django-sphinx-hosting

Release 1.3.1

Caltech IMSS ADS

Aug 16, 2023

OVERVIEW

CHAPTER

ONE

USER AUTHORIZATION

`django-sphinx-hosting` uses Django model permissions to restrict access to the various views within it.

We provide Django groups to which you can assign your users to grant them different levels of privileges. Users who are assigned to none of these groups are `Viewers`: they can search and read the documentation sets within, but they cannot create, modify or delete anything.

1.1 Administrators

Users in the `Administrators` group have full privileges within the system.

- Create, edit, delete *sphinx_hosting.models.Project* objects
- Create, edit, delete *sphinx_hosting.models.Version* objects
- Create, edit, delete *sphinx_hosting.models.Classifier* objects

1.2 Editors

Users in the `Editors` group can work with projects and versions but have no rights to manage *sphinx_hosting.models.Classifier* objects.

- Create, edit, delete *sphinx_hosting.models.Project* objects
- Create, edit, delete *sphinx_hosting.models.Version* objects

1.3 Project Managers

Users in the `Project Managers` group can only manage projects.

1.4 Version Managers

Users in the `Version Managers` group can only manage versions.

1.5 Classifier Managers

Users in the `Classifier Managers` group can only manage classiiers.

CHAPTER

TWO

CONFIGURING YOUR SPHINX PROJECT

`django-sphinx-hosting` expects your Sphinx docs to be in a specific format to be able to be imported, and to be built with specific sphinx extensions. On this page, we describe how to configure your Sphinx project appropriately.

2.1 Sphinx conf.py settings

2.1.1 project

To import a documentation set, there must be a *sphinx_hosting.models.Project* in the database whose `machine_name` matches the `project` in Sphinx's `conf.py` config file for the docs to be imported. The `machine_name` for a project is set at project create time within `django-sphinx-hosting`.

Create a project by navigating to the "Projects" page and clicking the "Create Project" button. You'll be asked for a human name, a machine name and a description. Whatever you use for your version control repository name is a good choice for `Machine Name`.

2.1.2 release

The `release` in the `conf.py` will be used to create or update a *sphinx_hosting.models.Version*. We will set *sphinx_hosting.models.Version.version* to the value of `release`.

2.1.3 extensions

sphinx_rtd_theme [required]

Miminally, you must use the Sphinx ReadTheDocs theme when packaging your documentation. The importers, views and stylesheets inside django-sphinx-hosting depend on the HTML structure, Javascript and CSS classes that that theme provides.

Ensure that you have `html_theme_options["collapse_navigation"]` set to `False`, otherwise your auto-built navigation within django-sphinx-hosting may look wrong.

```
extensions = [
    'sphinx_rtd_theme',
    ...
]

html_theme = 'sphinx_rtd_theme'
html_theme_options = {
```

(continues on next page)

(continued from previous page)

```
    "collapse_navigation": False
}
```

sphinxcontrib-jsonglobaldoc [optional]

If you have a complex page hierarchy in your documentation, you may benefit from sphinxcontrib-jsonglobaltoc. This extension extends `JSONHTMLBuilder` from `sphinxcontrib-serializinghtml` to add a `globaltoc` key to each `.fjson` file produced. `globaltoc` contains the HTML for the global table of contents for the entire set of documentation. This allows django-sphinx-hosting to more reliably build your navigation.

```
extensions = [
    'sphinx_rtd_theme',
    'sphinx_json_globaltoc'
    ...
]
```

CHAPTER

THREE

AUTHORING YOUR SPHINX PROJECT

When authoring your Sphinx project that will be imported into `django-sphinx-hosting`, the most important thing to be careful with is your global table of contents.

The global table of contents for a documentation set is something that the author of the Sphinx documents defines with `toctree` directives. Sphinx uses those `toctree` directives to construct the linkages between pages. This affects the following things within `django-sphinx-hosting`:

- The `Next`, `Previous` and `Parent` page buttons at the top of each page
- The navigation menu that appears in the main navigation sidebar

3.1 How the global table of contents is built

There are two methods for building the global table of contents navigation in the sidebar: use sphinxcontrib_jsonglobaltoc to add a `globaltoc` key to every `.ftjson` file created when doing `make json` from your source files; have `django-sphinx-contrib` build a global table of contents by starting at the root page and traversing the page tree via the `next` key in each `.ftjson` file.

The main reason to use sphinxcontrib_jsonglobaltoc over the traversal mechanism is so that the in-page anchors to page sections to show up in your sidebar navigation, or both. It also will obey the `:caption:` setting in your `toctree` directives.

3.2 How to make a good global table of contents

Our goal here is to make a global table of contents that looks good in the navigation sidebar of `django-sphinx-hosting`.

As we said above, it is the source `.rst` documents for the documentation set that determine the global table of contents, not `django-sphinx-hosting`. `django-sphinx-hosting` just interprets what Sphinx gives it, and uses that to build the main navigation in the sidebar and the the `Next`, `Previous` and `Parent` buttons at the top and bottom of each page.

3.2.1 Headings

You **MUST** get the heading levels right throughout your entire set of documentation if you want your global table of contents to look right.

First, let's review how to do headings in ReStructuredText, because it wil be really important in a minute. The Sphinx docs say:

> Normally, there are no heading levels assigned to certain characters as the structure is determined from the succession of headings.

The "is determined by from the succession of headings" is quite important and unfortunate here. Sphinx is overly forgiving where it might save a lot of heartache if it were to be a bit more draconian, and that can easily cause subtle problems in global table of contents creation.

Here is the Python Style Guide convention:

- Level 1: # under and overline, for parts
- Level 2: * under and overline, for chapters
- Level 3: = underline for sections
- Level 4: - underline for subsections
- Level 5: ^ underline for subsubsections
- Level 6: " underline for paragraphs

Note: Using Markdown with the `myst_parser` extension may make headings less easy to screw up, since Markdown has formal heading definitions, unlike ReStructuredText.

Guidelines:

- **Headings in the root page**: the document heading (the page title) on your root page **must be a level 1 heading**. If you have subsections in the root page, **make them level 3 headings or lower**. If you use level 2 headings on the root page, you'll compete with your page document headings, which should be level 2, and you'll get a mess in your navigation. If you're going to do nested `toctree` directives (see below), you may want subheadings on the root page to be level 4 or below.
- **Headings in all other pages**: pages under the root page must have a level 2 heading. In ReStructuredText that is * underline and overline. If you don't get the heading levels right, you end up with very odd nesting behavior in the resultant global table of contents.

3.2.2 toctree directives

`toctree` directives and only those directives determine the page/section hierarchy shown in the navigtion sidebar. Filesystem layout of your `.rst` documents has no impact on the global table of contents.

- You must put at least one `toctree` directive in your root page. This will form the root of your global table of contents.
- If you are using nested `toctree` directives on sub-pages, put your directive directly under the document heading on those sub-pages. Do this because, on sub-pages, the toctree recalibates the starting heading level for the pages it references to be relative to the **nearest preceding heading** for the `toctree`, not from the page heading for the page the directive is on.
- If all you're interested in for your global table of contents are the page titles, be sure to add `:titlesonly:` to your `toctree` directive.

- Unless you really want to show the global table of contents within the page contents in addition to the navigation sidebar, use the `:hidden:` parameter in your `toctree` directives.
- The `:caption:` parameter for a `toctree` directive only produces an actual caption if that directive **is on the root page**. `:caption:` parameters on sub-pages are ignored.
- You will only see captions in the `django-sphinx-hosting` if you used the `sphinxcontrib-jsonglobaltoc` extension when building your JSON package.

Now on to constructing your document hierarchy and `toctree` directives.

3.3 Examples

3.3.1 Single one-level toctree directive

If all you have is that single `toctree` directive in the root page of your documentation, then it's pretty difficult to make that not build and render properly.

Here's an example root page:

```
#######
My Book
#######

.. toctree::
:hidden:

chapter1
chapter2
chapter3

Introduction
============

Note this is under a level 3 heading, not a level 2.

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod
tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam,
quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo
consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse
cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat
non proident, sunt in culpa qui officia deserunt mollit anim id est laborum.
```

And here's `chapter1.rst`:

```
*********
Chapter 1
*********

.. toctree::
   :hidden:

page1
page2
```

(continues on next page)

(continued from previous page)

```
page3

Section 1
=========

Note that our document title is a level 2 heading, and here we are under a level
3 heading.

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod
tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam,
quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo
consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse
cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat
non proident, sunt in culpa qui officia deserunt mollit anim id est laborum.
```

3.3.2 Multiple one-level toctree directives

You may want multiple `toctree` directives in your root document so that you can separate pages into different logical sections at the same level, each with its own `:caption:`.

For example, here's `index.rst`, our root document:

```
#######
My Book
#######

.. toctree::
   :hidden:
   :caption: The first things

chapter1
chapter2
chapter3

.. toctree::
   :hidden:
   :caption: The second things

chapter4
chapter5
chapter6

Introduction
============

Note this is under a level 3 heading, not a level 2.

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod
tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam,
quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo
consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse
```

(continues on next page)

(continued from previous page)

```
cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat
non proident, sunt in culpa qui officia deserunt mollit anim id est laborum.
```

The `chapter1.rst` etc. pages should all follow the heading strategy in the example `chapter1.rst` in *Single one-level toctree directive*.

3.3.3 Nested toctree directives

Nested toctrees happen when you have a top level `toctree` directive in your root page and also `toctree` directives in child pages. You may want to do this because you have many pages in your set, and the navigation sidebar is getting too complicated to use as a flat set of links.

It is probably best to not go beyond two levels of `toctree` directives to avoid header collisions between document titles and subheadings on a page.

> **Warning:** If you are using the sphinxcontrib_jsonglobaltoc extension to build your JSON files, you may want to use the `:titlesonly:` parameter on your `toctree` directives to avoid mingling document titles with other headings at the same level. Mingling the document titles and subheadings makes the navigation.
>
> It is possible to make the global table of contents be sane without `:titlesonly:` but you do have to be very careful with your headings on all pages.

As an example of nested `toctree` direcrives here's our root document:

```
#######
My Book
#######

.. toctree::
   :hidden:
   :titlesonly:

chapter1
chapter2/index
chapter3

Introduction
------------

Note this is under a level 4 heading, not a level 2.  We need a level 4 here
because chapter2/index needs a level 2 heading as a document title, and
chapter2/section1 needs a level 3 heading as document title.   If we make our
subheading here be level 3, it will confuse the global table of contents by
putting "Introduction" and chapter2/section1 at the same level.

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod
tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam,
quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo
consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse
cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat
non proident, sunt in culpa qui officia deserunt mollit anim id est laborum.
```

Now let's say that `chapter2/index.rst` also has a `toctree` directive:

```
*********
Chapter 2
*********

.. toctree::
   :hidden:
   :titlesonly:

chapter2/section1
chapter2/section2
chapter2/section3

Introduction
------------

Note that our document title is a level 2 heading, and here we are under a
level 4 heading.

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod
tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam,
quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo
consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse
cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat
non proident, sunt in culpa qui officia deserunt mollit anim id est laborum.
```

Then this is what `chapter2/section1.rst` should look like:

```
Chapter 2, Section 1
====================

Introduction
------------

Note that our document title is a level 3 heading, and here we are under a
level 4 heading.

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod
tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam,
quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo
consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse
cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat
non proident, sunt in culpa qui officia deserunt mollit anim id est laborum.
```

CHAPTER

FOUR

IMPORTING YOUR SPHINX DOCS

Before importing your docs, ensure that you have configured your Sphinx project properly for `django-sphinx-hosting` by following the instructions on *Configuring your Sphinx project*.

4.1 Packaging

In order to be able to be imported into `django-sphinx-hosting`, you will need to publish your Sphinx docs as JSON files, and to bundle them in a specific way.

In your Sphinx docs folder, you will want to build your docs as `json`, not `html`.

Do either:

```
make json
```

or:

```
sphinx-build -n -b json build/json
```

To build the tarfile, the files in the tarfile should be contained in a folder. We want:

```
json/py-modindex.fjson
json/globalcontext.json
json/_static
json/last_build
json/genindex.fjson
json/objectstore.fjson
json/index.fjson
json/environment.pickle
json/searchindex.json
json/objects.inv
...
```

Not:

```
py-modindex.fjson
globalcontext.json
_static
last_build
genindex.fjson
index.fjson
```

(continues on next page)

(continued from previous page)

```
environment.pickle
searchindex.json
objects.inv
...
```

Here's how you do that:

```
$ cd build
$ tar zcf mydocs.tar.gz json
```

Now you can import `mydocs.tar.gz` into `django-sphinx-hosting`.

4.2 Importing

There are three ways to import your package into `django-sphinx-hosting`:

- Use the upload form on the project's detail page.
- Use the API endpoint `/api/v1/version/`.
- Use the `import_docs` Django management command.

4.2.1 The upload form

To use the upload form, browse to the project detail page of the project whose docs you want to import, and use the form titled "Import Docs" in the "Actions" column along the left side of the page.

Note: You must have the `sphinxhostingcore.change_project` Django permission or be a Django superuser in order to use the upload form. Either assign that directly to your Django user object, or use assign your user to either the "Administrators" or "Editors" Django groups to get that permission. See *User authorization*

4.2.2 Use the API endpoint

To upload your docs package via the API, you must submit as form-data, with a single key named `file`, and with the `Content-Disposition` header like so:

```
Content-Disposition: attachment;filename=mydocs.tar.gz
```

The filename you pass in the `Content-Disposition` header does not matter and is not used; set it to whatever you want.

To upload a file with `curl` to the endpoint for this view:

```
curl \
    -XPOST \
    -H "Authorization: Token __THE_API_TOKEN__" \
    -F 'file=@path/to/yourdocs.tar.gz' \
    https://sphinx-hosting.example.com/api/v1/version/import/
```

4.2.3 The `import_docs` management command

Load your tarfile into the database:

```
$ ./manage.py import_docs mydocs.tar.gz
```

To load the export and overwite any existing Sphinx pages in the database with that in the tarfile:

```
$ ./manage.py import_docs --force mydocs.tar.gz
```

CHAPTER

FIVE

THE DJANGO-SPHINX-HOSTING REST API

`django-sphinx-hosting` provides a REST API for interacting with the the application in a programmatic way. The API is implemented using Django REST Framework.

See *Configure Django REST Framework* for instructions on how generally to configure your `settings.py` file to use DRF for our API.

5.1 How to reach the API

The API is reachable at the following path of your install: `/api/v1/`. See *REST API* for the description of all endpoints.

5.2 Authentication

It's up to you to provide an authentication mechanism for the API via the `REST_FRAMEWORK` setting in your `settings.py` file. `django-sphinx-hosting` will use whatever you provide for the `DEFAULT_AUTHENTICATION_CLASSES` setting.

See the Django REST Framework: Authentication for more information on how to configure authentication for DRF.

5.2.1 Example

Here's an example of how to configure the API to use Token based authentication:

```
INSTALLED_APPS = [
    ...
    'rest_framework.authtoken',
    ...
]

REST_FRAMEWORK = {
    'DEFAULT_AUTHENTICATION_CLASSES': ('rest_framework.authentication.TokenAuthentication
↪',),
    # https://www.django-rest-framework.org/api-guide/parsers/#setting-the-parsers
    'DEFAULT_PARSER_CLASSES': ('rest_framework.parsers.JSONParser',),
    # https://django-filter.readthedocs.io/en/master/guide/rest_framework.html
    'DEFAULT_FILTER_BACKENDS': ('django_filters.rest_framework.DjangoFilterBackend',),
}
```

Note: We always need at least the `DEFAULT_PARSER_CLASSES` setting and the `DEFAULT_FILTER_BACKENDS` listed above for the API to work at all, regardless of the authentication mechanism you choose, so be sure to include them.

Then migrate the database to create the `Token` model:

```
$ python manage.py migrate
```

And then create a token for your user:

```
$ python manage.py drf_create_token <username>
```

To use this token, you must provide it in the `Authorization` header of your requests. Example:

```
$ curl -X GET \\
    -H 'Accept: application/json; indent=4' \\
    -H 'Authorization: Token __THE_TOKEN__' \\
    --insecure \\
    --verbose \\
    https://localhost/api/v1/projects/
```

5.3 Authoriztion

The API endpoints all require that the user be authenticated. All users have read-only access to all API endpoints, but for write access, they must be in the appropriate group or groups from *User authorization*.

CHAPTER

SIX

RUNBOOK

6.1 Contributing

6.1.1 Instructions for contributors

Make a clone of the github repo:

```
$ git clone https://github.com/caltechads/django-sphinx-hosting
```

Workflow is pretty straightforward:

1. Make sure you are reading the latest version of this document.
2. Setup your machine with the required development environment
3. Make your changes.
4. Ensure your changes work by running the demo app in `sandbox`.
5. Update the documentation to reflect your changes.
6. Commit changes to your branch.

6.1.2 Preconditions for working on django-sphinx-hosting

Python environment

The Amazon Linux 2 base image we use here for our sandbox service has Python 3.10.12, so we'll want that in our virtualenv.

Here is an example of using `pyenv` to make your virtualenv:

```
$ cd django-sphinx-hosting
$ pyenv virtualenv 3.10.12 django-sphinx-hosting
$ pyenv local django-sphinx-hosting
$ pip install --upgrade pip wheel
```

If you don't have a *pyenv* python 3.10.12 built, build it like so:

```
$ pyenv install 3.10.12
```

After that please install libraries required for working with the code and building the documentation.

```
$ pip install -r requirements.txt
```

Docker

Our current base image requires you to authenticate to AWS Public ECR in order to pull it. Do:

```
$ aws ecr-public get-login-password --region us-east-1 | docker login --username AWS --
↪password-stdin public.ecr.aws
$ docker pull public.ecr.aws/m3v9w5i2/caltech-imss-ads/amazonlinux2-python3.10
```

6.1.3 Running the local docker stack

Build the Docker image

```
$ cd sandbox
$ make build
```

Run the service and initialize the database

The first time you run the stack only, do:

```
$ docker-compose up mysql
```

Wait for the database to initialize itself, then stop the mysql container by doing ^C.

```
$ make dev
```

This will start the service and apply alll the Django database migrations.

Getting to the demo in your browser

Browse to https://localhost to get to the demo application.

There are 3 users available:

- `admin` with password `testy`: This user is in the `Administrators` Django group. This user can do anything.
- `editor` with password `testy`: This user is in the `Project Managers` and `Version Managers` Django groups. This user can do anything except manage `Classifiers`.
- `viewer` with password `testy`: This user is in no groups. This user can only view the documentation.

The `demo` container is running with `gunicorn` reload-on-change enabled, so you may edit files and see the changes reflected in the browser witout having to restart the container.

CHAPTER

SEVEN

MODELS

7.1 Fields

class sphinx_hosting.fields.**MachineNameField**(**args*, *max_length=50*, *db_index=True*, *allow_unicode=False*, ***kwargs*)

This is just a `django.forms.SlugField` that also allows "." characters. "." is not uncommon in some project names, especially if the project is named after the website domain it hosts.

formfield(***kwargs*)

Return a django.forms.Field instance for this field.

7.2 Managers

class sphinx_hosting.models.**ClassifierManager**(**args*, ***kwargs*)

tree() → Dict[str, *ClassifierNode*]

Given our classifiers, which are `::` separated lists of terms like:

```
Section :: Subsection :: Name
Section :: Subsection :: Name2
Section :: Subsection :: Name3
Section :: Subsection
```

Return a tree-like data structure that looks like:

```
{
    'Section': ClassifierNode(
        title='Section'
        items={
            'Subsection': ClassifierNode(
                title='Subsection',
                classifier=Classifier(name="Section :: Subsection"),
                items: {
                    'Name': ClassifierNode(
                        title='Name',
                        classifier=Classifier(
                            name='Section :: Subsection :: Name'
                        )
                    ),
```

(continues on next page)

(continued from previous page)

```
                    ...
                }
            )
        }
    )
}
```

7.3 Models

class sphinx_hosting.models.**Classifier**(**args*, ***kwargs*)

Database table: sphinxhostingcore_classifier

A *Project* can be tagged with one or more *Classifier* tags. This allows you to group projects by ecosystem, or type, for example.

Use PyPI classifiers as an example of how to use a single field for classifying across many dimensions.

Examples:

```
Ecosystem :: CMS
Language :: Python
Owner :: DevOps :: AWS
```

Parameters

- **id** (*AutoField*) – Primary key: ID
- **name** (*CharField*) – Classifier Name. The classifier spec for this classifier, e.g. "Language :: Python"

Reverse relationships:

Parameters

projects (Reverse ManyToManyField from *Project*) – All projects of this classifier (related name of *classifiers*)

exception DoesNotExist

exception MultipleObjectsReturned

save(**args*, ***kwargs*) → None

Overrides django.db.models.Model.save.

Override save to create any missing classifiers in our chain. For example, if we want to create this classifier:

```
Foo :: Bar :: Baz
```

But `Foo :: Bar` does not yet exist in the database, create that before creating `Foo :: Bar :: Baz`. We do this so that when we filter our projects by classifier, we can filter by `Foo :: Bar` and `Foo :: Bar :: Baz`.

id

Type: AutoField

Primary key: ID

A wrapper for a deferred-loading field. When the value is read from this

name: Field

Type: CharField

Classifier Name. The classifier spec for this classifier, e.g. "Language :: Python"

A wrapper for a deferred-loading field. When the value is read from this

objects = <sphinx_hosting.models.ClassifierManager object>

projects

Type: Reverse ManyToManyField from *Project*

All projects of this classifier (related name of *classifiers*)

Accessor to the related objects manager on the forward and reverse sides of a many-to-many relation.

In the example:

```
class Pizza(Model):
    toppings = ManyToManyField(Topping, related_name='pizzas')
```

`Pizza.toppings` and `Topping.pizzas` are `ManyToManyDescriptor` instances.

Most of the implementation is delegated to a dynamically defined manager

class sphinx_hosting.models.**Project**(**args*, ***kwargs*)

Database table: sphinxhostingcore_project

A Project is what a set of Sphinx docs describes: an application, a library, etc.

Projects have versions (*Version*) and versions have Sphinx pages (*SphinxPage*).

Parameters

- **id** (*AutoField*) – Primary key: ID
- **created** (*CreationDateTimeField*) – Created
- **modified** (*ModificationDateTimeField*) – Modified
- **title** (*CharField*) – Project Name. The human name for this project

  The page title
- **description** (*CharField*) – Brief Description. A brief description of this project
- **machine_name** (MachineNameField) – Machine Name. Must be unique. Set this to the slugified value of "project" in Sphinx's. conf.py

Relationship fields:

Parameters

- **permission_groups** (ManyToManyField to ProjectPermissionGroup) – Permission groups (related name: projects)
- **classifiers** (ManyToManyField to *Classifier*) – Classifiers (related name: *projects*)

Reverse relationships:

Parameters

- **versions** (Reverse ForeignKey from *Version*) – All versions of this project (related name of *project*)

- **related_links** (Reverse ForeignKey from *ProjectRelatedLink*) – All related links of this project (related name of *project*)

exception DoesNotExist

exception MultipleObjectsReturned

get_absolute_url() → str

Return the standard URL for viewing/editing this instance of this model.

Returns
The update URL for this instance.

get_latest_version_url() → Optional[str]

get_next_by_created(*, *field=<django_extensions.db.fields.CreationDateTimeField: created>*, *is_next=True*, ***kwargs*)

Finds next instance based on *created*. See get_next_by_FOO for more information.

get_next_by_modified(*, *field=<django_extensions.db.fields.ModificationDateTimeField: modified>*, *is_next=True*, ***kwargs*)

Finds next instance based on *modified*. See get_next_by_FOO for more information.

get_previous_by_created(*, *field=<django_extensions.db.fields.CreationDateTimeField: created>*, *is_next=False*, ***kwargs*)

Finds previous instance based on *created*. See get_previous_by_FOO for more information.

get_previous_by_modified(*, *field=<django_extensions.db.fields.ModificationDateTimeField: modified>*, *is_next=False*, ***kwargs*)

Finds previous instance based on *modified*. See get_previous_by_FOO for more information.

get_update_url() → str

Return a URL suitable for POSTing to to update this instance of this model.

Returns
The update URL for this instance.

classifiers: ManyToManyField

Type: ManyToManyField to *Classifier*

Classifiers (related name: *projects*)

Accessor to the related objects manager on the forward and reverse sides of a many-to-many relation.

In the example:

```
class Pizza(Model):
    toppings = ManyToManyField(Topping, related_name='pizzas')
```

`Pizza.toppings` and `Topping.pizzas` are `ManyToManyDescriptor` instances.

Most of the implementation is delegated to a dynamically defined manager

created

Type: CreationDateTimeField

Created

A wrapper for a deferred-loading field. When the value is read from this

description: Field

Type: CharField

Brief Description. A brief description of this project

A wrapper for a deferred-loading field. When the value is read from this

id

Type: AutoField

Primary key: ID

A wrapper for a deferred-loading field. When the value is read from this

property latest_version: Optional[*Version*]

Return the latest version (by version number) of our project documentation, if any.

Returns
The latest version of our project.

machine_name: Field

Type: *MachineNameField*

Machine Name. Must be unique. Set this to the slugified value of "project" in Sphinx's. conf.py

A wrapper for a deferred-loading field. When the value is read from this

modified

Type: ModificationDateTimeField

Modified

A wrapper for a deferred-loading field. When the value is read from this

objects = <django.db.models.Manager object>

permission_groups: ManyToManyField

Type: ManyToManyField to ProjectPermissionGroup

Permission groups (related name: projects)

Accessor to the related objects manager on the forward and reverse sides of a many-to-many relation.

In the example:

```
class Pizza(Model):
    toppings = ManyToManyField(Topping, related_name='pizzas')
```

Pizza.toppings and Topping.pizzas are ManyToManyDescriptor instances.

Most of the implementation is delegated to a dynamically defined manager

related_links

Type: Reverse ForeignKey from *ProjectRelatedLink*

All related links of this project (related name of *project*)

Accessor to the related objects manager on the reverse side of a many-to-one relation.

In the example:

```
class Child(Model):
    parent = ForeignKey(Parent, related_name='children')
```

Parent.children is a ReverseManyToOneDescriptor instance.

Most of the implementation is delegated to a dynamically defined manager

title: Field

Type: CharField

Project Name. The human name for this project

A wrapper for a deferred-loading field. When the value is read from this

versions

Type: Reverse ForeignKey from *Version*

All versions of this project (related name of *project*)

Accessor to the related objects manager on the reverse side of a many-to-one relation.

In the example:

```
class Child(Model):
    parent = ForeignKey(Parent, related_name='children')
```

Parent.children is a ReverseManyToOneDescriptor instance.

Most of the implementation is delegated to a dynamically defined manager

class sphinx_hosting.models.**ProjectRelatedLink**(**args*, ***kwargs*)

Database table: sphinxhostingcore_projectrelatedlink

A ProjectRelatedLink is a link to an external resource that is related to a *Project*.

Parameters

- **id** (*AutoField*) – Primary key: ID
- **created** (*CreationDateTimeField*) – Created
- **modified** (*ModificationDateTimeField*) – Modified
- **title** (*CharField*) – Link Title. The title for this link

  The page title
- **uri** (*URLField*) – Link URL. The URL for this link

Relationship fields:

Parameters

project (ForeignKey to *Project*) – Project. The project to which this link is related (related name: *related_links*)

exception DoesNotExist

exception MultipleObjectsReturned

get_delete_url() → str

get_next_by_created(**, field=<django_extensions.db.fields.CreationDateTimeField: created>, is_next=True, **kwargs*)

Finds next instance based on *created*. See get_next_by_FOO for more information.

get_next_by_modified(*, *field=<django_extensions.db.fields.ModificationDateTimeField: modified>*, *is_next=True*, ***kwargs*)

Finds next instance based on *modified*. See get_next_by_FOO for more information.

get_previous_by_created(*, *field=<django_extensions.db.fields.CreationDateTimeField: created>*, *is_next=False*, ***kwargs*)

Finds previous instance based on *created*. See get_previous_by_FOO for more information.

get_previous_by_modified(*, *field=<django_extensions.db.fields.ModificationDateTimeField: modified>*, *is_next=False*, ***kwargs*)

Finds previous instance based on *modified*. See get_previous_by_FOO for more information.

get_update_url() → str

created

Type: CreationDateTimeField

Created

A wrapper for a deferred-loading field. When the value is read from this

id

Type: AutoField

Primary key: ID

A wrapper for a deferred-loading field. When the value is read from this

modified

Type: ModificationDateTimeField

Modified

A wrapper for a deferred-loading field. When the value is read from this

objects = <django.db.models.Manager object>

project: ForeignKey

Type: ForeignKey to *Project*

Project. The project to which this link is related (related name: *related_links*)

Accessor to the related object on the forward side of a many-to-one or one-to-one (via ForwardOneToOneDescriptor subclass) relation.

In the example:

```
class Child(Model):
    parent = ForeignKey(Parent, related_name='children')
```

project_id

Internal field, use *project* instead.

title: Field

Type: CharField

Link Title. The title for this link

A wrapper for a deferred-loading field. When the value is read from this

uri: Field

Type: URLField

Link URL. The URL for this link

A wrapper for a deferred-loading field. When the value is read from this

class sphinx_hosting.models.**Version**(**args*, ***kwargs*)

Database table: sphinxhostingcore_version

A Version is a specific version of a *Project*. Versions own *SphinxPage* objects.

Parameters

- **id** (*AutoField*) – Primary key: ID
- **created** (*CreationDateTimeField*) – Created
- **modified** (*ModificationDateTimeField*) – Modified
- **version** (*CharField*) – Version. The version number for this release of the Project

  The :py:class:`Version that this tree examines
- **sphinx_version** (*CharField*) – Sphinx Version. The version of Sphinx used to create this documentation set
- **archived** (*BooleanField*) – Archived?. Whether this version should be excluded from search indexes

Relationship fields:

Parameters

- **project** (ForeignKey to *Project*) – Project. The Project to which this Version belongs (related name: *versions*)
- **head** (OneToOneField to *SphinxPage*) – Head. The top page of the documentation set for this version of our project (related name: +)

  The top page in the page hierarchy

Reverse relationships:

Parameters

- **pages** (Reverse ForeignKey from *SphinxPage*) – All pages of this version (related name of *version*)
- **images** (Reverse ForeignKey from *SphinxImage*) – All images of this version (related name of *version*)

exception DoesNotExist

exception MultipleObjectsReturned

get_absolute_url() → str

get_next_by_created(**, field=<django_extensions.db.fields.CreationDateTimeField: created>, is_next=True, **kwargs*)

Finds next instance based on *created*. See get_next_by_FOO for more information.

get_next_by_modified(**, field=<django_extensions.db.fields.ModificationDateTimeField: modified>, is_next=True, **kwargs*)

Finds next instance based on *modified*. See get_next_by_FOO for more information.

get_previous_by_created(*, *field=<django_extensions.db.fields.CreationDateTimeField: created>*, *is_next=False*, ***kwargs*)

Finds previous instance based on *created*. See get_previous_by_FOO for more information.

get_previous_by_modified(*, *field=<django_extensions.db.fields.ModificationDateTimeField: modified>*, *is_next=False*, ***kwargs*)

Finds previous instance based on *modified*. See get_previous_by_FOO for more information.

mark_searchable_pages() → None

Set the *SphinxPage.searchable* flag on the searchable pages in this version.

Searchable pages are ones that:

- Are not in *SphinxPage.SPECIAL_PAGES*
- Do not have a part of their relative path that starts with _.

Go through the pages in this version, and set *SphinxPage.searchable* to `True` for all those which meet the above requirements, `False` otherwise.

purge_cached_globaltoc() → None

Purge the cached output from our *globaltoc* property.

save(**args*, ***kwargs*)

Overriding django.db.models.Model.save here so that we can purge our cached global table of contents.

archived: Field

Type: BooleanField

Archived?. Whether this version should be excluded from search indexes

A wrapper for a deferred-loading field. When the value is read from this

created

Type: CreationDateTimeField

Created

A wrapper for a deferred-loading field. When the value is read from this

globaltoc

Build a struct that looks like this:

```
{
    items: [
        {'text': 'foo'},
        {'text': 'bar', 'url': '/foo', 'icon': None}
        {'text': 'bar', 'url': '/foo', 'icon': None, items: [{'text': 'blah' ...
↪} ...]}
        ...
    ]
}
```

suitable for constructing a *sphinx_hosting.wildewidgets.SphinxPageGlobalTableOfContentsMenu*

head: ForeignKey

Type: OneToOneField to *SphinxPage*

Head. The top page of the documentation set for this version of our project (related name: +)

Accessor to the related object on the forward side of a one-to-one relation.

In the example:

```
class Restaurant(Model):
    place = OneToOneField(Place, related_name='restaurant')
```

head_id

Internal field, use *head* instead.

id

Type: AutoField

Primary key: ID

A wrapper for a deferred-loading field. When the value is read from this

images

Type: Reverse ForeignKey from *SphinxImage*

All images of this version (related name of *version*)

Accessor to the related objects manager on the reverse side of a many-to-one relation.

In the example:

```
class Child(Model):
    parent = ForeignKey(Parent, related_name='children')
```

`Parent.children` is a `ReverseManyToOneDescriptor` instance.

Most of the implementation is delegated to a dynamically defined manager

property is_latest: bool

modified

Type: ModificationDateTimeField

Modified

A wrapper for a deferred-loading field. When the value is read from this

objects = <django.db.models.Manager object>

property page_tree: *SphinxPageTree*

Return the page hierarchy for the set of *SphinxPage* pages in this version.

The page hierarchy is build by traversing the pages in the set, starting with *head*.

Returns
The page hierarchy for this version.

pages

Type: Reverse ForeignKey from *SphinxPage*

All pages of this version (related name of *version*)

Accessor to the related objects manager on the reverse side of a many-to-one relation.

In the example:

```
class Child(Model):
    parent = ForeignKey(Parent, related_name='children')
```

Parent.children is a ReverseManyToOneDescriptor instance.

Most of the implementation is delegated to a dynamically defined manager

project: ForeignKey

Type: ForeignKey to *Project*

Project. The Project to which this Version belongs (related name: *versions*)

Accessor to the related object on the forward side of a many-to-one or one-to-one (via ForwardOneToOneDescriptor subclass) relation.

In the example:

```
class Child(Model):
    parent = ForeignKey(Parent, related_name='children')
```

project_id

Internal field, use *project* instead.

sphinx_version: Field

Type: CharField

Sphinx Version. The version of Sphinx used to create this documentation set

A wrapper for a deferred-loading field. When the value is read from this

version: Field

Type: CharField

Version. The version number for this release of the Project

A wrapper for a deferred-loading field. When the value is read from this

class sphinx_hosting.models.**SphinxPage**(**args*, ***kwargs*)

Database table: sphinxhostingcore_sphinxpage

A SphinxPage is a single page of a set of Sphinx documentation. SphinxPage objects are owned *Version* objects, which are in turn owned by *Project* objects.

Parameters

- **id** (*AutoField*) – Primary key: ID
- **created** (*CreationDateTimeField*) – Created
- **modified** (*ModificationDateTimeField*) – Modified
- **relative_path** (*CharField*) – Relative page path. The path to the page under our top slug
- **content** (*TextField*) – Content. The full JSON payload for the page
- **title** (*CharField*) – Title. Just the title for the page, extracted from the page JSON

  The page title
- **orig_body** (*TextField*) – Body (Original). The original body for the page, extracted from the page JSON. Some pages have no body. We save this here in case we need to reprocess the body at some later date.
- **body** (*TextField*) – Body. The body for the page, extracted from the page JSON, and modified to suit us. Some pages have no body. The body is actually stored as a Django template.

- **orig_local_toc** (*TextField*) – Local Table of Contents (original). The original table of contents for headings in this page.We save this here in case we need to reprocess the table of contents at some later date.
- **local_toc** (*TextField*) – Local Table of Contents. Table of Contents for headings in this page, modified to work in our templates
- **orig_global_toc** (*TextField*) – Global Table of Contents (original). The original global table of contents HTML attached to this page, if any. This will only be present if you had "sphinxcontrib-jsonglobaltoc" installed in your "extensions" in the Sphinx conf.py
- **searchable** (*BooleanField*) – Searchable. Should this page be included in the search index?

Relationship fields:

Parameters

- **version** (ForeignKey to *Version*) – Version. The Version to which this page belongs (related name: *pages*)

  The :py:class:`Version that this tree examines
- **parent** (ForeignKey to *SphinxPage*) – Parent. The parent page of this page (related name: *children*)

  The *SphinxPage* that is this page's parent
- **next_page** (ForeignKey to *SphinxPage*) – Next page. The next page in the documentation set (related name: *previous_page*)

Reverse relationships:

Parameters

- **children** (Reverse ForeignKey from *SphinxPage*) – All children of this sphinx page (related name of *parent*)

  The *TreeNode* objects that are this page's children
- **previous_page** (Reverse ForeignKey from *SphinxPage*) – All previous page of this sphinx page (related name of *next_page*)

exception DoesNotExist

exception MultipleObjectsReturned

get_absolute_url() → str

get_next_by_created(*, *field=<django_extensions.db.fields.CreationDateTimeField: created>*, *is_next=True*, ***kwargs*)

Finds next instance based on *created*. See get_next_by_FOO for more information.

get_next_by_modified(*, *field=<django_extensions.db.fields.ModificationDateTimeField: modified>*, *is_next=True*, ***kwargs*)

Finds next instance based on *modified*. See get_next_by_FOO for more information.

get_previous_by_created(*, *field=<django_extensions.db.fields.CreationDateTimeField: created>*, *is_next=False*, ***kwargs*)

Finds previous instance based on *created*. See get_previous_by_FOO for more information.

get_previous_by_modified(*, *field=<django_extensions.db.fields.ModificationDateTimeField: modified>*, *is_next=False*, ***kwargs*)

Finds previous instance based on *modified*. See get_previous_by_FOO for more information.

SPECIAL_PAGES: Dict[str, str] = {'_modules/index': 'Module code', 'genindex': 'General Index', 'np-modindex': 'Module Index', 'py-modindex': 'Module Index', 'search': 'Search'}

This is a mapping between filename and title that identifies the special pages that Sphinx produces on its own and gives them reasonable titles. These pages have no `title` key in their

body: Field

Type: TextField

Body. The body for the page, extracted from the page JSON, and modified to suit us. Some pages have no body. The body is actually stored as a Django template.

A wrapper for a deferred-loading field. When the value is read from this

children

Type: Reverse ForeignKey from *SphinxPage*

All children of this sphinx page (related name of *parent*)

Accessor to the related objects manager on the reverse side of a many-to-one relation.

In the example:

```
class Child(Model):
    parent = ForeignKey(Parent, related_name='children')
```

`Parent.children` is a `ReverseManyToOneDescriptor` instance.

Most of the implementation is delegated to a dynamically defined manager

content: Field

Type: TextField

Content. The full JSON payload for the page

A wrapper for a deferred-loading field. When the value is read from this

created

Type: CreationDateTimeField

Created

A wrapper for a deferred-loading field. When the value is read from this

id

Type: AutoField

Primary key: ID

A wrapper for a deferred-loading field. When the value is read from this

local_toc: Field

Type: TextField

Local Table of Contents. Table of Contents for headings in this page, modified to work in our templates

A wrapper for a deferred-loading field. When the value is read from this

modified

Type: ModificationDateTimeField

Modified

A wrapper for a deferred-loading field. When the value is read from this

next_page: ForeignKey

Type: ForeignKey to *SphinxPage*

Next page. The next page in the documentation set (related name: *previous_page*)

Accessor to the related object on the forward side of a many-to-one or one-to-one (via ForwardOneToOneDescriptor subclass) relation.

In the example:

```
class Child(Model):
    parent = ForeignKey(Parent, related_name='children')
```

next_page_id

Internal field, use *next_page* instead.

objects = <django.db.models.Manager object>

orig_body: Field

Type: TextField

Body (Original). The original body for the page, extracted from the page JSON. Some pages have no body. We save this here in case we need to reprocess the body at some later date.

A wrapper for a deferred-loading field. When the value is read from this

orig_global_toc: Field

Type: TextField

Global Table of Contents (original). The original global table of contents HTML attached to this page, if any. This will only be present if you had "sphinxcontrib-jsonglobaltoc" installed in your "extensions" in the Sphinx conf.py

A wrapper for a deferred-loading field. When the value is read from this

orig_local_toc: Field

Type: TextField

Local Table of Contents (original). The original table of contents for headings in this page.We save this here in case we need to reprocess the table of contents at some later date.

A wrapper for a deferred-loading field. When the value is read from this

parent: ForeignKey

Type: ForeignKey to *SphinxPage*

Parent. The parent page of this page (related name: *children*)

Accessor to the related object on the forward side of a many-to-one or one-to-one (via ForwardOneToOneDescriptor subclass) relation.

In the example:

```
class Child(Model):
    parent = ForeignKey(Parent, related_name='children')
```

parent_id

Internal field, use *parent* instead.

previous_page

Type: Reverse ForeignKey from *SphinxPage*

All previous page of this sphinx page (related name of *next_page*)

Accessor to the related objects manager on the reverse side of a many-to-one relation.

In the example:

```
class Child(Model):
    parent = ForeignKey(Parent, related_name='children')
```

`Parent.children` is a `ReverseManyToOneDescriptor` instance.

Most of the implementation is delegated to a dynamically defined manager

relative_path: Field

Type: CharField

Relative page path. The path to the page under our top slug

A wrapper for a deferred-loading field. When the value is read from this

searchable: Field

Type: BooleanField

Searchable. Should this page be included in the search index?

A wrapper for a deferred-loading field. When the value is read from this

title: Field

Type: CharField

Title. Just the title for the page, extracted from the page JSON

A wrapper for a deferred-loading field. When the value is read from this

version: ForeignKey

Type: ForeignKey to *Version*

Version. The Version to which this page belongs (related name: *pages*)

Accessor to the related object on the forward side of a many-to-one or one-to-one (via ForwardOneToOneDescriptor subclass) relation.

In the example:

```
class Child(Model):
    parent = ForeignKey(Parent, related_name='children')
```

version_id

Internal field, use *version* instead.

class sphinx_hosting.models.**SphinxImage**(**args*, ***kwargs*)

Database table: `sphinxhostingcore_sphinximage`

A `SphinxImage` is an image file referenced in a Sphinx document. When importing documenation packages, we extract all images from the package, upload them into Django storage and update the Sphinx HTML in *SphinxPage.body* to reference the URL for the uploaded image instead of its original url.

Parameters

- **id** (*AutoField*) – Primary key: ID
- **created** (*CreationDateTimeField*) – Created
- **modified** (*ModificationDateTimeField*) – Modified
- **orig_path** (*CharField*) – Original Path. The original path to this file in the Sphinx documentation package
- **file** (*FileField*) – An image file. The actual image file

Relationship fields:

Parameters

version (ForeignKey to *Version*) – Version. The version of our project documentation with which this image is associated (related name: *images*)

The :py:class:`Version that this tree examines

exception DoesNotExist

exception MultipleObjectsReturned

get_next_by_created(*, *field=<django_extensions.db.fields.CreationDateTimeField: created>*, *is_next=True*, ***kwargs*)

Finds next instance based on *created*. See get_next_by_FOO for more information.

get_next_by_modified(*, *field=<django_extensions.db.fields.ModificationDateTimeField: modified>*, *is_next=True*, ***kwargs*)

Finds next instance based on *modified*. See get_next_by_FOO for more information.

get_previous_by_created(*, *field=<django_extensions.db.fields.CreationDateTimeField: created>*, *is_next=False*, ***kwargs*)

Finds previous instance based on *created*. See get_previous_by_FOO for more information.

get_previous_by_modified(*, *field=<django_extensions.db.fields.ModificationDateTimeField: modified>*, *is_next=False*, ***kwargs*)

Finds previous instance based on *modified*. See get_previous_by_FOO for more information.

created

Type: CreationDateTimeField

Created

A wrapper for a deferred-loading field. When the value is read from this

file: Field

Type: FileField

An image file. The actual image file

The descriptor for the file attribute on the model instance. Return a FieldFile when accessed so you can write code like:

```
>>> from myapp.models import MyModel
>>> instance = MyModel.objects.get(pk=1)
>>> instance.file.size
```

Assign a file object on assignment so you can do:

```
>>> with open('/path/to/hello.world') as f:
```

id

Type: AutoField

Primary key: ID

A wrapper for a deferred-loading field. When the value is read from this

modified

Type: ModificationDateTimeField

Modified

A wrapper for a deferred-loading field. When the value is read from this

objects = <django.db.models.Manager object>

orig_path: Field

Type: CharField

Original Path. The original path to this file in the Sphinx documentation package

A wrapper for a deferred-loading field. When the value is read from this

version: ForeignKey

Type: ForeignKey to *Version*

Version. The version of our project documentation with which this image is associated (related name: *images*)

Accessor to the related object on the forward side of a many-to-one or one-to-one (via ForwardOneToOneDescriptor subclass) relation.

In the example:

```
class Child(Model):
    parent = ForeignKey(Parent, related_name='children')
```

version_id

Internal field, use *version* instead.

7.4 Utility functions

sphinx_hosting.models.**sphinx_image_upload_to**(*instance:* SphinxImage, *filename: str*) → str

Set the upload path within our MEDIA_ROOT for any images used by our Sphinx documentation to be:

```
{project machine_name}/{version}/images/{image basename}
```

Parameters

- **instance** – the *SphinxImage* object
- **filename** – the original path to the file

Returns

The properly formatted path to the file

7.5 Utility classes used by models

class sphinx_hosting.models.**SphinxPageTree**(*version:* Version)

A class that holds the page hierarchy for the set of *SphinxPage* pages in a *Version*.as a linked set of *TreeNode* objects.

The page heirarchy is built by starting at *Version.head* and following the page linkages by looking at *SphinxPage.next_page*, stopping the traversal when we find a *SphinxPage.next_page* that is None.

As we traverse, if a *SphinxPage.parent* is not None, find the *TreeNode* for that parent, and add the page to *TreeNode.children*.

For pages who have no SpinxPage.parent, assume they are top level children of the set, and make them children of *Version.head*.

Load it like so:

```
>>> project = Project.objects.get(machine_name='my-project')
>>> version = project.versions.get(version='1.0.0')
>>> tree = SphinxPageTree(version)
```

You can then traverse the built hierarchy by starting at *SphinxPageTree.head*, looking at its children, then looking at their children, etc..

```
>>>
```

__init__(*version:* Version)

traverse() → List[*SphinxPage*]

Return a list of the pages represented in this tree.

head: *TreeNode*

version: *Version*

class:`Version that this tree examines

Type
The

class sphinx_hosting.models.**SphinxPageTreeProcessor**

build(*items: List[Dict[str, Any]]*, *node:* TreeNode) → None

Build a wildewdigets.MenuItem compatible dict representing node, and append it to items.

if node has children, recurse into those children, building out our submenus.

Parameters

- **items** – the current list of MenuItem compatible dicts for the current level of the menu
- **node** – the current node in our page tree

build_item(*node:* TreeNode) → Dict[str, Any]

Build a wildewdigets.MenuItem compatible dict representing node.

Parameters
node – the current node in our page tree

Returns
A dict suitable for loading into a wildewidgets.MenuItem.

run(*version:* Version) → List[Dict[str, Any]]

Parse the *Version.page_tree* and return a struct that works with *sphinx_hosting.wildewidgets.SphinxPageGlobalTableOfContentsMenu.parse_obj*

The returned struct should look something like this:

```
[
    {'text': 'foo'},
    {'text': 'bar', 'url': '/foo', 'icon': None}
    {'text': 'bar', 'url': '/foo', 'icon': None, items: [{'text': 'blah' ...} ..
↪.]}
    ...
]
```

Parameters

version – the version whose global table of contents we are parsing

Returns

A list of dicts representing the global menu structure

class sphinx_hosting.models.**SphinxGlobalTOCHTMLProcessor**(*max_level: int = 2*)

Usage: `SphinxGlobalTOCHTMLProcessor().run(version, globaltoc_html)`

This importer is used to parse the `globaltoc` key in JSON output of Sphinx pages built with the sphinxcontrib-jsonglobaltoc extension.

Sphinx uses your `.. toctree:` declarations in your `.rst` files to build site navigation for your document tree, and `sphinxcontrib-jsonglobaltoc` saves the Sphinx HTML produced by those `..toctree` as the `globaltoc` key in the *.fjson* output.

Note: Sphinx `.. toctree:` are ad-hoc – they're up to how the author wants to organize their content, and may not reflect how files are filled out in the filesystem.

__init__(*max_level: int = 2*) → None

parse_globaltoc(*html: HtmlElement*) → List[Dict[str, Any]]

Parse our global table of contents HTML blob and return a list of `sphinx_hosting.wildewidgets.MenuItem` objects.

Add a first node that points to the root doc, also. The root doc can't add itself to its `toctree` blocks, so we need to do it ourselves.

How our mapping works:

- Multiple top level <ul> tags separated by tags will be merged into a single list.
- CONTENTS becomes `{'text': 'CONTENTS'}`
- Any `href` will be converted to its full `django-sphinx-hosting` path

Parameters

- **version** – the version whose global table of contents we are parsing
- **html** – the lxml parsed HTML of the global table of contents from Sphinx

parse_ul(*html: HtmlElement*, *level: int = 1*) → List[Dict[str, Any]]

Process `html`, an `lxml` parsed set of elements representing the contents of a <ul> from a Sphinx table of contents and return a list of `sphinx_hosting.wildewidgets.MenuItem` objects.

Any `href` in links found will be converted to its full `django-sphinx-hosting` path.

If we find another <ul> inside `html`, process it by passing its contents to *parse_ul* again, incrementing the menu level.

If `level` is greater than *max_level*, return an empty list, stopping our recursion.

Parameters
html – the list of elements that are the contents of the parent

Keyword Arguments
level – the current menu level

Returns
The <ul> contents as a list of dicts

run(*version: Version*, *verbose: bool = False*) → List[Dict[str, Any]]

Parse the global table of contents found as `version.head.orig_global_toc` into a data struct suitable for use with *sphinx_hosting.wildewidgets.SphinxPageGlobalTableOfContentsMenu.parse_obj* and return it.

How our mapping works:

- Multiple top level <ul> tags separated by tags will be merged into a single list.
- CONTENTS becomes {'text': 'CONTENTS'}`
- Any `href` for links found will be converted to its full `django-sphinx-hosting` path

The returned struct should look something like this:

```
[
    {'text': 'foo'},
    {'text': 'bar', 'url': '/project/version/foo', 'icon': None}
    {'text': 'bar', 'url': '/project/version/bar', 'icon': None, items: [{'text
↪': 'blah' ...} ...]}
    ...
]
```

Parameters
version – the version whose global table of contents we are parsing

Keyword Arguments
verbose – if `True`, pretty print the HTML of the globaltoc

Returns
A list of dicts representing the global menu structure

max_level: int

class sphinx_hosting.models.**TreeNode**(*title: str*, *page: ~typing.Optional[~sphinx_hosting.models.SphinxPage] = None*, *prev: ~typing.Optional[~sphinx_hosting.models.SphinxPage] = None*, *next: ~typing.Optional[~sphinx_hosting.models.SphinxPage] = None*, *parent: ~typing.Optional[~sphinx_hosting.models.SphinxPage] = None*, *children: ~typing.List[~sphinx_hosting.models.TreeNode] = <factory>*)

This is a `dataclass` that we use with *SphinxPageTree* to build out the global navigation structure for a set of documentation for a *Version*.

__init__(*title: str*, *page: ~typing.Optional[~sphinx_hosting.models.SphinxPage] = None*, *prev: ~typing.Optional[~sphinx_hosting.models.SphinxPage] = None*, *next: ~typing.Optional[~sphinx_hosting.models.SphinxPage] = None*, *parent: ~typing.Optional[~sphinx_hosting.models.SphinxPage] = None*, *children: ~typing.List[~sphinx_hosting.models.TreeNode] = <factory>*) → None

classmethod from_page(*page:* SphinxPage) → *TreeNode*

Build a *TreeNode* from `page`.

Note: This does not populate *children*; *SphinxPageTree* will populate it as appropriate as it ingests pages.

Parameters
page – the *SphinxPage* from which to build a node

Returns
A configured node.

children: List[*TreeNode*]

next: Optional[*SphinxPage*] = None

page: Optional[*SphinxPage*] = None

parent: Optional[*SphinxPage*] = None

prev: Optional[*SphinxPage*] = None

title: str

class sphinx_hosting.models.**ClassifierNode**(*title: str*, *classifier: Union[ForwardRef('Classifier'), NoneType] = None*, *items: Dict[str, ForwardRef('ClassifierNode')] = <factory>*)

__init__(*title: str*, *classifier: ~typing.Optional[~sphinx_hosting.models.Classifier] = None*, *items: ~typing.Dict[str, ~sphinx_hosting.models.ClassifierNode] = <factory>*) → None

CHAPTER

EIGHT

FORMS

class sphinx_hosting.forms.**GlobalSearchForm**(**args*, ***kwargs*)

This is the search form at the top of the sidebar, underneath the logo. It is a subclass of `haystack.forms.SearchForm`, and does a search of our haystack backend.

Form fields:

- q: Search (CharField)

__init__(**args*, ***kwargs*)

property media

Return all media required to render the widgets on this form.

class sphinx_hosting.forms.**ProjectCreateForm**(**args*, ***kwargs*)

This is the form we use to create a new *sphinx_hosting.models.Project*. The difference between this and *sphinx_hosting.forms.ProjectUpdateForm* is that the user can set *sphinx_hosting.models.Project.machine_name* here, but can't in *sphinx_hosting.forms.ProjectUpdateForm*. machine_name should not change after the project is created.

Form fields:

- title: Project Name (CharField)
- description: Brief Description (CharField)
- machine_name: Machine Name (*MachineNameField*)

__init__(**args*, ***kwargs*)

property media

Return all media required to render the widgets on this form.

class sphinx_hosting.forms.**ProjectUpdateForm**(**args*, ***kwargs*)

This is the form we use to update an existing *sphinx_hosting.models.Project*. The difference between this and *ProjectCreateForm* is that the user cannot change *sphinx_hosting.models.Project.machine_name* here, but can in *ProjectCreateForm*. machine_name should not change after the project is created.

Form fields:

- title: Project Name (CharField)
- description: Brief Description (CharField)

__init__(**args*, ***kwargs*)

property media

Return all media required to render the widgets on this form.

class sphinx_hosting.forms.**ProjectReadonlyUpdateForm**(**args*, ***kwargs*)

This is the form we use to on the `sphinx_hosting.views.ProjectDetailView` to show the viewer the project title and description. The difference between this and *ProjectUpdateForm* is that all the fields are readonly, and there are no submit buttons. We're doing it this way instead of just rendering a non-form widget so that we can ensure that the page looks the same.

Form fields:

- `title`: Project Name (`CharField`)
- `description`: Brief Description (`CharField`)

__init__(**args*, ***kwargs*)

property media

Return all media required to render the widgets on this form.

class sphinx_hosting.forms.**VersionUploadForm**(**args*, *project: Optional[*Project*] = None*, ***kwargs*)

This is the form on `sphinx_hosting.views.ProjectDetailView` that allows the user to upload a new documentation set.

Keyword Arguments
project – the project to which this documentation set should be associated

Form fields:

- `file`: File (`FileField`)

__init__(**args*, *project: Optional[*Project*] = None*, ***kwargs*)

property media

Return all media required to render the widgets on this form.

8.1 Fields

class sphinx_hosting.form_fields.**MachineNameField**(***, *allow_unicode=False*, ***kwargs*)

This is a form field for our *sphinx_hosting.fields.MachineNameField* that applies the appropriate validators.

The difference this field and `django.forms.SlugField` is that this field will allow "-" characters in the value

CHAPTER

NINE

IMPORTERS

class sphinx_hosting.importers.**PageTreeNode**(*page:* SphinxPage, *parent_title: Optional[str] = None*, *next_title: Optional[str] = None*)

A data structure to temporarily hold relationships between *sphinx_hosting.models.SphinxPage* objects while importing pages.

In the page JSON we get from Sphinx, we only know the titles of related pages, so we store them here along with the *sphinx_hosting.models.SphinxPage* we created from our JSON, and then do another pass through these *PageTreeNode* objects to link our pages together.

This is used in *SphinxPackageImporter.link_pages*.

__init__(*page:* SphinxPage, *parent_title: Optional[str] = None*, *next_title: Optional[str] = None*) → None

next_title: Optional[str] = None

page: *SphinxPage*

parent_title: Optional[str] = None

class sphinx_hosting.importers.**SphinxPackageImporter**

Usage: `SphinxPackageImporter().run(sphinx_tarfilename)`

Import a tarfile of a built set of Sphinx documentation into the database.

Important: Before importing, there must be a *sphinx_hosting.models.Project* in the database whose `machine_name` matches the `project` in Sphinx's `conf.py` config file for the docs to be imported.

The documentation package should have been built via the `json` output from `sphinx-build`, so either:

```
make json
```

or:

```
sphinx-build -n -b json build/json
```

The tarfile should be built like so:

```
cd build
tar zcf mydocs.tar.gz json
```

ensuring that the package contents are enclosed in a folder.

When run, *SphinxPackageImporter* will look inside the tarfile at the `globalcontext.json` file to determine which project and version we should associate these pages with.

- The `project` key in will be used to look up the *sphinx_hosting.models.Project* to associate these Sphinx pages with, using `project` as *sphinx_hosting.models.Project.machine_name*
- The `version` key will be used to create a new *sphinx_hosting.models.Version* object tied to that project

Once the *sphinx_hosting.models.Version* has been created, the pages in the tarfile will be created as *sphinx_hosting.models.SphinxPage* objects, and the images will be created as *sphinx_hosting.models.SphinxImage* objects.

__init__() → None

get_version(*package: TarFile*, *force: bool = False*) → *Version*

Look in `package` for a member named `globalcontext.json`, and load that file as JSON.

Extract these things from that JSON:

- the version string from the `release` key.
- the `machine_name` of the `Project` for this documentation tarfile as the slugified version of the `project` key

Return a new `Version` instance on the project.

Parameters
package – the opened Sphinx documentation tarfile

Keyword Arguments
force – if `True`, re-use an existing version, purging any docs and images associated with it first

Raises

- ***Project.DoesNotExist*** – no `Project` exists whose `machine_name` matches the slugified `project` setting in the Sphinx package's `conf.py`
- **VersionAlreadyExists** – a `Version` with version string `release` from the Sphinx `conf.py` already exists for our project, and `force` was not `True`

import_images(*package: TarFile*, *version:* Version) → None

Import all images in our Sphinx documentation into the database before importing any pages, then return a lookup dict for doing `<img src="image_path">` replacements in the page bodies.

Parameters

- **package** – the opened Sphinx documentation tarfile
- **version** – the `Version` which which to associate our images

import_pages(*package: TarFile*, *version:* Version) → None

Import a all pages from `package` into the database as *sphinx_hosting.models.SphinxPage* objects, associating them with `Version` `version`.

Parameters

- **package** – the tarfile of the sphinx docs
- **version** – the `Version` object to associated data

Returns
The page linkage tree for consumption by *link_pages*.

link_pages() → None

Given `page_tree`, a list of page linkages (parent, next, prev), link all the *sphinx_hosting.models.SphinxPage* objects in that list to their next page and their parent page.

Parameters
tree – the page linkage tree

load_config(*package: TarFile*) → None

Load the `globalcontext.json` file for later reference.

Parameters
package – the opened Sphinx documentation tarfile

run(*filename: Optional[str] = None*, *file_obj: Optional[IO] = None*, *force: bool = False*) → *Version*

Load the pages in the tarfile identified by `filename` into the database as `Version` `version` of `Project` `project`. See the class docs for *SphinxPackageImporter* for more background on how to prepare the package named by `filename`.

Parameters
filename – the filename of the gzipped tar archive of the Sphinx pages

Keyword Arguments
force – if `True`, overwrite the docs for an existing version

Raises

- ***Project.DoesNotExist*** – no `Project` exists whose `machine_name` matches the slugified `project` setting in the Sphinx package's `conf.py`
- **VersionAlreadyExists** – a `Version` with version string `release` from the Sphinx package's `conf.py` already exists for our project, and `force` was not `True`

config: `Dict[str, Any]`

image_map: `Dict[str, SphinxImage]`

page_tree: `Dict[str, PageTreeNode]`

CHAPTER

TEN

WIDGETS

This part of the documentation covers all the reusable django-wildewidgets widgets provided by `django-sphinx-hosting`.

10.1 Navigation

These widgets are used on every page.

class sphinx_hosting.wildewidgets.**SphinxHostingSidebar**(**args*, *wide: Optional[bool] = None*, ***kwargs*)

This is the vertical menu area on the left of the page. It houses our search form, *sphinx_hosting.wildewidgets.search.GlobalSearchFormWidget*, our main menu *SphinxHostingMainMenu*, possibly our utility menu `SphinxHostingLookupsMenu` and finally the global navigation for a *sphinx_hosting.models.Version* when we're reading our documentation.

branding: Block = <wildewidgets.widgets.base.Block object>

contents: Iterable[Block] = [<sphinx_hosting.wildewidgets.navigation.SphinxHostingMainMenu object>]

hide_below_viewport: str = 'xl'

The viewport size at which our menu container collapses to be hidden,

wide: bool = True

class sphinx_hosting.wildewidgets.**SphinxHostingMainMenu**(**items*, ***kwargs*)

This is the primary menu that appears in *SphinxHostingSidebar* directly underneath the "Search" form, *sphinx_hosting.wildewidgets.search.GlobalSearchFormWidget*.

It gives access to all the views that normal, non privileged users should be allowed to use.

__init__(**items*, ***kwargs*) → None

activate(*text: str*) → bool

Normally, how activate works is that it looks through our menu items, finds the one whose `text` or `url` matches `text`.

In our case, we're building the menu items in *build_menu*, which occurs after *activate* would be called, so we have to save the `text` here, and use it later in *build_menu*.

Parameters

text – the text to search for among our *items*

Returns
We always return `True` here, since we won't know if we match until later.

build_menu(*items: List[MenuItem]*) → None

Programatically build our menu here. We have this code because the presence of some items in this menu depends on the user's permissions.

We have to do it here because if we do it in `__init__`, that's too early in the Django boostrap and the global Django `urlpatterns` have not finished building, and even `reverse_lazy` fails.

items: `Iterable[MenuItem] = [MenuItem(text='Projects', icon='bookshelf', url='/', items=[], active=False)]`

title: `str = 'Main'`

class sphinx_hosting.wildewidgets.**SphinxHostingBreadcrumbs**(**args*, *title_class: Optional[str] = None*, ***kwargs*)

items: `List[BreadcrumbItem] = [BreadcrumbItem(title='Sphinx Hosting', url='/')]`

The list of `BreadcrumbItem` objects from which we will

10.2 Projects

These widgets are used on the project listing and details pages.

class sphinx_hosting.wildewidgets.**ClassifierFilterForm**(*table_name: str*, *column_number: int*, ***kwargs*)

This is the tree-like classifier filter form that appears to the right of the *sphinx_hosting.wildewidgets.project.ProjectTable*. It is embedded in *ClassifierFilterBlock*.

It allows the user to select a set of classifiers by which to filter the projects listing table.

__init__(*table_name: str*, *column_number: int*, ***kwargs*)

add_subtree(*contents: UnorderedList*, *nodes: Dict[str,* ClassifierNode*]*) → None

Add a subtree of classifier checkboxes.

Parameters

- **contents** – the <ul> block to which to add our list of classifier checkboxes
- **nodes** (*_type_*) – _description_

get_checkbox(*node:* ClassifierNode) → HorizontalLayoutBlock

Build and return the `wildewidgets.CheckboxInputBlock` for the classifier `node`.

Parameters
node – the classifier data

Returns
A configured `wildewidgets.CheckboxInputBlock`

block: `str = 'classifiers__filter__form'`

block is the official wildewidgets name of the block; it can't be changed

tag: `str = 'form'`

class sphinx_hosting.wildewidgets.**ClassifierFilterBlock**(*table_name: str*, *column_number: int*, ***kwargs*)

A `wildewidgets.CardWidget` that contains the *ClassifierFilterForm*. This the right of the *sphinx_hosting.wildewidgets.project.ProjectTable*. This widget is embedded in `sphinx_hosting.wildewdigets.project.ProjectTableWidget`

Parameters

- **table_name** – the name of the dataTables table to control
- **column_number** – the number of the column in the dataTable that contains classifier names

__init__(*table_name: str*, *column_number: int*, ***kwargs*)

name: str = 'classifiers__filter'

The CSS class that will be added to this element to as an identifier for

class sphinx_hosting.wildewidgets.**ProjectCreateModalWidget**(**args*, ***kwargs*)

This is a modal dialog that holds the *sphinx_hosting.forms.ProjectCreateForm*.

__init__(**args*, ***kwargs*)

class sphinx_hosting.wildewidgets.**ProjectDetailWidget**(**blocks*, *form: Optional[Form] = None*, ***kwargs*)

This widget renders an update form for a *sphinx_hosting.models.Project*.

Use directly it like so:

```
>>> project = Project.objects.get(pk=1)
>>> form = ProjectUpdateForm(instance=project)
>>> widget = ProjectDetailWidget(form=form)
```

Or you can simply add the form to your view context and *ProjectDetailWidget* will pick it up automatically.

css_class: str = ' p-4'

modifier: str = 'general'

If specified, also add a class named `{name}--{modifier}` to the CSS

name: str = 'project-detail__section'

The CSS class that will be added to this element to as an identifier for

class sphinx_hosting.wildewidgets.**ProjectTableWidget**(*user: AbstractUser*, ***kwargs*)

This is a `wildewidgets.CardWidget` that gives our *ProjectTable* dataTable a nice header with a total book count and an "Add Project" button that opens a modal dialog.

__init__(*user: AbstractUser*, ***kwargs*)

class sphinx_hosting.wildewidgets.**ProjectTableWidget**(*user: AbstractUser*, ***kwargs*)

This is a `wildewidgets.CardWidget` that gives our *ProjectTable* dataTable a nice header with a total book count and an "Add Project" button that opens a modal dialog.

__init__(*user: AbstractUser*, ***kwargs*)

class sphinx_hosting.wildewidgets.**ProjectVersionsTableWidget**(*project_id: int*, ***kwargs*)

This is a `wildewidgets.CardWidget` that gives our *ProjectVersionTable* dataTable a nice header with a total version count.

__init__(*project_id: int*, ***kwargs*)

class sphinx_hosting.wildewidgets.**ProjectTable**(**args*, *actions: Optional[List[RowActionButton]] = None*, *button_size: Optional[str] = None*, *justify: Optional[str] = None*, ***kwargs*)

This widget displays a dataTable of our *sphinx_hosting.models.Project* instances.

It's used as a the main widget in by *ProjectTableWidget*.

model

alias of *Project*

filter_classifiers_column(*qs: QuerySet*, *column: str*, *value: str*) → QuerySet

Filter our results by the `value`, a comma separated list of *sphinx_hosting.models.Classifier* names.

Parameters

- **qs** – the current `QuerySet`
- **column** – the name of the column to filter on
- **value** – a comma-separated list of classifier names

Returns

A `QuerySet` filtered for rows that contain the selected classifiers.

render_classifiers_column(*row:* Project, *column: str*) → str

Render our `classifiers` column.

Parameters

- **row** – the `Project` we are rendering
- **column** – the name of the column to render

Returns

A
 separated list of classifier names

render_latest_version_column(*row:* Project, *column: str*) → str

Render our `latest_version` column. This is the version string of the *sphinx_hosting.models.Version* that has the most recent *sphinx_hosting.models.Version.modified* timestamp.

If there are not yet any *sphinx_hosting.models.Version* instances for this project, return empty string.

Parameters

- **row** – the `Project` we are rendering
- **column** – the name of the column to render

Returns

The version string of the most recently published version, or empty string.

render_latest_version_date_column(*row:* Project, *column: str*) → str

Render our `latest_version_date` column. This is the last modified date of the *sphinx_hosting.models.Version* that has the most recent *sphinx_hosting.models.Version.modified* timestamp.

If there are not yet any *sphinx_hosting.models.Version* instances for this project, return empty string.

Parameters

- **row** – the `Project` we are rendering
- **column** – the name of the column to render

Returns
The of the most recently published version, or empty string.

alignment: Dict[str, str] = {'classifiers': 'left', 'description': 'left', 'latest_version': 'left', 'latest_version_date': 'left', 'machine_name': 'left', 'title': 'left'}

fields: List[str] = ['title', 'machine_name', 'classifiers', 'latest_version', 'description', 'latest_version_date']

A list of fields that we will list as columns. These are either fields on our *model*, or defined as `render_FIELD_NAME_column` methods

hidden: List[str] = ['classifiers', 'machine_name', 'latest_version_date']

page_length: int = 25

striped: bool = True

unsearchable: List[str] = ['lastest_version', 'latest_version_date']

A list of names of columns that will will not be searched when doing a

verbose_names: Dict[str, str] = {'latest_version': 'Latest Version', 'latest_version_date': 'Import Date', 'machine_name': 'Machine Name', 'title': 'Project Name'}

A dict of column name to column label. We use it to override the

class sphinx_hosting.wildewidgets.**ProjectVersionTable**(**args*, ***kwargs*)

This widget displays a dataTable of our *sphinx_hosting.models.Version* instances for a particular *sphinx_hosting.models.Project*.

It's used as a the main widget in by `ProjectVersionTableWidget`.

model

alias of *Version*

__init__(**args*, ***kwargs*) → None

One of our `kwargs` must be `project_id`, the `pk` of the *sphinx_hosting.models.Project* for which we want to list *sphinx_hosting.models.Version* objects.

This will get added to the `extra_data` dict in the `kwargs` key, from which we reference it.

get_initial_queryset() → QuerySet

Filter our *sphinx_hosting.models.Version* objects by *project_id*.

Returns
A filtered `QuerySet` on *sphinx_hosting.models.Version*

render_num_images_column(*row:* Version, *column: str*) → str

Render our `num_images` column. This is the number of *sphinx_hosting.models.SphinxImage* objects imported for this version.

Parameters

- **row** – the `Version` we are rendering
- **column** – the name of the column to render

Returns
The number of images for this version.

render_num_pages_column(*row:* Version, *column: str*) → str

Render our num_pages column. This is the number of *sphinx_hosting.models.SphinxPage* objects imported for this version.

Parameters

- **row** – the Version we are rendering
- **column** – the name of the column to render

Returns
The number of pages for this version.

actions: bool = True

Per row action buttons. If not False, this will simply add a rightmost column named Actions with a button named default_action_button_label which when clicked will take the

alignment: Dict[str, str] = {'created': 'left', 'modified': 'left', 'num_images': 'right', 'num_pages': 'right', 'version': 'left'}

fields: List[str] = ['version', 'num_pages', 'num_images', 'created', 'modified']

A list of fields that we will list as columns. These are either fields on our *model*, or defined as render_FIELD_NAME_column methods

order_columns: List[str] = ['version']

page_length: int = 25

project_id: Optional[int]

sort_ascending: bool = False

striped: bool = True

unsearchable: List[str] = ['num_pages', 'num_images']

A list of names of columns that will will not be searched when doing a

verbose_names: Dict[str, str] = {'num_images': '# Images', 'num_pages': '# Pages', 'title': 'Version'}

A dict of column name to column label. We use it to override the

10.3 ProjectRelatedLinks

class sphinx_hosting.wildewidgets.**ProjectRelatedLinkCreateModalWidget**(*project:* Project, **args*, ***kwargs*)

This is a modal dialog that holds the sphinx_hosting.forms.ProjectRelatedLinkForm for creating links.

__init__(*project:* Project, **args*, ***kwargs*)

class sphinx_hosting.wildewidgets.**ProjectRelatedLinkUpdateModalWidget**(*link:* ProjectRelatedLink, **args*, ***kwargs*)

This is a modal dialog that holds the sphinx_hosting.forms.ProjectRelatedLinkForm for updating links.

One of these is created for each *sphinx_hosting.models.ProjectRelatedLink* object, and lives in the sphinx_hosting.widgets.ProjectRelatedLinksWidget.

__init__(*link:* ProjectRelatedLink, **args*, ***kwargs*)

class sphinx_hosting.wildewidgets.**ProjectRelatedLinksListWidget**(**args*, *remove_url=None*, ***kwargs*)

This widget renders a list of *sphinx_hosting.models.ProjectRelatedLink* objects as a list of `wildewidgets.Link` objects.

This is used on the project detail page, and is the read-only version of *ProjectRelatedLinksWidget*.

item_label: `str` = **'Related Link'**

The label to use for each instance. This is used in the confirmation

class sphinx_hosting.wildewidgets.**ProjectRelatedLinksWidget**(*project:* Project, ***kwargs*)

This is a `wildewidgets.CardWidget` that allows us to manage the *sphinx_hosting.models.ProjectRelatedLink* objects for this *sphinx_hosting.models.Project*.

It renders a list of *ProjectRelatedLinkListItemWidget* objects and a "Add Related Link" button that opens a modal dialog.

This is used on the project update page, and is the editable version of *ProjectRelatedLinksListWidget*.

__init__(*project:* Project, ***kwargs*)

class sphinx_hosting.wildewidgets.**ProjectRelatedLinkListItemWidget**(*object:* ProjectRelatedLink)

This is used by *ProjectRelatedLinksWidget* to render a single *sphinx_hosting.models.ProjectRelatedLink* object in its list.

__init__(*object:* ProjectRelatedLink)

css_class: `str` = **'mb-2'**

10.4 Search

These widgets are used on the search results page.

class sphinx_hosting.wildewidgets.**GlobalSearchFormWidget**(**args*, ***kwargs*)

This widget encapsulates the *sphinx_hosting.forms.GlobalSearchForm*.

__init__(**args*, ***kwargs*)

css_class: `str` = **'mb-3'**

name: `str` = **'global-search'**

The CSS class that will be added to this element to as an identifier for

class sphinx_hosting.wildewidgets.**PagedSearchLayout**(*results: SearchQuerySet*, *query: Optional[str] = None*, *facets: Optional[Dict[str, List[str]]] = None*, ***kwargs*)

This is the page layout for the entire search results page.

Parameters

query – the text entered into the search form that got us to this results page

Keyword Arguments

- **query** – the text entered into the search form that got us to this results page
- **facets** – the active facet filters. This will be a dict where the key is the facet name, and the value is a list of facet values to filter by.

__init__(*results: SearchQuerySet*, *query: Optional[str] = None*, *facets: Optional[Dict[str, List[str]]] = None*, ***kwargs*)

modifier: `str = 'paged'`

If specified, also add a class named `{name}--{modifier}` to the CSS

name: `str = 'search-layout'`

The CSS class that will be added to this element to as an identifier for

class sphinx_hosting.wildewidgets.**SearchResultsPageHeader**(*query: Optional[str]*, *facets: Optional[Dict[str, List[str]]] = None*, ***kwargs*)

The header for the entire search results page. This shows the search string that got us here, and any active facet filters, allowing the user to remove any active filter by clicking the "X" next to the filter name.

Parameters

query – the text entered into the search form that got us to this results page

Keyword Arguments

facets – the active facet filters. This will be a dict where the key is the facet name, and the value is a list of facet values to filter by.

__init__(*query: Optional[str]*, *facets: Optional[Dict[str, List[str]]] = None*, ***kwargs*)

block: `str = 'search-results__header'`

block is the official wildewidgets name of the block; it can't be changed

css_class: `str = 'mb-5'`

class sphinx_hosting.wildewidgets.**FacetBlock**(*results: SearchQuerySet*, *query: Optional[str]*, ***kwargs*)

This is a base class for blocks that appear to the right of the search results listing on the search results page that allows you to refine your results by a particular facet that is present in the result set.

Subclass this to create facet filtering blocks for specific facets. Any facet you want to filter by must be defined on your search index by adding `faceted=True` to the field definition.

Parameters

- **results** – the Haystack search queryset containing our search results
- **query** – the text entered into the search form that got us to this results page

__init__(*results: SearchQuerySet*, *query: Optional[str]*, ***kwargs*)

facet: `str`

model: `Type[Model]`

model_field: `str`

class sphinx_hosting.wildewidgets.**SearchResultsClassifiersFacet**(*results: SearchQuerySet*, *query: Optional[str]*, ***kwargs*)

A *FacetBlock* that allows the user to filter search results by classifier.

model

alias of *Classifier*

facet: `str = 'classifiers'`

model_field: `str = 'name'`

title: *str* = 'Classifiers'

class sphinx_hosting.wildewidgets.**SearchResultsProjectFacet**(*results: SearchQuerySet*, *query: Optional[str]*, ***kwargs*)

A *FacetBlock* that allows the user to filter search results by project.

model

alias of *Project*

facet: *str* = 'project_id'

model_field: *str* = 'pk'

title: *str* = 'Projects'

class sphinx_hosting.wildewidgets.**PagedSearchResultsBlock**(*results: SearchQuerySet*, *query: Optional[str]*, ***kwargs*)

This is a paged listing of *SearchResultBlock* entries describing our search results.

model

alias of *SphinxPage*

model_widget

alias of *SearchResultBlock*

__init__(*results: SearchQuerySet*, *query: Optional[str]*, ***kwargs*)

class sphinx_hosting.wildewidgets.**SearchResultsHeader**(*results: SearchQuerySet*, ***kwargs*)

The header for the search results listing (not the page header – that is *SearchResultsPageHeader*).

Parameters
results – the Haystack search queryset containing our search results

__init__(*results: SearchQuerySet*, ***kwargs*)

justify: *str* = 'between'

`start`, `center`, `end`, `between`, `around`, `evenly`. See Bootstrap: Flex, justify content. If not supplied here and *justify* is `None`, do whatever horizontal aligment Bootstrap does by default.

name: *str* = 'search-results__title'

The CSS class that will be added to this element to as an identifier for

class sphinx_hosting.wildewidgets.**SearchResultBlock**(*object: Optional[SearchResult] = None*, ***kwargs*)

This is the block we use for rendering a particular search result on the search results page.

Keyword Arguments
object – the `haystack.models.SearchResult` object to render

class Header(*result: SearchResult*, ***kwargs*)

__init__(*result: SearchResult*, ***kwargs*)

name: *str* = 'search-result__header'

The CSS class that will be added to this element to as an identifier for

__init__(*object: Optional[SearchResult] = None*, ***kwargs*)

block: *str* = 'search-result'

block is the official wildewidgets name of the block; it can't be changed

10.5 Versions

class sphinx_hosting.wildewidgets.**VersionInfoWidget**(*version:* Version, ***kwargs*)

This widget gives a `wildewidget.Datagrid` type overview of information about this version:

- A link to the project that owns this *sphinx_hosting.models.Version*
- Create and last modified timestamps
- What version of Sphinx was used to generate the pages

Parameters
version – the `Version` object we're describing

__init__(*version:* Version, ***kwargs*)

class sphinx_hosting.wildewidgets.**VersionSphinxPageTableWidget**(*version_id: int*, ***kwargs*)

This is a `wildewidgets.CardWidget` that gives our *VersionSphinxPageTable* dataTable a nice header with a total page count.

__init__(*version_id: int*, ***kwargs*)

class sphinx_hosting.wildewidgets.**VersionUploadBlock**(**blocks*, *form=None*, ***kwargs*)

This block holds the upload form for uploading documentation tarballs. Once uploaded, the tarball will be run through *sphinx_hosting.importers.SphinxPackageImporter* to actually import it into the database.

__init__(**blocks*, *form=None*, ***kwargs*)

css_class: `str` = `'my-3 border'`

class sphinx_hosting.wildewidgets.**VersionSphinxImageTableWidget**(*version_id: int*, ***kwargs*)

This is a `wildewidgets.CardWidget` that gives our *VersionSphinxImageTable* dataTable a nice header with a total image count.

__init__(*version_id: int*, ***kwargs*)

class sphinx_hosting.wildewidgets.**VersionSphinxPageTable**(**args*, ***kwargs*)

This widget displays a dataTable of our *sphinx_hosting.models.SphinxPage* instances for a particular *sphinx_hosting.models.Version*.

It's used as a the main widget in by *VersionSphinxPageTableWidget*.

model

alias of *SphinxPage*

__init__(**args*, ***kwargs*) → None

One of our `kwargs` must be `version_id`, the `pk` of the *sphinx_hosting.models.Version* for which we want to list *sphinx_hosting.models.SphinxPage* objects.

This will get added to the `extra_data` dict in the `kwargs` key, from which we reference it.

get_initial_queryset() → QuerySet

Filter our *sphinx_hosting.models.SphinxPage* objects by *version_id*.

actions: `bool` = `True`

Per row action buttons. If not `False`, this will simply add a rightmost column named `Actions` with a button named `default_action_button_label` which when clicked will take the

alignment: Dict[str, str] = {'relative_path': 'left', 'size': 'right', 'title': 'left'}

A mapping of field name to field alignment. Valid values are `left`, `right`, and

fields: List[str] = ['title', 'relative_path', 'size']

This is either `None`, the string `__all__` or a list of column names to use in our table. For the list, entries can either be field names from our *model*, or names of computed fields that will be rendered with a `render_FIELD_column` method. If `None`, empty list

page_length: int = 25

striped: bool = True

version_id: Optional[int]

class sphinx_hosting.wildewidgets.**VersionSphinxImageTable**(**args*, ***kwargs*)

This widget displays a dataTable of our *sphinx_hosting.models.SphinxImage* instances for a particular *sphinx_hosting.models.Version*.

It's used as a the main widget in by *VersionSphinxImageTableWidget*.

model

alias of *SphinxImage*

__init__(**args*, ***kwargs*) → None

One of our `kwargs` must be `version_id`, the `pk` of the *sphinx_hosting.models.Version* for which we want to list *sphinx_hosting.models.SphinxPage* objects.

This will get added to the `extra_data` dict in the `kwargs` key, from which we reference it.

get_initial_queryset() → QuerySet

Filter our *sphinx_hosting.models.SphinxPage* objects by *version_id*.

render_file_path_column(*row:* Version, *column: str*) → str

Render our `file_path` column. This is the path to the file in `MEDIA_ROOT`.

Parameters

- **row** – the `Version` we are rendering
- **column** – the name of the column to render

Returns

The size in bytes of the uploaded file.

render_size_column(*row:* Version, *column: str*) → str

Render our `size` column. This is the size in bytes of the *sphinx_hosting.models.SphinxImage.file* field.

Parameters

- **row** – the `Version` we are rendering
- **column** – the name of the column to render

Returns

The size in bytes of the uploaded file.

alignment: Dict[str, str] = {'file_path': 'left', 'orig_path': 'left', 'size': 'right'}

A mapping of field name to field alignment. Valid values are `left`, `right`, and

fields: List[str] = ['orig_path', 'file_path', 'size']

This is either None, the string __all__ or a list of column names to use in our table. For the list, entries can either be field names from our *model*, or names of computed fields that will be rendered with a render_FIELD_column method. If None, empty list

page_length: int = 25

striped: bool = True

version_id: Optional[int]

10.6 Sphinx Pages

class sphinx_hosting.wildewidgets.**SphinxPageGlobalTableOfContentsMenu**(**items: MenuItem*, *title: Optional[str] = None*, *title_tag: Optional[str] = None*, *title_css_classes: Optional[str] = None*, ***kwargs*)

This is the version-specific navigation menu that gets inserted into the page sidebar when viewing the documentation for a *sphinx_hosting.models.Version*. It will appear on all pages for that version.

classmethod parse_obj(*version:* Version) → *SphinxPageGlobalTableOfContentsMenu*

Parse the globaltoc of a *sphinx_hosting.models.Version* into a wildewidgets.Menu suitable for insertion into a wildewidgets.Navbar

The *sphinx_hosting.models.Version.globaltoc* is a dict that looks like this:

```
{
    items: [
        {'text': 'foo'},
        {'text': 'bar', 'url': '/foo', 'icon': 'blah'}
        {'text': 'bar', 'url': '/foo', 'icon': 'blah', items: [{'text': 'blah' .
↪..} ...]}
        ...
    ]
}
```

Parameters
version – the Version for which we are building the menu

Returns
A configured SphinxPageGlobalTableOfContentsMenu.

css_class: str = 'mt-4'

title_css_classes: str = 'mt-3'

class sphinx_hosting.wildewidgets.**SphinxPageLayout**(*page:* SphinxPage, ***kwargs*)

The page layout for a single *sphinx_hosting.models.SphinxPage*. It consists of a two column layout with the page's table of contents in the left column, and the content of the page in the right column.

Parameters
page – the SphinxPage to render

__init__(*page:* SphinxPage, ***kwargs*)

class sphinx_hosting.wildewidgets.**SphinxPagePagination**(*page:* SphinxPage, ***kwargs*)

This widget draws the "Previous Page", Parent Page and Next Page buttons that are found at the top of each sphinx_hosting.views.SphinxPageDetailView.

It is built out of a Tabler/Bootstrap row, with each of the buttons in an equal sized col.

__init__(*page:* SphinxPage, ***kwargs*)

name: str = 'sphinxpage-pagination'

The CSS class that will be added to this element to as an identifier for

class sphinx_hosting.wildewidgets.**SphinxPageTitle**(*page:* SphinxPage, ***kwargs*)

The title block for a *sphinx_hosting.models.SphinxPage* page.

Parameters
page – the SphinxPage to render

__init__(*page:* SphinxPage, ***kwargs*)

block: str = 'sphinxpage-title'

block is the official wildewidgets name of the block; it can't be changed

css_class: str = 'mb-5'

class sphinx_hosting.wildewidgets.**SphinxPageBodyWidget**(*page:* SphinxPage, ***kwargs*)

This widget holds the body of the page. The body as stored in the model is actually a Django template, so we retrieve the body, run it through the Django template engine, and display the results.

Parameters
page – the SphinxPage we are rendering

__init__(*page:* SphinxPage, ***kwargs*)

css_class: str = 'sphinxpage-body'

class sphinx_hosting.wildewidgets.**SphinxPageTableOfContentsWidget**(*page:* SphinxPage, ***kwargs*)

This widget draws the in-page navigation – the header hierarchy.

Parameters
page – the SphinxPage we are rendering

__init__(*page:* SphinxPage, ***kwargs*)

css_class: str = 'sphinxpage-toc'

CHAPTER

ELEVEN

REST API

11.1 classifiers

`GET /api/v1/classifiers/`

Query Parameters

- **limit** (*integer*) – Number of results to return per page.
- **name** (*string*) – Filter by classifier name [case insensitive, partial match]
- **offset** (*integer*) – The initial index from which to return the results.

Example request:

```
GET /api/v1/classifiers/ HTTP/1.1
Host: example.com
```

Status Codes

- 200 OK – **Example response:**

```
HTTP/1.1 200 OK
Content-Type: application/json

{
    "count": 1,
    "next": "https://example.com",
    "previous": "https://example.com",
    "results": [
        {
            "url": "https://example.com",
            "id": 1,
            "name": "string"
        }
    ]
}
```

`POST /api/v1/classifiers/`

Example request:

```
POST /api/v1/classifiers/ HTTP/1.1
Host: example.com
Content-Type: application/json

{
    "name": "string"
}
```

Status Codes

- 201 Created – **Example response:**

```
HTTP/1.1 201 Created
Content-Type: application/json

{
    "url": "https://example.com",
    "id": 1,
    "name": "string"
}
```

GET /api/v1/classifiers/{id}/

Parameters

- **id** (*integer*) – A unique integer value identifying this classifier.

Example request:

```
GET /api/v1/classifiers/{id}/ HTTP/1.1
Host: example.com
```

Status Codes

- 200 OK – **Example response:**

```
HTTP/1.1 200 OK
Content-Type: application/json

{
    "url": "https://example.com",
    "id": 1,
    "name": "string"
}
```

PUT /api/v1/classifiers/{id}/

Parameters

- **id** (*integer*) – A unique integer value identifying this classifier.

Example request:

```
PUT /api/v1/classifiers/{id}/ HTTP/1.1
Host: example.com
Content-Type: application/json

{
    "name": "string"
}
```

Status Codes

- 200 OK – **Example response:**

```
HTTP/1.1 200 OK
Content-Type: application/json

{
    "url": "https://example.com",
    "id": 1,
    "name": "string"
}
```

`PATCH /api/v1/classifiers/{id}/`

Parameters

- **id** (*integer*) – A unique integer value identifying this classifier.

Example request:

```
PATCH /api/v1/classifiers/{id}/ HTTP/1.1
Host: example.com
Content-Type: application/json

{
    "name": "string"
}
```

Status Codes

- 200 OK – **Example response:**

```
HTTP/1.1 200 OK
Content-Type: application/json

{
    "url": "https://example.com",
    "id": 1,
    "name": "string"
}
```

`DELETE /api/v1/classifiers/{id}/`

Parameters

- **id** (*integer*) – A unique integer value identifying this classifier.

Status Codes

- 204 No Content – No response body

11.2 images

`GET /api/v1/images/`

This is a read-only model set for *sphinx_hosting.models.SphinxImage* models. It is purposely read-only because images are dependent objects of *sphinx_hosting.models.SphinxPage* instances, and it makes no sense to update them independently.

Query Parameters

- **archived** (*boolean*) – Filter by archived status
- **limit** (*integer*) – Number of results to return per page.
- **offset** (*integer*) – The initial index from which to return the results.
- **orig_path** (*string*) – Filter by original path [case insensitive, partial match]
- **project** (*integer*) – Filter by project ID
- **project_classifier** (*string*) – Filter by project classifier name [case insensitive, partial match]
- **project_machine_name** (*string*) – Filter by project machine name [case insensitive, partial match]
- **project_title** (*string*) – Filter by project title [case insensitive, partial match]
- **sphinx_version** (*string*) – Filter by Sphinx version [case insensitive, partial match to start of string]
- **version** (*integer*) – Filter by version ID
- **version_number** (*string*) – Filter by version number [case insensitive, exact match]

Example request:

```
GET /api/v1/images/ HTTP/1.1
Host: example.com
```

Status Codes

- 200 OK – **Example response:**

```
HTTP/1.1 200 OK
Content-Type: application/json

{
    "count": 1,
    "next": "https://example.com",
    "previous": "https://example.com",
    "results": [
        {
            "url": "https://example.com",
            "id": 1,
```

(continues on next page)

(continued from previous page)

```
            "version": [
                "https://example.com"
            ],
            "orig_path": "string"
        }
    ]
}
```

GET /api/v1/images/{id}/

This is a read-only model set for *sphinx_hosting.models.SphinxImage* models. It is purposely read-only because images are dependent objects of *sphinx_hosting.models.SphinxPage* instances, and it makes no sense to update them independently.

Parameters

- **id** (*integer*) – A unique integer value identifying this sphinx image.

Example request:

```
GET /api/v1/images/{id}/ HTTP/1.1
Host: example.com
```

Status Codes

- 200 OK – **Example response:**

```
HTTP/1.1 200 OK
Content-Type: application/json

{
    "url": "https://example.com",
    "id": 1,
    "version": [
        "https://example.com"
    ],
    "orig_path": "string"
}
```

11.3 pages

GET /api/v1/pages/

This is a read-only model set for *sphinx_hosting.models.SphinxPage* models. It is purposely read-only because we only want to update pages in the source Sphinx project, not here in the database.

Even for our derived fields that we built out of the source, pages have a lot of interdependencies that need to be accounted for while editing.

Query Parameters

- **archived** (*boolean*) – Filter by archived status
- **limit** (*integer*) – Number of results to return per page.
- **offset** (*integer*) – The initial index from which to return the results.

- **project** (*integer*) – Filter by project ID
- **project_classifier** (*string*) – Filter by project classifier name [case insensitive, partial match]
- **project_machine_name** (*string*) – Filter by project machine name [case insensitive, partial match]
- **project_title** (*string*) – Filter by project title [case insensitive, partial match]
- **relative_path** (*string*) – Filter by page relative path [case insensitive, partial match]
- **sphinx_version** (*string*) – Filter by Sphinx version [case insensitive, partial match to start of string]
- **title** (*string*) – Filter by page title [case insensitive, partial match]
- **version** (*integer*) – Filter by version ID
- **version_number** (*string*) – Filter by version number [case insensitive, exact match]

Example request:

```
GET /api/v1/pages/ HTTP/1.1
Host: example.com
```

Status Codes

- 200 OK – **Example response:**

```
HTTP/1.1 200 OK
Content-Type: application/json

{
    "count": 1,
    "next": "https://example.com",
    "previous": "https://example.com",
    "results": [
        {
            "url": "https://example.com",
            "id": 1,
            "version": "https://example.com",
            "title": "string",
            "relative_path": "string",
            "content": "string",
            "orig_body": "string",
            "body": "string",
            "orig_local_toc": "string",
            "local_toc": "string",
            "orig_global_toc": "string",
            "searchable": true,
            "parent": "https://example.com",
            "next_page": "https://example.com",
            "previous_page": [
                "https://example.com"
            ]
        }
```

(continues on next page)

(continued from previous page)

```
    ]
}
```

GET /api/v1/pages/{id}/

This is a read-only model set for *sphinx_hosting.models.SphinxPage* models. It is purposely read-only because we only want to update pages in the source Sphinx project, not here in the database.

Even for our derived fields that we built out of the source, pages have a lot of interdependencies that need to be accounted for while editing.

Parameters

- **id** (*integer*) – A unique integer value identifying this sphinx page.

Example request:

```
GET /api/v1/pages/{id}/ HTTP/1.1
Host: example.com
```

Status Codes

- 200 OK – **Example response:**

```
HTTP/1.1 200 OK
Content-Type: application/json

{
    "url": "https://example.com",
    "id": 1,
    "version": "https://example.com",
    "title": "string",
    "relative_path": "string",
    "content": "string",
    "orig_body": "string",
    "body": "string",
    "orig_local_toc": "string",
    "local_toc": "string",
    "orig_global_toc": "string",
    "searchable": true,
    "parent": "https://example.com",
    "next_page": "https://example.com",
    "previous_page": [
        "https://example.com"
    ]
}
```

11.4 projects

`GET /api/v1/projects/`

Query Parameters

- **classifier** (*string*) – Filter by project classifier name [case insensitive, partial match]]
- **description** (*string*) – Filter by project description, [case insensitive, partial match]
- **limit** (*integer*) – Number of results to return per page.
- **machine_name** (*string*) – Filter by project machine name, [case insensitive, partial match]
- **offset** (*integer*) – The initial index from which to return the results.
- **title** (*string*) – Filter by project title, [case insensitive, partial match]

Example request:

```
GET /api/v1/projects/ HTTP/1.1
Host: example.com
```

Status Codes

- 200 OK – **Example response:**

```
HTTP/1.1 200 OK
Content-Type: application/json

{
    "count": 1,
    "next": "https://example.com",
    "previous": "https://example.com",
    "results": [
        {
            "url": "https://example.com",
            "id": 1,
            "title": "string",
            "machine_name": "string",
            "description": "string",
            "related_links": [
                "https://example.com"
            ],
            "classifiers": [
                {
                    "url": "https://example.com",
                    "id": 1,
                    "name": "string"
                }
            ],
            "versions": [
                "https://example.com"
            ]
        }
    ]
}
```

`POST /api/v1/projects/`

Example request:

```
POST /api/v1/projects/ HTTP/1.1
Host: example.com
Content-Type: application/json

{
    "title": "string",
    "description": "string",
    "classifiers": [
        {
            "name": "string"
        }
    ]
}
```

Status Codes

- 201 Created – **Example response:**

```
HTTP/1.1 201 Created
Content-Type: application/json

{
    "url": "https://example.com",
    "id": 1,
    "title": "string",
    "machine_name": "string",
    "description": "string",
    "related_links": [
        "https://example.com"
    ],
    "classifiers": [
        {
            "url": "https://example.com",
            "id": 1,
            "name": "string"
        }
    ],
    "versions": [
        "https://example.com"
    ]
}
```

`GET /api/v1/projects/{id}/`

Parameters

- **id** (*integer*) – A unique integer value identifying this project.

Example request:

```
GET /api/v1/projects/{id}/ HTTP/1.1
Host: example.com
```

Status Codes

- 200 OK – **Example response:**

```
HTTP/1.1 200 OK
Content-Type: application/json

{
    "url": "https://example.com",
    "id": 1,
    "title": "string",
    "machine_name": "string",
    "description": "string",
    "related_links": [
        "https://example.com"
    ],
    "classifiers": [
        {
            "url": "https://example.com",
            "id": 1,
            "name": "string"
        }
    ],
    "versions": [
        "https://example.com"
    ]
}
```

`PUT /api/v1/projects/{id}/`

Parameters

- **id** (*integer*) – A unique integer value identifying this project.

Example request:

```
PUT /api/v1/projects/{id}/ HTTP/1.1
Host: example.com
Content-Type: application/json

{
    "title": "string",
    "description": "string",
    "classifiers": [
        {
            "name": "string"
        }
    ]
}
```

Status Codes

- 200 OK – **Example response:**

```
HTTP/1.1 200 OK
Content-Type: application/json

{
    "url": "https://example.com",
    "id": 1,
    "title": "string",
    "machine_name": "string",
    "description": "string",
    "related_links": [
        "https://example.com"
    ],
    "classifiers": [
        {
            "url": "https://example.com",
            "id": 1,
            "name": "string"
        }
    ],
    "versions": [
        "https://example.com"
    ]
}
```

PATCH /api/v1/projects/{id}/

Parameters

- **id** (*integer*) – A unique integer value identifying this project.

Example request:

```
PATCH /api/v1/projects/{id}/ HTTP/1.1
Host: example.com
Content-Type: application/json

{
    "title": "string",
    "description": "string",
    "classifiers": [
        {
            "name": "string"
        }
    ]
}
```

Status Codes

- 200 OK – **Example response:**

```
HTTP/1.1 200 OK
Content-Type: application/json

{
```

(continues on next page)

(continued from previous page)


```
    "url": "https://example.com",
    "id": 1,
    "title": "string",
    "machine_name": "string",
    "description": "string",
    "related_links": [
        "https://example.com"
    ],
    "classifiers": [
        {
            "url": "https://example.com",
            "id": 1,
            "name": "string"
        }
    ],
    "versions": [
        "https://example.com"
    ]
}
```


`DELETE /api/v1/projects/{id}/`

Parameters

- **id** (*integer*) – A unique integer value identifying this project.

Status Codes

- 204 No Content – No response body

`GET /api/v1/projects/{id}/latest_version/`

Parameters

- **id** (*integer*) – A unique integer value identifying this project.

Example request:

```
GET /api/v1/projects/{id}/latest_version/ HTTP/1.1
Host: example.com
```

Status Codes

- 200 OK – **Example response:**


```
HTTP/1.1 200 OK
Content-Type: application/json

{
    "url": "https://example.com",
    "id": 1,
    "title": "string",
    "machine_name": "string",
    "description": "string",
    "related_links": [
```


(continues on next page)

(continued from previous page)

```
        "https://example.com"
    ],
    "classifiers": [
        {
            "url": "https://example.com",
            "id": 1,
            "name": "string"
        }
    ],
    "versions": [
        "https://example.com"
    ]
}
```

11.5 related-links

`GET /api/v1/related-links/`

Query Parameters

- **limit** (*integer*) – Number of results to return per page.
- **offset** (*integer*) – The initial index from which to return the results.
- **project_classifier** (*string*) – Filter by project classifier name [case insensitive, partial match]]
- **project_description** (*string*) – Filter by project description, [case insensitive, partial match]
- **project_machine_name** (*string*) – Filter by project machine name, [case insensitive, partial match]
- **project_title** (*string*) – Filter by project title, [case insensitive, partial match]
- **title** (*string*) – Filter by link title, [case insensitive, partial match]

Example request:

```
GET /api/v1/related-links/ HTTP/1.1
Host: example.com
```

Status Codes

- 200 OK – **Example response:**

```
HTTP/1.1 200 OK
Content-Type: application/json

{
    "count": 1,
    "next": "https://example.com",
    "previous": "https://example.com",
    "results": [
```

(continues on next page)

(continued from previous page)

```
        {
            "url": "https://example.com",
            "id": 1,
            "title": "string",
            "uri": "https://example.com",
            "project": "https://example.com"
        }
    ]
}
```

`POST /api/v1/related-links/`

Example request:

```
POST /api/v1/related-links/ HTTP/1.1
Host: example.com
Content-Type: application/json

{
    "title": "string",
    "uri": "https://example.com",
    "project": "https://example.com"
}
```

Status Codes

- 201 Created – **Example response:**

```
HTTP/1.1 201 Created
Content-Type: application/json

{
    "url": "https://example.com",
    "id": 1,
    "title": "string",
    "uri": "https://example.com",
    "project": "https://example.com"
}
```

`GET /api/v1/related-links/{id}/`

Parameters

- **id** (*integer*) – A unique integer value identifying this project related link.

Example request:

```
GET /api/v1/related-links/{id}/ HTTP/1.1
Host: example.com
```

Status Codes

- 200 OK – **Example response:**

```
HTTP/1.1 200 OK
Content-Type: application/json

{
    "url": "https://example.com",
    "id": 1,
    "title": "string",
    "uri": "https://example.com",
    "project": "https://example.com"
}
```

PUT /api/v1/related-links/{id}/

Parameters

- **id** (*integer*) – A unique integer value identifying this project related link.

Example request:

```
PUT /api/v1/related-links/{id}/ HTTP/1.1
Host: example.com
Content-Type: application/json

{
    "title": "string",
    "uri": "https://example.com",
    "project": "https://example.com"
}
```

Status Codes

- 200 OK – **Example response:**

```
HTTP/1.1 200 OK
Content-Type: application/json

{
    "url": "https://example.com",
    "id": 1,
    "title": "string",
    "uri": "https://example.com",
    "project": "https://example.com"
}
```

PATCH /api/v1/related-links/{id}/

Parameters

- **id** (*integer*) – A unique integer value identifying this project related link.

Example request:

```
PATCH /api/v1/related-links/{id}/ HTTP/1.1
Host: example.com
Content-Type: application/json
```

(continues on next page)

(continued from previous page)

```
{
    "title": "string",
    "uri": "https://example.com",
    "project": "https://example.com"
}
```

Status Codes

- 200 OK – **Example response:**

```
HTTP/1.1 200 OK
Content-Type: application/json

{
    "url": "https://example.com",
    "id": 1,
    "title": "string",
    "uri": "https://example.com",
    "project": "https://example.com"
}
```

`DELETE /api/v1/related-links/{id}/`

Parameters

- **id** (*integer*) – A unique integer value identifying this project related link.

Status Codes

- 204 No Content – No response body

11.6 schema

`GET /api/v1/schema/`

OpenApi3 schema for this API. Format can be selected via content negotiation.

- YAML: application/vnd.oai.openapi
- JSON: application/vnd.oai.openapi+json

Query Parameters

- **format** (*string*) –

Example request:

```
GET /api/v1/schema/ HTTP/1.1
Host: example.com
```

Status Codes

- 200 OK – **Example response:**

```
HTTP/1.1 200 OK
Content-Type: application/json

{}
```

11.7 version

POST /api/v1/version/import/

This is the view to use to upload our sphinx tarballs. It uploads to a temporary directory that disappears at the end of this view.

To upload a file, you must submit as form-data, with a single file key named `file`, with the `Content-Disposition` header like so:

```
Content-Disposition: attachment;filename=yourdocs.tar.gz
```

The filename you pass in the `Content-Disposition` header does not matter and is not used; set it to whatever you want.

Example:

To upload a file with `curl` to the endpoint for this view:

```
curl \
    -XPOST \
    -H "Authorization: Token __THE_API_TOKEN__" \
    -F 'file=@path/to/yourdocs.tar.gz' \
    https://sphinx-hosting.example.com/api/v1/version/import/
```

Status Codes

- 200 OK – **Example response:**

```
HTTP/1.1 200 OK
Content-Type: application/json

{
    "file": "https://example.com"
}
```

11.8 versions

GET /api/v1/versions/

Users can get, list and delete *sphinx_hosting.models.Version* objects, but they can't create or update them the normal Django way.

Query Parameters

- **archived** (*boolean*) – Filter by archived status
- **limit** (*integer*) – Number of results to return per page.

- **offset** (*integer*) – The initial index from which to return the results.
- **project** (*integer*) –
- **project_classifier** (*string*) – Filter by project classifier name [case insensitive, partial match]
- **project_machine_name** (*string*) – Filter by project machine name [case insensitive, partial match]
- **project_title** (*string*) – Filter by project title [case insensitive, partial match]
- **sphinx_version** (*string*) – Filter by Sphinx version [case insensitive, partial match to start of string]
- **version** (*string*) –
- **version_number** (*string*) – Filter by version number [case insensitive, exact match]

Example request:

```
GET /api/v1/versions/ HTTP/1.1
Host: example.com
```

Status Codes

- 200 OK – **Example response:**

```
HTTP/1.1 200 OK
Content-Type: application/json

{
    "count": 1,
    "next": "https://example.com",
    "previous": "https://example.com",
    "results": [
        {
            "url": "https://example.com",
            "id": 1,
            "project": "https://example.com",
            "version": "string",
            "sphinx_version": "string",
            "archived": true,
            "head": "https://example.com",
            "pages": [
                "https://example.com"
            ],
            "images": [
                "https://example.com"
            ]
        }
    ]
}
```

GET /api/v1/versions/{id}/

Users can get, list and delete *sphinx_hosting.models.Version* objects, but they can't create or update them the normal Django way.

Parameters

- **id** (*integer*) – A unique integer value identifying this version.

Example request:

```
GET /api/v1/versions/{id}/ HTTP/1.1
Host: example.com
```

Status Codes

- 200 OK – **Example response:**

```
HTTP/1.1 200 OK
Content-Type: application/json

{
    "url": "https://example.com",
    "id": 1,
    "project": "https://example.com",
    "version": "string",
    "sphinx_version": "string",
    "archived": true,
    "head": "https://example.com",
    "pages": [
        "https://example.com"
    ],
    "images": [
        "https://example.com"
    ]
}
```

`DELETE /api/v1/versions/{id}/`

Users can get, list and delete *sphinx_hosting.models.Version* objects, but they can't create or update them the normal Django way.

Parameters

- **id** (*integer*) – A unique integer value identifying this version.

Status Codes

- 204 No Content – No response body

Current version is 1.3.1.

This reusable Django application provides models, views, permissions, REST API endpoints and management commands for making a private Sphinx documentation hosting platform.

This is useful for when you want Sphinx documentation for your internal software projects, but you do not want that documentation do be shared with a third-party.

CHAPTER

TWELVE

INSTALLATION

To install from PyPI:

```
pip install django-sphinx-hosting
```

If you want, you can run the tests:

```
python -m unittest discover
```

CHAPTER

THIRTEEN

FEATURES

- Users must be authenticated to view docs
- Multiple levels of privileges within the system based on Django authentication
- Manage multiple versions of your docs per project
- Automatically build and display navigation for each version of your documentaion
- Renders all documentation published within with a consistent theme
- Tag projects with classifiers to refine searching and filtering
- Search across all projects
- Use REST API to programmatically interact with the system. Useful for integrating into a CI/CD system

CHAPTER

FOURTEEN

CONFIGURATION

14.1 Update `INSTALLED_APPS`

As with most Django applications, you should add `django-sphinx-hosting` and its key dependencies to the `INSTALLED_APPS` within your settings file (usually `settings.py`).

Example:

```
INSTALLED_APPS = [
   'django.contrib.admin',
   'django.contrib.auth',
   'django.contrib.contenttypes',
   'django.contrib.messages',
   'django.contrib.sessions',
   'django.contrib.sites',

   # ---------- add these ----------------
   'rest_framework',
   'rest_framework.authtoken',
   'django_filters',
   'drf_spectacular',
   'crispy_forms',
   'crispy_bootstrap5',
   'haystack',
   'academy_theme',
   'wildewidgets',
   'sphinx_hosting',
   'sphinx_hosting.api'
   # -------- done add these -------------

   # Then your usual apps...
   'blog',
]
```

14.2 Configure `django-sphinx-hosting` itself

For `django-sphinx-hosting` itself, you'll typically want to add a `SPHINX_HOSTING_SETTINGS` dict to localize django-sphinx-hosting to your organization.

Full example:

```
SPHINX_HOSTING_SETTINGS = {
   'LOGO_IMAGE': 'core/images/my-org-logo.png',
   'LOGO_WIDTH': '75%',
   'LOGO_URL': 'https://www.example.com',
   'SITE_NAME': 'MyOrg Documentation'
}
```

LOGO_IMAGE
: A Django filesystem path to the image you want to use for the logo at the top of the navigation sidebar.

LOGO_WIDTH
: Any valid CSS width specifier. This will be applied to the `LOGO_IMAGE`.

LOGO_URL
: When a user clicks the `LOGO_IMAGE`, they'll be sent to this URL.

SITE_NAME
: This will be used in the HTML title tags for each page, and wil be used as the `alt` tag for the `LOGO_IMAGE`.

14.3 Configure django-wildewidgets

All HTML in `django-sphinx-hosting` is generated by django-wildewidgets widgets. We use the django-theme-academy to provide the HTML boilerplate to house the widget output, and django-crispy-forms for our form rendering.

Add this code to your `settings.py` to configure them properly:

```
# crispy-forms
# ------------------------------------------------------------------------------
CRISPY_ALLOWED_TEMPLATE_PACKS = 'bootstrap5'
CRISPY_TEMPLATE_PACK = 'bootstrap5'

# django-theme-academy
# ------------------------------------------------------------------------------
ACADEMY_THEME_SETTINGS = {
   # Header
   'APPLE_TOUCH_ICON': 'core/images/apple-touch-icon.png',
   'FAVICON_32': 'core/images/favicon-32x32.png',
   'FAVICON_16': 'core/images/favicon-16x16.png',
   'FAVICON': 'core/images/favicon.ico',
   'SITE_WEBMANIFEST': 'core/images/site.webmanifest',

   # Footer
   'ORGANIZATION_LINK': 'https://github.com/caltechads/django-sphinx-hosting',
   'ORGANIZATION_NAME': 'Sphinx Hosting',
   'ORGANIZATION_ADDRESS': '123 Main Street, Everytown, ST',
   'COPYRIGHT_ORGANIZATION': 'Sphinx Hosting',
   'FOOTER_LINKS': [
```

(continues on next page)

(continued from previous page)

```
        ('https://example.com', 'Organization Home'),
        ('https://example.com/documents/privacy.pdf', "Privacy Policy")
    ]
}

# django-wildewidgets
# ------------------------------------------------------------------------------
WILDEWIDGETS_DATETIME_FORMAT = "%Y-%m-%d %H:%M %Z"
```

For `ACADEMY_THEME_SETTINGS`, localize to your organization by updating all the settings appropriately.

`FAVICON_*`, `APPLE_TOUCH_ICON` and `SITE_WEBMANIFEST`
: Set these to the Django filesystem path to the files you want to use.

`FOOTER_LINKS`
: Add links to the footer by listing 2-tuples of `("__URL__", "__LABEL__")`

14.4 Configure Haystack

We use django-haystack to support our documentation search feature, but it is up to you what specific backend you want to configure. See Haystack: Configuration for instructions on how to configure Haystack for different backends.

Here is example `settings.py` code for using Elasticsearch 7.x as our search backend:

```
HAYSTACK_CONNECTIONS = {
    'default': {
        'ENGINE': 'haystack.backends.elasticsearch7_backend.Elasticsearch7SearchEngine',
        'URL': 'http://sphinx-hosting-search.example.com:9200/',
        'INDEX_NAME': 'sphinx_hosting',
    },
}
```

If you want your search index to be updated automatically when versions of your documentation are uploaded, add this to `settings.py`:

```
# This will cause the search index to be updated whenever a SphinxPage is
# saved or deleted.
HAYSTACK_SIGNAL_PROCESSOR = 'sphinx_hosting.signals.SphinxHostingSignalProcessor'
```

14.5 Configure Django REST Framework

To make the `django-sphinx-hosting` API work, add this code to your `settings.py`:

```
# djangorestframework
# ------------------------------------------------------------------------------
REST_FRAMEWORK = {
    # https://www.django-rest-framework.org/api-guide/parsers/#setting-the-parsers
    'DEFAULT_PARSER_CLASSES': ('rest_framework.parsers.JSONParser',),
    # https://www.django-rest-framework.org/api-guide/authentication/#tokenauthentication
    'DEFAULT_AUTHENTICATION_CLASSES': ('rest_framework.authentication.TokenAuthentication
```

(continues on next page)

(continued from previous page)

```
↪',),
   # https://django-filter.readthedocs.io/en/master/guide/rest_framework.html
   'DEFAULT_FILTER_BACKENDS': ('django_filters.rest_framework.DjangoFilterBackend',),
   # https://www.django-rest-framework.org/api-guide/pagination/#limitoffsetpagination
   'DEFAULT_PAGINATION_CLASS': 'rest_framework.pagination.LimitOffsetPagination',
   # https://github.com/tfranzel/drf-spectacular
   'DEFAULT_SCHEMA_CLASS': 'drf_spectacular.openapi.AutoSchema',
   'PAGE_SIZE': 100,
}

# drf-spectacular
# ------------------------------------------------------------------------------
# https://drf-spectacular.readthedocs.io/en/latest/settings.html
SPECTACULAR_SETTINGS = {
   'SCHEMA_PATH_PREFIX': r'/api/v1',
   'SERVERS': [
      {
            'url': 'https://localhost',
            'description': 'Django Sphinx Hosting'
      }
   ],
   'TITLE': 'YOUR_SITE_NAME'
   'VERSION': __version__,
   'DESCRIPTION': """__YOUR_DESCRIPTION__HERE"""
}
```

For `DEFAULT_AUTHENTICATION_CLASSES`, use whatever authentication class you want – we're just using `TokenAuthentication` here as an example. For the rest of the settings, you'll need at least what we've detailed above, but feel free to add to them if you have additional API views you need to support in your own application code.

14.6 Update your top-level urlconf

```
from django.urls import path, include

from sphinx_hosting import urls as sphinx_hosting_urls
from sphinx_hosting.api import urls as sphinx_hosting_api_urls
from wildewidgets import WildewidgetDispatch

urlpatterns = [
   path('/docs/', include(sphinx_hosting_urls, namespace='sphinx_hosting')),
   path('/docs/wildewidgets_json', WildewidgetDispatch.as_view(), name='wildewidgets_json
↪'),
   path('api/v1/', include(sphinx_hosting_api_urls, namespace='sphinx_hosting_api')),
]
```

PYTHON MODULE INDEX

S

HTTP ROUTING TABLE

/api

INDEX

Symbols

G

H

I

J

L

M

N

O

P

R

S

T

U

V

W