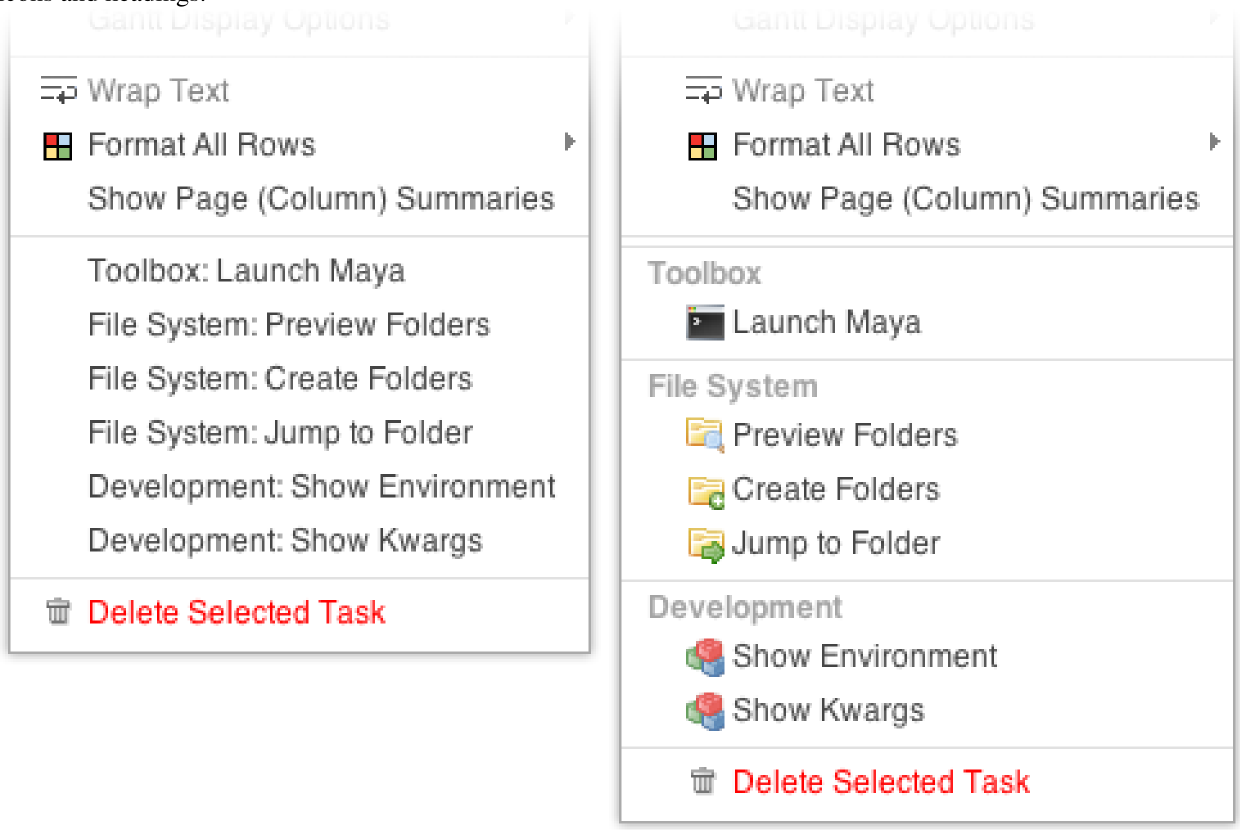
sgactions Documentation

Release 1.0.1

Western X

September 10, 2015

This Python package is a wrapper around `ActionMenuItems` in Shotgun providing a simple interface to Python callables via a YAML configuration. It also has an optional Greasemonkey-style user script to allow to injection of icons and headings.



CHAPTER 1

Installation

1.1 Shotgun API Keys

There are two ways to provide Shotgun API keys to the dispatcher: environment variables or a `shotgun_api3_registry.connect()` function.

For the first, set `SHOTGUN_SERVER`, `SHOTGUN_SCRIPT_NAME`, and `SHOTGUN_SCRIPT_KEY` in your execution environment.

For the second, create a `shotgun_api3_registry` module with a `connect` function that returns a `shotgun_api3.Shotgun` instance.

1.2 Python

The `agaction` package must be importable from the environment that your browser runs in. Check `~/.xsession-errors` on Linux or the Console.app on OS X if your commands seems to be vanishing into a black hole.

1.3 Protocol Handlers and Chrome Extensions

Shotgun action menu items allow for execution of your code in two ways: via a `POST` to another webserver under your control, or navigating to an arbitrary URI. We hook into the second method via protocol handers on OS X and Linux by responding to the `sgaction` protocol. We have unified both cases into a single script:

```
python -m sgactions.register
```

This script must be run once per user per machine. It is recommended to run this command on user login, and we have provided a `launchctl` agent which will perform this for you on OS X. Simply copy it to each user's home folder:

```
mkdir -p ~/Library/LaunchAgents
cp LaunchAgents/com.westernx.sgactions.plist ~/Library/LaunchAgents/
```

or into the system wide folder:

```
cp LaunchAgents/com.westernx.sgactions.plist /Library/LaunchAgents/
```

This script will also automatically install our Google Chrome extension for adding icons, and headings. The changes will only take hold if the register command is run after Chrome has run for the first time and while Chrome is not running. This extension is known to work for Shotgun v4.0 through v4.1

Whenever a new version of sgactions is installed, re-registering may be required, and it will clean up any old hooks it may have left in your system.

Configuration

Actions are specified in a YAML file as a list of dictionaries, one for each action. A simple configuration looks like:

```
- entrypoint: my.python.package:run_sgaction
  title: Run my action!
```

This will register "Run my action!" on every Shotgun entity to call `run_sgaction` within the `my.python.package` module.

We can specify the rest of the standard `ActionMenuItem` fields in a similar way:

```
- entrypoint: my.python.package:run_sgaction
  folder: My Actions
  title: Run my action on Shots or Tasks!
  entity_types: [Shot, Task]
  list_order: 1
  selection_required: true
```

A special syntax will also be interpreted by the browser plugin in order to create headings and icons in titles and folders. This syntax is:

```
Header / Title [icon]
```

For example, one of the actions from the screenshot above are specified via:

```
- entrypoint: sgfs.commands.launch_maya:sgaction
  title: "3D Department / Launch Maya [application-osx-terminal]"
  folder: "Toolbox [cog]"
  list_order: 11
  entity_types: [Task]
  selection_required: true
```

For even more graceful degradation, you can give a "rich" title, heading, and icon that should be used, while falling back onto the original title if the browser extension fails:

```
- entrypoint: sgfs.commands.launch_maya:sgaction
  title: "Launch Maya"
  folder: "Toolbox [cog]"
  rich:
    title: Launch Maya
    header: 3D Department
    icon: application-osx-terminal
  list_order: 11
  entity_types: [Task]
  selection_required: true
```

This does not work for URL-based ActionMenuItems.

We currently support the Silk icon set; simply replace underscores with dashes in the icon names.

CHAPTER 3

Deployment

Deploying actions is a matter of calling the deployment script and giving it a configuration file:

```
python -m sgactions.deploy my_actions.yml
```

This will install new actions, and update old ones (comparing them by their entrypoint). You must manually delete them via:

```
python -m sgactions.deploy --list
python -m sgactions.deploy --delete <entrypoint or id from previous list>
```

CHAPTER 4

Writing Handlers

Action handlers are any Python callable that accept keyword arguments. The following is a simple example:

```
def sgaction(**kwargs):
    for item in sorted(kwargs.iteritems()):
        print '%s = %r' % item
```

and on a Task on our testing server outputs:

```
cols = ['content', 'step', 'sg_sort_order', 'task_assignees', 'sg_status_list', 'start_date', 'due_da
column_display_names = ['Task Name', 'Pipeline Step', 'Sort Order', 'Assigned To', 'Status', 'Start',
entity_type = 'Task'
ids = [43588, 43587, 43590]
page_id = 992
project_id = 66
project_name = 'Testing Sandbox'
referrer_path = '/detail/Shot/5773'
selected_ids = [43589]
server_hostname = '<snipped>'
session_uuid = '<snipped>'
sort_column = 'sg_sort_order'
sort_direction = 'asc'
title = 'Shot'
user_id = 108
user_login = '<snipped>'
```

Of particular interest is `project_id`, which will appear in most cases and so may appear to be a constant argument, but it will not be passed along from cross-project pages, such as user pages.

CHAPTER 5

Debugging

Standard output and error are dumped to a file in `/var/tmp` that starts with `sgactions`.