
signac-dashboard Documentation

Release 0.2.2

Bradley Dice

Apr 26, 2019

Contents

| | | |
|----------|-----------------------------------|-----------|
| 1 | Contents | 3 |
| 1.1 | Installation | 3 |
| 1.2 | API Reference | 4 |
| 1.3 | Changes | 10 |
| 1.4 | Support and Development | 13 |
| 2 | Indices and tables | 15 |
| | Python Module Index | 17 |

Note: This is documentation for the **signac-dashboard** package, which is part of the **signac** framework. See [here](#) for a comprehensive introduction to the **signac** *framework*.

1.1 Installation

The recommended installation method for **signac-dashboard** is via `conda` or `pip`. The software is tested for Python versions 3.4+. Its primary dependencies are `signac` and `flask`.

1.1.1 Install with conda

You can install **signac-dashboard** via `conda` (available on the `conda-forge` channel), with:

```
$ conda install -c conda-forge signac-dashboard
```

All additional dependencies will be installed automatically. To upgrade the package, execute:

```
$ conda update signac-dashboard
```

1.1.2 Install with pip

To install the package with the package manager `pip`, execute

```
$ pip install signac-dashboard --user
```

Note: It is highly recommended to install the package into the user space and not as superuser!

To upgrade the package, simply execute the same command with the `--upgrade` option.

```
$ pip install signac-dashboard --user --upgrade
```

1.1.3 Source Code Installation

Alternatively you can clone the [git repository](#) and execute the `setup.py` script to install the package.

```
git clone https://github.com/glotzerlab/signac-dashboard.git
cd signac-dashboard
git submodule update --init # This step is required!
python setup.py install --user
```

1.2 API Reference

This is the API for the **signac-dashboard** application.

1.2.1 The Dashboard

Attributes

| | | |
|---|------------------------|---|
| <code>Dashboard.add_url(import_name[, ...])</code> | <code>url_rules</code> | Add a route to the dashboard. |
| <code>Dashboard.job_sorter(job)</code> | | Override this method for custom job sorting. |
| <code>Dashboard.job_subtitle(job)</code> | | Override this method for custom job subtitles. |
| <code>Dashboard.job_title(job)</code> | | Override this method for custom job titles. |
| <code>Dashboard.main()</code> | | Runs the command line interface. |
| <code>Dashboard.register_module_asset(asset)</code> | | Register an asset required by a dashboard module. |
| <code>Dashboard.run(*args, **kwargs)</code> | | Runs the dashboard webserver. |

class `signac_dashboard.Dashboard` (`config={}`, `project=None`, `modules=[]`)
A dashboard application to display a `signac.Project`.

The `Dashboard` class is designed to be used as a base class for a child class such as `MyDashboard` which can be customized and launched via its command line interface (CLI). The CLI is invoked by calling `main()` on an instance of this class.

Configuration options: The `config` dictionary recognizes the following options:

- **HOST:** Sets binding address (default: localhost).
- **PORT:** Sets port to listen on (default: 8888).
- **DEBUG:** Enables debug mode if `True` (default: `False`).
- **PROFILE:** Enables the profiler `werkzeug.middleware.profiler.ProfilerMiddleware` if `True` (default: `False`).
- **PER_PAGE:** Maximum number of jobs to show per page (default: 25).

Parameters

- **config** (`dict`) – Configuration dictionary (default: `{}`).
- **project** (`signac.Project`) – signac project (default: `None`, autodetected).
- **modules** (`list`) – List of `Module` instances to display.

add_url (*import_name*, *url_rules=[]*, *import_file='signac_dashboard'*, ***options*)

Add a route to the dashboard.

This method allows custom view functions to be triggered for specified routes. These view functions are imported lazily, when their route is triggered. For example, write a file `my_views.py`:

```
def my_custom_view (dashboard) :
    return 'This is a custom message.'
```

Then, in `dashboard.py`:

```
from signac_dashboard import Dashboard

class MyDashboard (Dashboard) :
    pass

if __name__ == '__main__':
    dashboard = MyDashboard()
    dashboard.add_url ('my_custom_view', url_rules=['/custom-url'],
                      import_file='my_views')
    dashboard.main()
```

Finally, launching the dashboard with `python dashboard.py run` and navigating to `/custom-url` will show the custom message. This can be used in conjunction with user-provided jinja templates and the method `flask.render_template()` for extending dashboard functionality.

Parameters

- **import_name** (*str*) – The view function name to be imported.
- **url_rules** (*list*) – A list of URL rules, see `flask.Flask.add_url_rule()`.
- **import_file** (*str*) – The module from which to import (default: `'signac_dashboard'`).
- ****options** – Additional options to pass to `flask.Flask.add_url_rule()`.

job_sorter (*job*)

Override this method for custom job sorting.

This method returns a key that can be compared to sort jobs. By default, the sorting key is based on `Dashboard.job_title()`, with natural sorting of numbers. Good examples of such keys are strings or tuples of properties that should be used to sort.

Parameters **job** (`signac.contrib.job.Job`) – The job being sorted.

Returns Key for sorting.

Return type any comparable type

job_subtitle (*job*)

Override this method for custom job subtitles.

This method generates job subtitles. By default, the subtitle is a minimal unique substring of the job id.

Parameters **job** (`signac.contrib.job.Job`) – The job being subtitled.

Returns Subtitle to be displayed.

Return type `str`

job_title (*job*)

Override this method for custom job titles.

This method generates job titles. By default, the title is a pretty (but verbose) form of the job state point, based on the project schema.

Parameters `job` (`signac.contrib.job.Job`) – The job being titled.

Returns Title to be displayed.

Return type `str`

main()

Runs the command line interface.

Call this function to use signac-dashboard from its command line interface. For example, save this script as `dashboard.py`:

```
from signac_dashboard import Dashboard

class MyDashboard(Dashboard):
    pass

if __name__ == '__main__':
    MyDashboard().main()
```

Then the dashboard can be launched with:

```
python dashboard.py run
```

register_module_asset (`asset`)

Register an asset required by a dashboard module.

Some modules require special scripts or stylesheets, like the `signac_dashboard.modules.Notes` module. It is recommended to use a namespace for each module that matches the example below:

```
dashboard.register_module_asset({
    'file': 'templates/my-module/js/my-script.js',
    'url': '/module/my-module/js/my-script.js'
})
```

Parameters `asset` (`dict`) – A dictionary with keys 'file' and 'url'.

run (`*args`, `**kwargs`)

Runs the dashboard webserver.

Use `main()` instead of this method for the command-line interface. Arguments to this function are passed directly to `flask.Flask.run()`.

1.2.2 Dashboard Modules

| | |
|---|---|
| <code>Module</code> (<code>name</code> , <code>context</code> , <code>template</code> [, <code>enabled</code>]) | Base class for dashboard modules. |
| <code>modules.DocumentEditor</code> (<code>[name, context, template]</code>) | Provides an interface to edit the job document. |
| <code>modules.DocumentList</code> (<code>[name, context, ...]</code>) | Displays the job document. |
| <code>modules.FileList</code> (<code>[name, context, template, ...]</code>) | Lists files in the job workspace with download links. |
| <code>modules.FlowStatus</code> (<code>[name, context, ...]</code>) | Show job labels from a <code>flow.FlowProject</code> . |
| <code>modules.ImageViewer</code> (<code>[name, context, ...]</code>) | Displays images in the job workspace that match a glob. |
| <code>modules.Notes</code> (<code>[name, context, template, key]</code>) | Displays a text box that is synced to the job document. |

Continued on next page

Table 2 – continued from previous page

| | |
|--|---|
| <code>modules.StatepointList([name, context, template])</code> | Displays the job state point. |
| <code>modules.TextDisplay([name, message, markdown])</code> | Render custom text or Markdown in a card. |
| <code>modules.VideoViewer([name, context, ...])</code> | Displays videos in the job workspace that match a glob. |

class `signac_dashboard.Module` (*name, context, template, enabled=True*)

Base class for dashboard modules.

Modules provide *cards* of content, for a specific *context*. Each module must have a **name** which appears in its cards' titles, a **context** (such as `'JobContext'`) in which its contents will be displayed, and a **template** file (written in HTML/Jinja-compatible syntax) for rendering card content. Modules can be disabled by default, by setting `enabled=False` in the constructor.

Custom modules: User-defined module classes should be a subclass of `Module` and define the function `get_cards()`. See [this example](#).

Module assets: If a module requires scripts or stylesheets to be included for its content to be rendered, they must be handled by the callback `register()`. For example, a module inheriting from the base `signac_dashboard.Module` class may implement this by overriding the default method as follows:

```
def register(self, dashboard):
    assets = ['js/my-script.js', 'css/my-style.css']
    for asset in assets:
        dashboard.register_module_asset({
            'file': 'templates/my-module/{}'.format(asset),
            'url': '/module/my-module/{}'.format(asset)
        })
```

Then, when the module is active, its assets will be included and a route will be created that returns the asset file.

Module routes: The callback `register()` allows modules to implement custom routes, such as methods that should be triggered by POST requests or custom APIs. For example, a module inheriting from the base `signac_dashboard.Module` class may implement this by overriding the default method as follows:

```
def register(self, dashboard):
    @dashboard.app.route('/module/my-module/update', methods=['POST'])
    def my_module_update():
        # Perform update
        return "Saved."
```

Parameters

- **name** (*str*) – Name of this module (appears in card titles).
- **context** (*str*) – Context in which this module's cards should be displayed (e.g. `'JobContext'`).
- **template** (*str*) – Path to a template file for this module's cards (e.g. `cards/my_module.html`, without the template directory prefix `templates/`).

disable()

Disable this module.

enable()

Enable this module.

get_cards ()

Returns this module's cards for rendering.

The cards are returned as a list of dictionaries with keys 'name' and 'content'.

Returns List of module cards.

Return type list

register (*dashboard*)

Callback to register this module with the dashboard.

This callback should register assets and routes, as well as any other initialization that accesses or modifies the dashboard.

Parameters **dashboard** (*signac_dashboard.Dashboard*) – The dashboard invoking this callback method.

toggle ()

Toggle this module.

```
class signac_dashboard.modules.DocumentEditor (name='Document Editor',  
context='JobContext', tem-  
plate='cards/document_editor.html',  
**kwargs)
```

Provides an interface to edit the job document.

This module shows keys in the job document with a form that allows users to edit their contents. When saving, the edited strings are parsed into JSON-compatible Python data structures (e.g., list and dict). Job document keys beginning with an underscore _ are treated as private and are not displayed.

```
class signac_dashboard.modules.DocumentList (name='Job Document', con-  
text='JobContext', tem-  
plate='cards/document_list.html',  
max_chars=None, **kwargs)
```

Displays the job document.

Long values can be optionally truncated.

Parameters **max_chars** (*int*) – Truncation length for document values (default: None).

```
class signac_dashboard.modules.FileList (name='File List', context='JobContext', tem-  
plate='cards/file_list.html', prefix_jobid=True,  
**kwargs)
```

Lists files in the job workspace with download links.

Parameters **prefix_jobid** (*bool*) – Whether filenames should be prefixed with the job id when being downloaded (default: True).

```
class signac_dashboard.modules.FlowStatus (name='Flow Status', context='JobContext',  
template='cards/flow_status.html',  
project_module='project',  
project_class='Project', **kwargs)
```

Show job labels from a `flow.FlowProject`.

This module displays a card with labels from `flow.FlowProject.labels ()`. The user must provide an instance of `flow.FlowProject` to the dashboard constructor. Example:

```
from project import Project # FlowProject subclass with labels
from signac_dashboard import Dashboard

if __name__ == '__main__':
    Dashboard(project=Project()).main()
```

```
class signac_dashboard.modules.ImageViewer (name='Image Viewer', context='JobContext',
                                             template='cards/image_viewer.html',
                                             img_globs=['*.png', '*.jpg', '*.gif'],
                                             **kwargs)
```

Displays images in the job workspace that match a glob.

This module can display images in any format that works with a standard `` tag. The module defaults to showing all images of PNG, JPG, or GIF types. A filename or glob can be defined to select specific filenames. Multiple `ImageViewer` modules can be defined with different filenames or globs to enable/disable cards for each image or image group. Examples:

```
from signac_dashboard.modules import ImageViewer
img_mod = ImageViewer() # Shows all PNG/JPG/GIF images
img_mod = ImageViewer(name='Bond Order Diagram', img_globs=['bod.png'])
```

Parameters `img_globs` (*list*) – A list of glob expressions or exact filenames to be displayed, one per card (default: `['*.png', '*.jpg', '*.gif']`).

```
class signac_dashboard.modules.Notes (name='Notes', context='JobContext',
                                         template='cards/notes.html', key='notes', **kwargs)
```

Displays a text box that is synced to the job document.

The contents of the text box are saved to `job.document['notes']`. The `Notes` module can be used to annotate a large data space with tags or human-readable descriptions for post-processing, parsing, or searching.

Parameters `key` (*str*) – Document key to display and update (default: `'notes'`).

```
class signac_dashboard.modules.StatepointList (name='Statepoint Parameters',
                                                  context='JobContext',
                                                  template='cards/statepoint_list.html',
                                                  **kwargs)
```

Displays the job state point.

```
class signac_dashboard.modules.TextDisplay (name='Text Display', message=<function
                                             TextDisplay.<lambda>>, markdown=False,
                                             **kwargs)
```

Render custom text or Markdown in a card.

This module calls a user-provided function to display text or Markdown content in a card. Rendering Markdown requires the `markdown` library to be installed. Example:

```
def my_text(job):
    return 'This job id is {}'.format(str(job))

modules = [TextDisplay(message=my_text)]
```

Parameters

- **message** (*callable*) – A callable accepting one argument of type `signac.contrib.job.Job` and returning text or Markdown content.
- **markdown** (*bool*) – Enables Markdown rendering if True (default: False).

```
class signac_dashboard.modules.VideoViewer (name='Video Viewer', context='JobContext',
                                              template='cards/video_viewer.html',
                                              video_globs=['*.mp4', '*.m4v'],
                                              preload='none', poster=None, **kwargs)
```

Displays videos in the job workspace that match a glob.

The `VideoViewer` module displays videos using an HTML `<video>` tag. The module defaults to showing all videos of MP4 or M4V types. A filename or glob can be defined to select specific filenames, which may be of any format supported by your browser with the `<video>` tag. A “poster” can be defined, which shows a thumbnail with that filename before the video is started. Videos do not preload by default, since file sizes can be large and there may be many videos on a page. To enable preloading, use the argument `preload='auto'` or `preload='metadata'`. Multiple `VideoViewer` modules can be defined with different filenames or globs to enable/disable cards individually. Examples:

```
from signac_dashboard.modules import VideoViewer
video_mod = VideoViewer() # Shows all MP4/M4V videos
video_mod = VideoViewer(name='Cool Science Video',
                        video_globs=['cool_science.mp4'],
                        poster='cool_science_thumbnail.jpg',
                        preload='none')
```

Parameters

- **video_globs** (*list*) – A list of glob expressions or exact filenames to be displayed, one per card (default: `['*.mp4', '*.m4v']`).
- **preload** (*str*) – Option for preloading videos, one of `'auto'`, `'metadata'`, or `'none'` (default: `'none'`).
- **poster** (*str*) – A path in the job workspace for a poster image to be shown before a video begins playback (default: `None`).

1.3 Changes

The `signac-dashboard` package follows [semantic versioning](#).

1.3.1 Version 0.2

[0.2.2] – 2019-04-25

Fixed

- Resolved issue with enabling/disabling modules.
- Long words in card content will now wrap.

[0.2.1] – 2019-04-24

Fixed

- Corrected PyPI deployment.

[0.2.0] – 2019-04-24

Added

- New modules: `DocumentEditor`, `FlowStatus`, `TextDisplay`.

- New examples: cli, custom-modules, document-editor, flow-status, plots.
- The default job sorter uses natural sorting for numbers via natsort.

Changed

- Improved API documentation, especially for modules.
- Unified module asset/route registration into one `register` method.
- The Notes module can be used with any job document key.

Fixed

- Corrected error in VideoViewer when no poster was provided.
- ImageViewer/VideoViewer match files in job workspace subdirectories.
- Files can now be retrieved from job workspace subdirectories.
- Corrected pagination error.

Removed

- The signac project document and user session are no longer used to store module settings. The dashboard user script is the single source for all configuration besides command line arguments.
- Job labels have been removed and replaced with the FlowStatus module.

1.3.2 Version 0.1

[0.1.6] – 2018-10-09

Changed

- Updated layouts for bulma 0.7.1.

Fixed

- PyPI upload was missing bulma and couldn't run.

Removed

- `cssmin` is no longer a dependency.

[0.1.5] – 2018-10-09

Added

- Example dashboards are in the `examples` folder.
- Console entry point, `signac-dashboard run` will launch a simple dashboard. This will be extended in a future release.
- Better support for custom module assets.

Changed

- Modules are now part of the user session and are saved to the project document. This requires all module arguments to be JSON-encodable.
- Restructured module design and how modules provide assets.
- Documentation has been updated, with instructions for port forwarding.
- Split views into a separate file.

Fixed

- Browsers will no longer cache dynamic content.
- Pagination rendering bug squashed.

[0.1.4] – 2018-07-23

Fixed

- Removed `flask_cache` and replaced with `lru_cache` to fix compatibility with Flask 1.0.

[0.1.3] – 2018-04-02

Added

- README documentation on searching.
- Added support for signac cache.
- Added LRU cache for job details.
- Added pagination support for much faster loading.
- Added VideoViewer module.

Changed

- Error handling is cleaner.
- Refactored job views.
- Job titles show statepoint booleans as `True/False`.

- Added job id prefix to downloaded filenames so they can be distinguished.

Fixed

- Heterogeneous schemas can generate job titles.

[0.1.2] – 2018-02-08

Added

- Search jobs with a document filter with `doc:{"key": "value"}`.
- Unit tests for job search.
- README documentation is much more complete.
- flake8 checked in CI.
- Added CHANGELOG.

[0.1.1] – 2017-09-25

Added

- Continuous integration support.
- Added first unit test.
- Mobile support is significantly improved.

Fixed

- Flask package was incorrectly configured in the previous release.
- Corrected menu activation script to trigger on `turbolinks:load` event.

[0.1.0] – 2017-09-17

Added

- First release.

1.4 Support and Development

To get help using the **signac-dashboard** package, either send an email to signac-support@umich.edu or join the [signac gitter chatroom](#).

The **signac-dashboard** package is hosted on [GitHub](#) and licensed under the open-source BSD 3-Clause license. Please use the [repository's issue tracker](#) to report bugs or request new features.

CHAPTER 2

Indices and tables

- `genindex`
- `modindex`
- `search`

S

`signac_dashboard.modules`, 8

A

`add_url()` (*signac_dashboard.Dashboard* method), 4

D

`Dashboard` (class in *signac_dashboard*), 4

`disable()` (*signac_dashboard.Module* method), 7

`DocumentEditor` (class in *signac_dashboard.modules*), 8

`DocumentList` (class in *signac_dashboard.modules*), 8

E

`enable()` (*signac_dashboard.Module* method), 7

F

`FileList` (class in *signac_dashboard.modules*), 8

`FlowStatus` (class in *signac_dashboard.modules*), 8

G

`get_cards()` (*signac_dashboard.Module* method), 7

I

`ImageViewer` (class in *signac_dashboard.modules*), 8

J

`job_sorter()` (*signac_dashboard.Dashboard* method), 5

`job_subtitle()` (*signac_dashboard.Dashboard* method), 5

`job_title()` (*signac_dashboard.Dashboard* method), 5

M

`main()` (*signac_dashboard.Dashboard* method), 6

`Module` (class in *signac_dashboard*), 7

N

`Notes` (class in *signac_dashboard.modules*), 9

R

`register()` (*signac_dashboard.Module* method), 8

`register_module_asset()` (*signac_dashboard.Dashboard* method), 6

`run()` (*signac_dashboard.Dashboard* method), 6

S

`signac_dashboard.modules` (module), 8

`StatepointList` (class in *signac_dashboard.modules*), 9

T

`TextDisplay` (class in *signac_dashboard.modules*), 9

`toggle()` (*signac_dashboard.Module* method), 8

V

`VideoViewer` (class in *signac_dashboard.modules*), 9