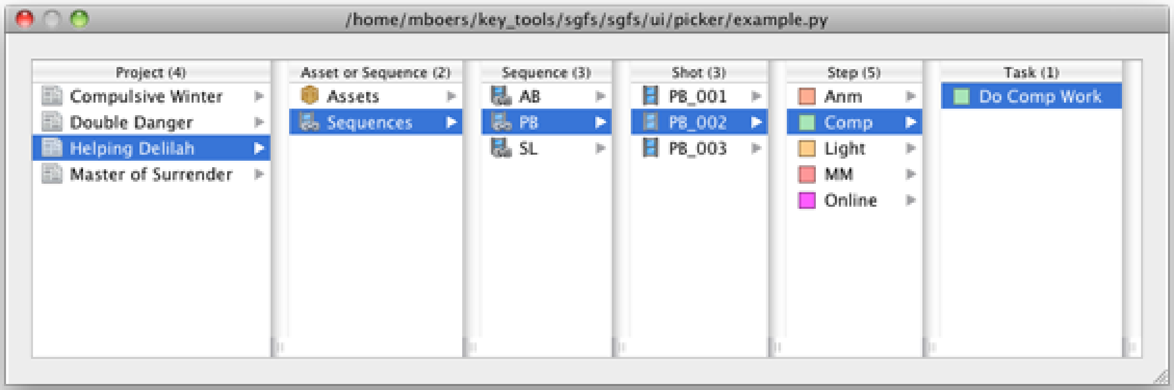
March 03, 2016

Contents

This Python package is a translation layer between Shotgun entities and a file structure on disk. In general, its overarching tasks are to:

- map Shotgun entities to their canonical path on disk;
- map paths on disk to the coresponding Shotgun entities;
- create new structures on disk;
- indentify existing structures on disk for use in above translation.

We also provide a number of Qt Widgets to assist in working with Shotgun:



CHAPTER 1

Contents

1.1 Overview

1.1.1 Tags

Once folders have been created, the mapping between those folders and the Shotgun entities from which they originated are maintained via **tags**. These tags exist within `.sgfs.yml` files within the top-level of the directory that corresponds to the given entity.

Location

With the WesternX structure, the project tree (including tags) looks roughly like:

```
The_Awesome_Project/
    .sgfs.yml # Project tag.
    .sgfs-cache.sqlite # Reverse cache.
    SEQ/
        AA/
            .sgfs.yml # "AA" Sequence tag.
            AA_001_001/
                .sgfs.yml # "AA_001_001" Shot tag.
                Light/
                    .sgfs.yml # Tags for Tasks with step code "Light".
    Assets/
        Character/
            Cow/
                .sgfs.yml # "Cow" Asset tag.
                Model/
                    .sgfs.yml # "Tags for Tasks with step code "Model".
```

Contents

The `.sgfs.yml` files are YAML documents containing a logical document for each tag. Those documents are mappings including a timestamp, the entity for that tag, and other arbitrary metadata. The entities have been dumped with all the information that was known about their lineage up to the project. For example, a tag for a shot may look like:

```
---
created_at: 2012-10-23 18:27:24.312373
entity:
    code: RG_006_001
    id: 5847
    project:
        id: 70
        name: Super Buddies
        type: Project
        updated_at: 2012-09-17 22:40:23
    sg_sequence:
        code: RG
        id: 107
        name: RG
        project:
            id: 70
            type: Project
        type: Sequence
        updated_at: 2012-10-23 19:29:58
    type: Shot
    updated_at: 2012-10-24 01:31:37
```

Rules

Usage of tags follows a few general rules:

- tags must contain an entity;
- tags may optionally contain metadata in addition to that entity;
- a directory may be tagged multiple times with different entities;
- if a directory is tagged more than once with the same entity, only the most recent tag will be returned and older metadata will be lost (although older Shotgun data will be merged into the session if not outdated).

1.1.2 The Path Cache

While *tags* create a link from directories to their corresponding entities, the **path cache** maintains the links from entities to directories in which they are tagged.

The path cache is implemented as a sqlite3 database located at `.sgfs/cache.sqlite` within each project, and accessible via the `PathCache` API.

Since the data in the path cache and tags is redundant, the path cache should be treated as a derivative of the tags and may be reconstructed from the tags at any time via the *sgfs-relink* command.

1.1.3 Caveats or Known Issues

- Projects must be tagged manually in order for other tools to be able to create structures within them (by default). This is partially a technical restriction (for the creation of the *path cache*, but also for safety. Manual tagging is done via the *sgfs-tag* command:

```
sgfs-tag Project 1234 path/to/project
```

1.1.4 Contexts, Schemas, and Structures

The `Context`, `Schema`, and `Structure` are three different (but related) directed acyclic graphs used in the construction of file structures on disk.

A `Context` represents a set of Shotgun entities and their relationships.

A `Schema` represents a template for file structures, and is defined via template structures and YAML files describing them.

A `Structure` is the specific directories and files that should exist for a set of entities, and allows for creation or inspection of those structures. It is created by rendering Schema for a given Context.

1.2 Command-line Tools

A number of command line tools have been created to deal with common situations. When asked to specify an entity to work on, they will generally accept the following forms:

- a path (e.g. `.` or `SEQ/GB/GB_001_001`);
- an entity type and ID (e.g. `shot 1234`);
- a sequence code (e.g. `pv` or `GB`);
- a shot code (e.g. `gb_001` or `PV_007_002`);
- nothing, and it will use the current working directory.

1.2.1 Basics

sgfs-cd

Move your terminal to the given entity.

sgfs-open

Open the folder for the given entity.

sgfs-shotgun

Open the Shotgun page for the given entity.

1.2.2 Tags and Caches

sgfs-tag

```
$ sgfs-tag <entity_type> <entity_id> <path_to_folder>
```

sgfs-update

When operating with paths instead of entities, SGFS uses entity fields cached in the folder tags. We tend to only cache fields that rarely change, but sometimes, e.g. when a shot or sequence is renamed, those fields need to be updated.

This command will rewrite the tags with up-to-date data:

```
Update the cached tag data for the current folder.
$ sgfs-update .

Update the cached tag data for every entity in the current folder.
$ sgfs-update -r .
```

sgfs-relink

When a folder is moved on disk, the one of the two links between Shotgun and that folder is broken, and you will not be able to get a path from an entity any more.

The links must be recreated with this tool:

```
Relink the entity for the current folder.
$ sgfs-relink .

Relink all entities under this folder.
$ sgfs-relink -r .
```

Since this is a common situation after renaming shots or sequences, this tool can automatically call the updater on paths that were relinked:

```
$ sgfs-relink -r --update .
```

1.3 SGFS API Reference

The `SGFS` object is the main entrypoint into most functions of this package. Generally, you construct a `SGFS` object and use it to map entities to paths, get contexts from entities, and create structures.

Secondary classes such as `Context` are not created directly since they must remain connected to their original `SGFS`.

1.3.1 Entities

1.3.2 Templates

1.3.3 Contexts

1.3.4 Structure

1.3.5 Tags and Caches

1.4 Known Issues

Most of the issues with SGFS come up with invalid data in the *tags* that are placed in the file structure, either due to information changing on Shotgun, or folders being moved/copied.

1.4.1 Missing Links

Starting a Project

The folder for a project cannot be created from Shotgun (simply because we have deemed it special). You must manually create it and then tag it via *sgfs-tag*:

```
$ sgfs-tag Project 1234 /Volumes/VFX/Projects/New_Project_Folder
```

(This example assumes the Shotgun ID of the project's entity is `1234`.)

Manually Copied Folders

Folders (or parents of folders) linked to a Shotgun entity should not be copied as a template for another structure. If they are relinked (via *sgfs-relink*) then the copy may override the original as far as Shotgun is concerned.

To fix, delete all `.sgfs.yml` files in the duplicate folder:

```
$ find $copy -name .sgfs.yml -delete
```

relink the original:

```
$ sgfs-relink -r $original
```

and then create the folders for the copied entity from Shotgun itself.

1.4.2 Invalid Data

Quite a bit of data is cached in the *tags*, although much of it isn't critical. Some of it, however, can lead to some very strange Python exceptions.

Changing the pipeline step, name, or code of a task after the folders have been created has led to some strange exceptions in the past. This is likely due to some of the older tools using these fields as part of some string-based path construction.

Either put the value back to what it was, or use *sgfs-update* on the folders.

CHAPTER 2

Low-Level APIs

2.1 Templates

2.1.1 Bound Templates

`BoundTemplate.`**`template`**

The `Template` that is bound.

`BoundTemplate.`**`structure`**

The `Structure` that the *`template`* is bound to.

2.1.2 Match Results

2.2 Context Graphs

2.3 Schema Graphs

2.4 Structure Graphs

2.5 Path Caches

2.6 Utilities

2.7 Processors

CHAPTER 3

Indices and tables

- genindex
- modindex
- search

Index