Read the Docs Documentation

Release 1.0

Eric Holscher, Charlie Leifer, Bobby Grace

Feb 19, 2018

User Documentation

Read the Docs hosts documentation for the open source community. We support Sphinx docs written with reStructuredText and CommonMark. We pull your code from your Subversion, Bazaar, Git, and Mercurial repositories. Then we build documentation and host it for you. Think of it as *Continuous Documentation*.

The code is open source, and available on GitHub.

The main documentation for the site is organized into a couple sections:

- *User Documentation*
- *Feature Documentation*
- *About Read the Docs*

Information about development is also available:

- *Developer Documentation*
- *Designer Documentation*

CHAPTER 1

Getting Started

This document will show you how to get up and running with Read the Docs. You will have your docs imported on Read the Docs in 5 minutes, displayed beautifully for the world.

If you are already using Sphinx or Markdown for your docs, skip ahead to *Import Your Docs*.

Write Your Docs

You have two options for formatting your documentation:

- *In reStructuredText*
- *In Markdown*

In reStructuredText

There is a screencast that will help you get started if you prefer.

Sphinx is a tool that makes it easy to create beautiful documentation. Assuming you have Python already, install Sphinx:

```
$ pip install sphinx sphinx-autobuild
```

Create a directory inside your project to hold your docs:

```
$ cd /path/to/project
$ mkdir docs
```

Run `sphinx-quickstart` in there:

```
$ cd docs
$ sphinx-quickstart
```

This quick start will walk you through creating the basic configuration; in most cases, you can just accept the defaults. When it's done, you'll have an `index.rst`, a `conf.py` and some other files. Add these to revision control.

Now, edit your `index.rst` and add some information about your project. Include as much detail as you like (refer to the reStructuredText syntax or this template if you need help). Build them to see how they look:

```
$ make html
```

Note: You can use `sphinx-autobuild` to auto-reload your docs. Run `sphinx-autobuild . _build/html` instead.

Edit your files and rebuild until you like what you see, then commit your changes and push to your public repository. Once you have Sphinx documentation in a public repository, you can start using Read the Docs.

In Markdown

You can use Markdown and reStructuredText in the same Sphinx project. We support this natively on Read the Docs, and you can do it locally:

```
$ pip install recommonmark
```

Then in your `conf.py`:

```
from recommonmark.parser import CommonMarkParser

source_parsers = {
    '.md': CommonMarkParser,
}

source_suffix = ['.rst', '.md']
```

Note: Markdown doesn't support a lot of the features of Sphinx, like inline markup and directives. However, it works for basic prose content. reStructuredText is the preferred format for technical documentation, please read this blog post for motivation.

Sign Up and Connect an External Account

If you are going to import a repository from GitHub, Bitbucket or GitLab, you should connect your account to your provider first. Connecting your account allows for easier importing and enables Read the Docs to configure your repository webhooks automatically.

To connect your account, go to your *Settings* dashboard and select *Connected Services*. From here, you'll be able to connect to your GitHub, Bitbucket or GitLab account. This process will ask you to authorize a connection to Read the Docs, that allows us to read information about and clone your repositories.

Import Your Docs

To import a repository, visit your dashboard and click Import.

If you have a connected account, you will see a list of your repositories that we are able to import. To import one of these projects, just click the import icon next to the repository you'd like to import. This will bring up a form that is already filled with your project's information. Feel free to edit any of these properties, and the click **Next** to build your documentation.

Manually Import Your Docs

If you do not have a connected account, you will need select **Import Manually** and enter the information for your repository yourself. You will also need to manually configure the webhook for your repository as well. When importing your project, you will be asked for the repository URL, along with some other information for you new project. The URL is normally the URL or path name you'd use to checkout, clone, or branch your repository. Some examples:

- Git: `http://github.com/ericholscher/django-kong.git`
- Mercurial: `https://bitbucket.org/ianb/pip`
- Subversion: `http://varnish-cache.org/svn/trunk`
- Bazaar: `lp:pasta`

Add an optional homepage URL and some tags, and then click **Next**.

Once your project is created, you'll need to manually configure the repository webhook if you would like to have new changesets to trigger builds for your project on Read the Docs. Go to your project's **Integrations** page to configure a new webhook, or see *our steps for webhook creation* for more information on this process.

Within a few seconds your code will automatically be fetched from your public repository, and the documentation will be built. Check out our *Build Process* page to learn more about how we build your docs, and to troubleshoot any issues that arise.

Read the Docs will host multiple versions of your code. You can read more about how to use this well on our *Versions* page.

If you have any more trouble, don't hesitate to reach out to us. The *Support* page has more information on getting in touch.

CHAPTER 2

Versions

Read the Docs supports multiple versions of your repository. On the initial import, we will create a `latest` version. This will point at the default branch for your VCS control: `master`, `default`, or `trunk`.

We also create a `stable` version, if your project has any tagged releases. `stable` will be automatically kept up to date to point at your highest version.

How we envision versions working

In the normal case, the `latest` version will always point to the most up to date development code. If you develop on a branch that is different than the default for your VCS, you should set the **Default Branch** to that branch.

You should push a **tag** for each version of your project. These tags should be numbered in a way that is consistent with semantic versioning. This will map to your `stable` branch by default.

Note: We in fact are parsing your tag names against the rules given by PEP 440. This spec allows "normal" version numbers like `1.4.2` as well as pre-releases. An alpha version or a release candidate are examples of pre-releases and they look like this: `2.0a1`.

We only consider non pre-releases for the `stable` version of your documentation.

If you have documentation changes on a **long-lived branch**, you can build those too. This will allow you to see how the new docs will be built in this branch of the code. Generally you won't have more than 1 active branch over a long period of time. The main exception here would be **release branches**, which are branches that are maintained over time for a specific release number.

Tags and branches

Read the Docs supports two workflows for versioning: based on tags or branches. If you have at least one active tag, tags will take preference over branches when selecting the stable version.

Redirects on root URLs

When a user hits the root URL for your documentation, for example `http://pip.readthedocs.io/`, they will be redirected to the **Default version**. This defaults to **latest**, but could also point to your latest released version.

CHAPTER 3

Build Process

Files: tasks.py - doc_builder/

Every documentation build has limited resources. Our current build limits are:

- 15 minutes
- 1GB of memory

We can increase build limits on a per-project basis, if you provide a good reason your documentation needs more resources.

You can see the current Docker build images that we use in our docker repository. Docker Hub also shows the latest set of images that have been built.

Currently in production we're using the `readthedocs/build:2.0` docker image as our default image.

How we build documentation

When we import your documentation, we look at two things first: your *Repository URL* and the *Documentation Type*. We will clone your repository, and then build your documentation using the *Documentation Type* specified.

Sphinx

When you choose *Sphinx* as your *Documentation Type*, we will first look for a `conf.py` file in your repository. If we don't find one, we will generate one for you. We will look inside a `doc` or `docs` directory first, and then look within your entire project.

Then Sphinx will build any files with an `.rst` extension.

MkDocs

When you choose *Mkdocs* as your *Documentation Type*, we will first look for a `mkdocs.yml` file in your repository. If we don't find one, we will generate one for you. We will look inside a `doc` or `docs` directory first, and then default to the top-level of your documentation.

Then MkDocs will build any files with a `.md` extension. As MkDocs doesn't support automatic PDF generation, Read the Docs cannot create a PDF version of your documentation with the *Mkdocs* option.

Understanding what's going on

Understanding how Read the Docs builds your project will help you with debugging the problems you have with the site. It should also allow you to take advantage of certain things that happen during the build process.

The first step of the process is that we check out your code from the repository you have given us. If the code is already checked out, we update the copy to the branch that you have specified in your projects configuration.

Then we build the proper backend code for the type of documentation you've selected.

If you have the *Install Project* option enabled, we will run `setup.py install` on your package, installing it into a virtual environment. You can also define additional packages to install with the *Requirements File* option.

When we build your documentation, we run `sphinx-build -b html . _build/html`, where `html` would be replaced with the correct backend. We also create man pages and pdf's automatically based on your project.

Then these files are copied across to our application servers from the build server. Once on the application servers, they are served from nginx.

An example in code:

```
update_docs_from_vcs(version)
if exists('setup.py'):
    run('python setup.py install')
if project.requirements_file:
    run('pip install -r %s' % project.requirements_file)
build_docs(version=version)
copy_files(artifact_dir)
```

Builder Responsibility

Builders have a very specific job. They take the updated source code and generate the correct artifacts. The code lives in `self.version.project.checkout_path(self.version.slug)`. The artifacts should end up in `self.version.project.artifact_path(version=self.version.slug, type=self.type)` Where `type` is the name of your builder. All files that end up in the artifact directory should be in their final form.

Packages installed in the build environment

The build server does have a select number of C libraries installed, because they are used across a wide array of python projects. We can't install every C library out there, but we try and support the major ones. We currently have the following libraries installed:

- doxygen

- LaTeX (texlive-full)
- libevent (libevent-dev)
- dvipng
- graphviz
- libxslt1.1
- libxml2-dev

Writing your own builder

Note: Builds happen on a server using only the RTD Public API. There is no reason that you couldn't build your own independent builder that wrote into the RTD namespace. The only thing that is currently unsupported there is a saner way than uploading the processed files as a zip.

The documentation build system in RTD is made pluggable, so that you can build out your own backend. If you have a documentation format that isn't currently supported, you can add support by contributing a backend.

The *readthedocs.doc_builder* API explains the higher level parts of the API that you need to implement. A basic run goes something like this:

```
backend = get_backend(project.documentation_type)
if force:
    backend.force(version)
backend.clean(version)
backend.build(version)
if success:
    backend.move(version)
```

Deleting a stale or broken build environment

If you're having trouble getting your version to build, try wiping out the existing build/environment files. On your version list page `/projects/[project]/versions` there is a "Wipe" button that will remove all of the files associated with your documentation build, but not the documentation itself.

Build environment

The *Sphinx* and *Mkdocs* builders set the following RTD-specific environment variables when building your documentation:

Environment variable	Description	Example value
`READTHEDOCS`	Whether the build is running inside RTD	`True`
`READTHEDOCS_VERSION`	The RTD name of the version which is being built	`latest`
`READTHEDOCS_PROJECT`	The RTD name of the project which is being built	`myexampleproject`

CHAPTER 4

Read the Docs features

This will serve as a list of all of the features that Read the Docs currently has. Some features are important enough to have their own page in the docs, others will simply be listed here.

GitHub, Bitbucket and GitLab Integration

We now support linking by default in the sidebar. It links to the page on your host, which should help people quickly update typos and send pull requests to contribute to project documentation.

More information can be found in the *Version Control System Integration* page.

Auto-updating

The *Webhooks* page talks about the different ways you can ping RTD to let us know your project has been updated. We have official support for GitHub, Bitbucket and GitLab, and anywhere else we have a generic post-commit hook that allows you to POST to a URL to get your documentation built.

Internationalization

Read the Docs itself is localized, and we support documentation translated into multiple languages. Read more on the *Localization of Documentation* and *Internationalization* pages.

Canonical URLs

Canonical URLs give your docs better search performance, by pointing all URLs to one version. This also helps to solve the issues around users landing on outdated versions of documentation.

More information can be found in the *Canonical URLs* page.

Versions

We can build multiple versions of your documentation. Look on the "Versions" page of your project's admin (using the nav on the left) to find a list of available versions that we've inferred from the tags and branches in your source control system (according to the support matrix below). On the Versions page you can tell us which versions you'd like us to build docs for, whether each should be public, protected, or private, and what the default version should be (we'll redirect there when someone hits your main project page, e.g., http://my-project.rtfd.org/).

Version Control Support Matrix

	Git	hg	bzr	svn
Tags	Yes	Yes	Yes	No
Branches	Yes	Yes	Yes	No
Default	master	default		trunk

PDF Generation

When you build your project on RTD, we automatically build a PDF of your project's documentation. We also build them for every version that you upload, so we can host the PDFs of your latest documentation, as well as your latest stable releases as well.

Search

We provide full-text search across all of the pages of documentation hosted on our site. This uses the excellent Haystack project and Solr as the search backend. We hope to be integrating this into the site more fully in the future.

Alternate Domains

We provide support for CNAMEs, subdomains, and a shorturl for your project as well. This is outlined in the *Alternate Domains* section.

CHAPTER 5

Support

Usage Questions

If you have questions about how to use Read the Docs, or have an issue that isn't related to a bug, Stack Overflow is the best place to ask. Tag questions with `read-the-docs` so other folks can find them easily.

Good questions for Stack Overflow would be:

- "What is the best way to structure the table of contents across a project?"
- "How do I structure translations inside of my project for easiest contribution from users?"
- "How do I use Sphinx to use SVG images in HTML output but PNG in PDF output?"

Community Support

Read the Docs is a community supported site, nobody is paid to handle readthedocs.org support. We are hoping to bring in enough money with our Gold program to change that, so please sign up if you are able.

All people answering your questions are doing it with their own time, so please be kind and provide as much information as possible.

Bugs & Support Issues

You can file bug reports on our GitHub issue tracker, and they will be addressed as soon as possible. **Support is a volunteer effort**, and there is no guaranteed response time. If you need answers quickly, you can buy commercial support below.

Reporting Issues

When reporting a bug, please include as much information as possible that will help us solve this issue. This includes:

- Project name
- URL
- Action taken
- Expected result
- Actual result

Commercial Support

We offer commercial support for Read the Docs, commercial hosting, as well as consulting around all documentation systems. You can contact us at hello@readthedocs.com to learn more, or read more at https://readthedocs.com/services/#open-source-support.

CHAPTER 6

Frequently Asked Questions

My project isn't building with autodoc

First, you should check out the Builds tab of your project. That records all of the build attempts that RTD has made to build your project. If you see `ImportError` messages for custom Python modules, you should enable the virtualenv feature in the Admin page of your project, which will install your project into a virtualenv, and allow you to specify a `requirements.txt` file for your project.

If you are still seeing errors because of C library dependencies, please see the below section about that.

How do I change my slug (the URL your docs are served at)?

We don't support allowing folks to change the slug for their project. You can update the name which is shown on the site, but not the actual URL that documentation is served.

The main reason for this is that all existing URLs to the content will break. You can delete and re-create the project with the proper name to get a new slug, but you really shouldn't do this if you have existing inbound links, as it breaks the internet.

Help, my build passed but my documentation page is 404 Not Found!

This often happens because you don't have an `index.html` file being generated. Make sure you have one of the following files:

- `index.rst`
- `index.md`

At the top level of your built documentation, otherwise we aren't able to serve a "default" index page.

To test if your docs actually built correctly, you can navigate to a specific page (`/en/latest/README.html` for example).

How do I change behavior for Read the Docs?

When RTD builds your project, it sets the `READTHEDOCS` environment variable to the string `True`. So within your Sphinx `conf.py` file, you can vary the behavior based on this. For example:

```
import os
on_rtd = os.environ.get('READTHEDOCS') == 'True'
if on_rtd:
    html_theme = 'default'
else:
    html_theme = 'nature'
```

The `READTHEDOCS` variable is also available in the Sphinx build environment, and will be set to `True` when building on RTD:

```
{% if READTHEDOCS %}
Woo
{% endif %}
```

I get import errors on libraries that depend on C modules

Note: Another use case for this is when you have a module with a C extension.

This happens because our build system doesn't have the dependencies for building your project. This happens with things like libevent and mysql, and other python things that depend on C libraries. We can't support installing random C binaries on our system, so there is another way to fix these imports.

You can mock out the imports for these modules in your `conf.py` with the following snippet:

```
import sys
from unittest.mock import MagicMock

class Mock(MagicMock):
    @classmethod
    def __getattr__(cls, name):
            return MagicMock()

MOCK_MODULES = ['pygtk', 'gtk', 'gobject', 'argparse', 'numpy', 'pandas']
sys.modules.update((mod_name, Mock()) for mod_name in MOCK_MODULES)
```

Of course, replacing `MOCK_MODULES` with the modules that you want to mock out.

Tip: The library `unittest.mock` was introduced on python 3.3. On earlier versions install the `mock` library from PyPI with (ie `pip install mock`) and replace the above import:

```
from mock import Mock as MagicMock
```

If such libraries are installed via `setup.py`, you also will need to remove all the C-dependent libraries from your `install_requires` in the RTD environment.

`Client Error 401` when building documentation

If you did not install the `test_data` fixture during the installation instructions, you will get the following error:

```
slumber.exceptions.HttpClientError: Client Error 401: http://localhost:8000/api/v1/
↪version/
```

This is because the API admin user does not exist, and so cannot authenticate. You can fix this by loading the test_data:

```
./manage.py loaddata test_data
```

If you'd prefer not to install the test data, you'll need to provide a database account for the builder to use. You can provide these credentials by editing the following settings:

```
SLUMBER_USERNAME = 'test'
SLUMBER_PASSWORD = 'test'
```

Deleting a stale or broken build environment

See *Wiping a Build Environment*.

How do I host multiple projects on one CNAME?

We support the concept of Subprojects. If you add a subproject to a project, that documentation will also be served under the parent project's subdomain.

For example, Kombu is a subproject of Celery, so you can access it on the `celery.readthedocs.io` domain:

http://celery.readthedocs.io/projects/kombu/en/latest/

This also works the same for CNAMEs:

http://docs.celeryproject.org/projects/kombu/en/latest/

You can add subprojects in the Admin section for your project.

Where do I need to put my docs for RTD to find it?

Read the Docs will crawl your project looking for a `conf.py`. Where it finds the `conf.py`, it will run `sphinx-build` in that directory. So as long as you only have one set of sphinx documentation in your project, it should Just Work.

I want to use the Blue/Default Sphinx theme

We think that our theme is badass, and better than the default for many reasons. Some people don't like change though :), so there is a hack that will let you keep using the default theme. If you set the `html_style` variable in your `conf.py`, it should default to using the default theme. The value of this doesn't matter, and can be set to `/default.css` for default behavior.

I want to use the Read the Docs theme locally

There is a repository for that: https://github.com/snide/sphinx_rtd_theme. Simply follow the instructions in the README.

Image scaling doesn't work in my documentation

Image scaling in docutils depends on PIL. PIL is installed in the system that RTD runs on. However, if you are using the virtualenv building option, you will likely need to include PIL in your requirements for your project.

I want comments in my docs

RTD doesn't have explicit support for this. That said, a tool like Disqus (and the sphinxcontrib-disqus plugin) can be used for this purpose on RTD.

How do I support multiple languages of documentation?

See the section on *Localization of Documentation*.

Does Read The Docs work well with "legible" docstrings?

Yes. One criticism of Sphinx is that its annotated docstrings are too dense and difficult for humans to read. In response, many projects have adopted customized docstring styles that are simultaneously informative and legible. The NumPy and Google styles are two popular docstring formats. Fortunately, the default Read The Docs theme handles both formats just fine, provided your `conf.py` specifies an appropriate Sphinx extension that knows how to convert your customized docstrings. Two such extensions are numpydoc and napoleon. Only `napoleon` is able to handle both docstring formats. Its default output more closely matches the format of standard Sphinx annotations, and as a result, it tends to look a bit better with the default theme.

Can I document a python package that is not at the root of my repository?

Yes. The most convenient way to access a python package for example via Sphinx's autoapi in your documentation is to use the *Install your project inside a virtualenv using ''setup.py install''* option in the admin panel of your project. However this assumes that your `setup.py` is in the root of your repository.

If you want to place your package in a different directory or have multiple python packages in the same project, then create a pip requirements file. You can specify the relative path to your package inside the file. For example you want to keep your python package in the `src/python` directory, then create a `requirements.readthedocs.txt` file with the following contents:

```
src/python/
```

Please note that the path must be relative to the file. So the example path above would work if the file is in the root of your repository. If you want to put the requirements in a file called `requirements/readthedocs.txt`, the contents would look like:

```
../python/
```

After adding the file to your repository, go to the *Advanced Settings* in your project's admin panel and add the name of the file to the *Requirements file* field.

What commit of Read the Docs is in production?

We deploy readthedocs.org from the `rel` branch in our GitHub repository. You can see the latest commits that have been deployed by looking on GitHub: https://github.com/rtfd/readthedocs.org/commits/rel

CHAPTER 7

Read the Docs YAML Config

Read the Docs now has support for configuring builds with a YAML file. The file, `readthedocs.yml` (or `.readthedocs.yml`), must be in the root directory of your project.

Warning: This feature is in a beta state. Please file an issue if you find anything wrong.

Supported Settings

formats

- Default: `htmlzip`, `pdf`, `epub`
- Options: `htmlzip`, `pdf`, `epub`, `none`

The formats of your documentation you want to be built. Choose `none` to build none of the formats.

Note: We will always build an HTML & JSON version of your documentation. These are used for web serving & search indexing, respectively.

```
# Don't build any extra formats
formats:
    - none
```

```
# Build PDF & ePub
formats:
    - epub
    - pdf
```

requirements_file

- Default: `None`
- Type: Path (specified from the root of the project)

The path to your Pip requirements file.

```
requirements_file: requirements/docs.txt
```

conda

The `conda` block allows for configuring our support for Conda.

conda.file

- Default: `None`
- Type: Path (specified from the root of the project)

The file option specified the Conda environment file to use.

```
conda:
    file: environment.yml
```

Note: Conda is only supported via the YAML file.

build

The `build` block configures specific aspects of the documentation build.

build.image

- Default: `readthedocs/build:2.0`
- Options: `1.0`, `2.0`, `latest`

The build image to use for specific builds. This lets users specify a more experimental build image, if they want to be on the cutting edge.

Certain Python versions require a certain build image, as defined here:

- `1.0`: 2, 2.7, 3, 3.4
- `2.0`: 2, 2.7, 3, 3.5
- `latest`: 2, 2.7, 3, 3.3, 3.4, 3.5, 3.6

```
build:
    image: latest

python:
    version: 3.6
```

python

The `python` block allows you to configure aspects of the Python executable used for building documentation.

python.version

- Default: `2.7`
- Options: `2.7`, `2`, `3.5`, `3`

This is the version of Python to use when building your documentation. If you specify only the major version of Python, the highest supported minor version will be selected.

Warning: The supported Python versions depends on the version of the build image your project is using. The default build image that is used to build documentation contains support for Python `2.7` and `3.5`. See the *build.image* for more information on supported Python versions.

```
python:
   version: 3.5
```

python.setup_py_install

- Default: `False`
- Type: Boolean

When true, install your project into the Virtualenv with `python setup.py install` when building documentation.

```
python:
   setup_py_install: true
```

python.pip_install

- Default: `False`
- Type: Boolean

When true, install your project into the Virtualenv with pip when building documentation.

```
python:
   pip_install: true
```

python.extra_requirements

- Default: `[]`
- Type: List

List of extra requirements sections to install, additionally to the package default dependencies. Only works if `python.pip_install` option above is set to `True`.

Let's say your Python package has a `setup.py` which looks like this:

```
from setuptools import setup

setup(
    name="my_package",
    # (...)
    install_requires=[
        'requests',
        'simplejson'],
    extras_require={
        'tests': [
            'nose',
            'pycodestyle >= 2.1.0'],
        'docs': [
            'sphinx >= 1.4',
            'sphinx_rtd_theme']}
)
```

Then to have all dependencies from the `tests` and `docs` sections installed in addition to the default `requests` and `simplejson`, use the `extra_requirements` as such:

```
python:
    extra_requirements:
        - tests
        - docs
```

Behind the scene the following Pip command will be run:

```
$ pip install -e .[tests,docs]
```

CHAPTER 8

Guides

These guides will help walk you through the usage of Read the Docs.

Adding Custom CSS or JavaScript to a Sphinx Project

The easiest way to add custom stylesheets or scripts, and ensure that the files are added in a way that allows for overriding of existing styles or scripts, is to add these files using a `conf.py` Sphinx extension. Inside your `conf.py`, if a function `setup(app)` exists, Sphinx will call this function as a normal extension during application setup.

For example, if a custom stylesheet exists at `_static/css/custom.css`, a `conf.py` extension can be written to add this file to the list of stylesheets:

```
def setup(app):
    app.add_stylesheet('css/custom.css')
```

Using an extension to add the stylesheet allows for the file to be added to the list of stylesheets later in the Sphinx setup process, making overriding parts of the Read the Docs theme possible.

The same method can be used to add additional scripts that might override previously initialized scripts:

```
def setup(app):
    app.add_javascript("js/custom.js")
```

Enabling Build Notifications

Read the Docs allows you to configure emails that can be sent on failing builds. This makes sure you know when your builds have failed.

Take these steps to enable build notifications:

- Going to **Admin > Notifications** in your project.
- Fill in the **Email** field under the **New Email Notifications** heading

- Submit the form

You should now get notified when your builds fail!

Enabling Google Analytics on your Project

Read the Docs has native support for Google Analytics. You can enable it by:

- Going to **Admin > Advanced Settings** in your project.
- Fill in the **Analytics code** heading with your Google Tracking ID (example `UA-123456674-1`)

Once your documentation rebuilds it will include your Analytics tracking code and start sending data. Google Analytics usually takes 60 minutes, and sometimes can take up to a day before it starts reporting data.

Removing "Edit on ..." Buttons from Documentation

When building your documentation, Read the Docs automatically adds buttons at the top of your documentation that point readers to your repository to make changes. For instance, if your repository is on GitHub, a button that says "Edit on GitHub" is added to your documentation to make it easy for readers to author new changes.

The only way to remove these links currently is to override the Read the Docs theme templates:

- In your Sphinx project path, create a directory called `_templates`. If you use a different `templates_path` option in your `conf.py`, substitute that directory name.
- Create a file in this path called `breadcrumbs.html`

The new `breadcrumbs.html` should look like this:

```
{%- extends "sphinx_rtd_theme/breadcrumbs.html" %}

{% block breadcrumbs_aside %}
{% endblock %}
```

Now when you build your documentation, your documentation won't include an edit button or links to the page source.

Wiping a Build Environment

Sometimes it happen that your Builds start failing because the build environment where the documentation is created is stale or broken. This could happen for a couple of different reasons like `pip` not upgrading a package properly or a corrupted cached Python package.

In any of these cases (and many others), the solution could be just wiping out the existing build environment files and allow Read the Docs to create a new fresh one.

Follow these steps to wipe the build environment:

- Go to **Versions**
- Click on the **Edit** button of the version you want to wipe on the right side of the page
- Go to the bottom of the page and click the **wipe** link, next to the "Save" button

Note: By wiping the documentation build environment, all the `rst`, `md`, and code files associated with it will be removed but not the documentation already built (`HTML` and `PDF` files). Your documentation will still online after wiping the build environment.

Now you can re-build the version with a fresh build environment!

CHAPTER 9

Read the Docs Public API

We have a limited public API that is available for you to get data out of the site. This document covers only part of the API provided. We have plans to create a read/write API, so that you can easily automate interactions with your project.

Warning: This API is out of date and not currently maintained. We have a v2 API that is currently supported at http://readthedocs.org/api/v2/.

A basic API client using slumber

You can use Slumber to build basic API wrappers in python. Here is a simple example of using slumber to interact with the RTD API:

```
from __future__ import print_function
import slumber
import json

show_objs = True
api = slumber.API(base_url='http://readthedocs.org/api/v1/')

val = api.project.get(slug='pip')

if show_objs:
    for obj in val['objects']:
        print(json.dumps(obj, indent=4))
else:
    print(json.dumps(val, indent=4))
```

Alternatively you can try with the following value:

```
# fetch project pip without metadata.
val = api.project('pip').get()
```

```
# get a specific build
val = api.build(2592228).get()

# get the build of a specific project.
val = api.build.get(project__slug='read-the-docs')

# get a specific user by `username`
val = api.user.get(username='eric')

#val = api.version('pip').get()
#val = api.version('pip').get(slug='1.0.1')
```

API Endpoints

Feel free to use cURL and python to look at formatted json examples. You can also look at them in your browser, if it handles returned json.

```
curl http://readthedocs.org/api/v1/project/pip/?format=json | python -m json.tool
```

Doc Search

GET /api/v2/docsearch/

string project **Required**. The slug of a project.

string version **Required**. The slug of the version for this project.

string q **Required**. The search query

You can search a specific set of documentation using our doc search endpoint. It returns data in the format of Elastic Search, which requires a bit of traversing to use.

In the future we might change the format of this endpoint to make it more abstact.

An example URL: http://readthedocs.org/api/v2/docsearch/?project=docs&version=latest&q=subdomains

Results:

```
{
    "results": {
        "hits": {
            "hits": [
                {
                    "fields": {
                        "link": "http://localhost:9999/docs/test-docs/en/latest/
↪history/classes/coworking",
                        "path": [
                            "history/classes/coworking"
                        ],
                        "project": [
                            "test-docs"
                        ],
                        "title": [
```

```
                            "PIE coworking"
                        ],
                        "version": [
                            "latest"
                        ]
                    },
                    "highlight": {
                        "content": [
                            "\nhelp fund more endeavors. Beta <em>test</em>  This␣
↪first iteration of PIE was a very underground project"
                        ]
                    }
                },
            ],
            "max_score": 0.47553805,
            "total": 2
        }
    }
}
```


Root

GET /api/v1/

Retrieve a list of resources.


```
{
    "build": {
        "list_endpoint": "/api/v1/build/",
        "schema": "/api/v1/build/schema/"
    },
    "file": {
        "list_endpoint": "/api/v1/file/",
        "schema": "/api/v1/file/schema/"
    },
    "project": {
        "list_endpoint": "/api/v1/project/",
        "schema": "/api/v1/project/schema/"
    },
    "user": {
        "list_endpoint": "/api/v1/user/",
        "schema": "/api/v1/user/schema/"
    },
    "version": {
        "list_endpoint": "/api/v1/version/",
        "schema": "/api/v1/version/schema/"
    }
}
```


Response JSON Object

- **list_endpoint** (*string*) – API endpoint for resource.
- **schema** (*string*) – API endpoint for schema of resource.

Builds

GET /api/v1/build/

Retrieve a list of Builds.

```
{
    "meta": {
        "limit": 20,
        "next": "/api/v1/build/?limit=20&offset=20",
        "offset": 0,
        "previous": null,
        "total_count": 86684
    },
    "objects": [BUILDS]
}
```

Response JSON Object

- **limit** (*integer*) – Number of Builds returned.
- **next** (*string*) – URI for next set of Builds.
- **offset** (*integer*) – Current offset used for pagination.
- **previous** (*string*) – URI for previous set of Builds.
- **total_count** (*integer*) – Total number of Builds.
- **objects** (*array*) – Array of *Build* objects.

Build

GET /api/v1/build/{id}/

Parameters

- **id** – A Build id.

Retrieve a single Build.

```
{
    "date": "2012-03-12T19:58:29.307403",
    "error": "SPHINX ERROR",
    "id": "91207",
    "output": "SPHINX OUTPUT",
    "project": "/api/v1/project/2599/",
    "resource_uri": "/api/v1/build/91207/",
    "setup": "HEAD is now at cd00d00 Merge pull request #181 from Nagyman/solr_
↪setup\n",
    "setup_error": "",
    "state": "finished",
    "success": true,
    "type": "html",
    "version": "/api/v1/version/37405/"
}
```

Response JSON Object

- **date** (*string*) – Date of Build.
- **error** (*string*) – Error from Sphinx build process.
- **id** (*string*) – Build id.
- **output** (*string*) – Output from Sphinx build process.
- **project** (*string*) – URI for Project of Build.
- **resource_uri** (*string*) – URI for Build.
- **setup** (*string*) – Setup output from Sphinx build process.
- **setup_error** (*string*) – Setup error from Sphinx build process.
- **state** (*string*) – "triggered", "building", or "finished"
- **success** (*boolean*) – Was build successful?
- **type** (*string*) – Build type ("html", "pdf", "man", or "epub")
- **version** (*string*) – URI for Version of Build.

Files

GET /api/v1/file/

Retrieve a list of Files.

```
{
    "meta": {
        "limit": 20,
        "next": "/api/v1/file/?limit=20&offset=20",
        "offset": 0,
        "previous": null,
        "total_count": 32084
    },
    "objects": [FILES]
}
```

Response JSON Object

- **limit** (*integer*) – Number of Files returned.
- **next** (*string*) – URI for next set of Files.
- **offset** (*integer*) – Current offset used for pagination.
- **previous** (*string*) – URI for previous set of Files.
- **total_count** (*integer*) – Total number of Files.
- **objects** (*array*) – Array of *File* objects.

File

GET /api/v1/file/{id}/

Parameters

- **id** – A File id.

  Retrieve a single File.

```
{
    "absolute_url": "/docs/keystone/en/latest/search.html",
    "id": "332692",
    "name": "search.html",
    "path": "search.html",
    "project": {PROJECT},
    "resource_uri": "/api/v1/file/332692/"
  }
```

Response JSON Object

- **absolute_url** (*string*) – URI for actual file (not the File object from the API.)
- **id** (*string*) – File id.
- **name** (*string*) – Name of File.
- **path** (*string*) – Name of Path.
- **project** (*object*) – A *Project* object for the file's project.
- **resource_uri** (*string*) – URI for File object.

Projects

GET /api/v1/project/

Retrieve a list of Projects.

```
{
    "meta": {
        "limit": 20,
        "next": "/api/v1/project/?limit=20&offset=20",
        "offset": 0,
        "previous": null,
        "total_count": 2067
    },
    "objects": [PROJECTS]
}
```

Response JSON Object

- **limit** (*integer*) – Number of Projects returned.
- **next** (*string*) – URI for next set of Projects.
- **offset** (*integer*) – Current offset used for pagination.
- **previous** (*string*) – URI for previous set of Projects.
- **total_count** (*integer*) – Total number of Projects.
- **objects** (*array*) – Array of *Project* objects.

Project

GET /api/v1/project/{id}

Parameters

- **id** – A Project id.

  Retrieve a single Project.


```
{
    "absolute_url": "/projects/docs/",
    "analytics_code": "",
    "copyright": "",
    "crate_url": "",
    "default_branch": "",
    "default_version": "latest",
    "description": "Make docs.readthedocs.io work :D",
    "django_packages_url": "",
    "documentation_type": "sphinx",
    "id": "2599",
    "modified_date": "2012-03-12T19:59:09.130773",
    "name": "docs",
    "project_url": "",
    "pub_date": "2012-02-19T18:10:56.582780",
    "repo": "git://github.com/rtfd/readthedocs.org",
    "repo_type": "git",
    "requirements_file": "",
    "resource_uri": "/api/v1/project/2599/",
    "slug": "docs",
    "subdomain": "http://docs.readthedocs.io/",
    "suffix": ".rst",
    "theme": "default",
    "use_virtualenv": false,
    "users": [
        "/api/v1/user/1/"
    ],
    "version": ""
}
```


Response JSON Object

- **absolute_url** (*string*) – URI for project (not the Project object from the API.)
- **analytics_code** (*string*) – Analytics tracking code.
- **copyright** (*string*) – Copyright
- **crate_url** (*string*) – Crate.io URI.
- **default_branch** (*string*) – Default branch.
- **default_version** (*string*) – Default version.
- **description** (*string*) – Description of project.
- **django_packages_url** (*string*) – Djangopackages.com URI.
- **documentation_type** (*string*) – Either “sphinx” or “sphinx_html”.
- **id** (*string*) – Project id.

- **modified_date** (*string*) – Last modified date.
- **name** (*string*) – Project name.
- **project_url** (*string*) – Project homepage.
- **pub_date** (*string*) – Last published date.
- **repo** (*string*) – URI for VCS repository.
- **repo_type** (*string*) – Type of VCS repository.
- **requirements_file** (*string*) – Pip requirements file for packages needed for building docs.
- **resource_uri** (*string*) – URI for Project.
- **slug** (*string*) – Slug.
- **subdomain** (*string*) – Subdomain.
- **suffix** (*string*) – File suffix of docfiles. (Usually ".rst".)
- **theme** (*string*) – Sphinx theme.
- **use_virtualenv** (*boolean*) – Build project in a virtualenv? (True or False)
- **users** (*array*) – Array of readthedocs.org user URIs for administrators of Project.
- **version** (*string*) – DEPRECATED.

Users

GET /api/v1/user/

Retrieve List of Users

```
{
    "meta": {
        "limit": 20,
        "next": "/api/v1/user/?limit=20&offset=20",
        "offset": 0,
        "previous": null,
        "total_count": 3200
    },
    "objects": [USERS]
}
```

Response JSON Object

- **limit** (*integer*) – Number of Users returned.
- **next** (*string*) – URI for next set of Users.
- **offset** (*integer*) – Current offset used for pagination.
- **previous** (*string*) – URI for previous set of Users.
- **total_count** (*integer*) – Total number of Users.
- **USERS** (*array*) – Array of *User* objects.

User

GET /api/v1/user/{id}/

Parameters

- **id** – A User id.

Retrieve a single User

```
{
    "first_name": "",
    "id": "1",
    "last_login": "2010-10-28T13:38:13.022687",
    "last_name": "",
    "resource_uri": "/api/v1/user/1/",
    "username": "testuser"
}
```

Response JSON Object

- **first_name** (*string*) – First name.
- **id** (*string*) – User id.
- **last_login** (*string*) – Timestamp of last login.
- **last_name** (*string*) – Last name.
- **resource_uri** (*string*) – URI for this user.
- **username** (*string*) – User name.

Versions

GET /api/v1/version/

Retrieve a list of Versions.

```
{
    "meta": {
        "limit": 20,
        "next": "/api/v1/version/?limit=20&offset=20",
        "offset": 0,
        "previous": null,
        "total_count": 16437
    },
    "objects": [VERSIONS]
}
```

Response JSON Object

- **limit** (*integer*) – Number of Versions returned.
- **next** (*string*) – URI for next set of Versions.
- **offset** (*integer*) – Current offset used for pagination.
- **previous** (*string*) – URI for previous set of Versions.

- **total_count** (*integer*) – Total number of Versions.
- **objects** (*array*) – Array of *Version* objects.

Version

GET /api/v1/version/{id}

Parameters

- **id** – A Version id.

Retrieve a single Version.

```
{
    "active": false,
    "built": false,
    "id": "12095",
    "identifier": "remotes/origin/zip_importing",
    "project": {PROJECT},
    "resource_uri": "/api/v1/version/12095/",
    "slug": "zip_importing",
    "uploaded": false,
    "verbose_name": "zip_importing"
}
```

Response JSON Object

- **active** (*boolean*) – Are we continuing to build docs for this version?
- **built** (*boolean*) – Have docs been built for this version?
- **id** (*string*) – Version id.
- **identifier** (*string*) – Identifier of Version.
- **project** (*object*) – A *Project* object for the version's project.
- **resource_uri** (*string*) – URI for Version object.
- **slug** (*string*) – String that uniquely identifies a project
- **uploaded** (*boolean*) – Were docs uploaded? (As opposed to being build by Read the Docs.)
- **verbose_name** (*string*) – Usually the same as Slug.

Filtering Examples

File Search

```
http://readthedocs.org/api/v1/file/search/?format=json&q=virtualenvwrapper
```

GET /api/v1/file/search/?q={search_term}

Parameters

- **search_term** – Perform search with this term.

  Retrieve a list of File objects that contain the search term.


```
{
    "objects": [
        {
            "absolute_url": "/docs/python-guide/en/latest/scenarios/virtualenvs/
↪index.html",
            "id": "375539",
            "name": "index.html",
            "path": "scenarios/virtualenvs/index.html",
            "project": {
                "absolute_url": "/projects/python-guide/",
                "analytics_code": null,
                "copyright": "Unknown",
                "crate_url": "",
                "default_branch": "",
                "default_version": "latest",
                "description": "[WIP] Python best practices...",
                "django_packages_url": "",
                "documentation_type": "sphinx_htmldir",
                "id": "530",
                "modified_date": "2012-03-13T01:05:30.191496",
                "name": "python-guide",
                "project_url": "",
                "pub_date": "2011-03-20T19:40:03.599987",
                "repo": "git://github.com/kennethreitz/python-guide.git",
                "repo_type": "git",
                "requirements_file": "",
                "resource_uri": "/api/v1/project/530/",
                "slug": "python-guide",
                "subdomain": "http://python-guide.readthedocs.io/",
                "suffix": ".rst",
                "theme": "kr",
                "use_virtualenv": false,
                "users": [
                    "/api/v1/user/130/"
                ],
                "version": ""
            },
            "resource_uri": "/api/v1/file/375539/",
            "text": "...<span class=\"highlighted\">virtualenvwrapper</span>\n..."
        },
        ...
    ]
}
```


Anchor Search

```
http://readthedocs.org/api/v1/file/anchor/?format=json&q=virtualenv
```

GET /api/v1/file/anchor/?q={search_term}

Parameters

- **search_term** – Perform search of files containing anchor text with this term.

Retrieve a list of absolute URIs for files that contain the search term.

```
{
    "objects": [
        "http//django-fab-deploy.readthedocs.io/en/latest/...",
        "http//dimagi-deployment-tools.readthedocs.io/en/...",
        "http//openblock.readthedocs.io/en/latest/install/base_install.html
↪#virtualenv",
        ...
    ]
}
```

CHAPTER 10

Embed API

Read the Docs allow you to embed content from any of the projects we host. This allows for content re-use across sites, making sure the content is always up to date.

Workflow

There are many uses of our Embed API. One of our favorites is for inline help. We have started using this on Read the Docs itself to provide inline help on our main site pages. This allows us to keep the official documentation as the single source of truth, while having great inline help in our application as well.

We recommend you point at **tagged releases** instead of `latest`. Tags don't change over time, so you don't have to worry about the content you are embedding disappearing.

Note: All relative links to pages contained in the remote content will continue to point at the remote page.

How to use it

Sphinx Extension

You can embed content directly in Sphinx with builds on Read the Docs. We support default configuration variables for your `conf.py`:

- readthedocs_embed_project
- readthedocs_embed_version
- readthedocs_embed_doc

These are overridable per-call as well. Then you simply use the directive:

```
# All arguments
.. readthedocs-embed::
    :project: myproject
    :version: latest
    :doc: index
    :section: User Guide

# Or with some defaults
.. readthedocs-embed::
    :doc: index
    :section: User Guide
```

Javascript

We provide a Javascript library that you can embed on any site. An example:

```
<!-- In your <head> -->
<link rel="stylesheet" href="http://localhost:5555/static/css/public.css">
<script type="text/javascript" src="http://localhost:5555/static/javascript/bundle-
↪public.js"></script>
<script>
embed = ReadTheDocs.Embed(project, version)
rtd.into('page', 'section', function(content){
    $('#foobar').html(content)
})

</script>

<!-- In your <body> -->
<div id="help_container">
  <a href="#" class="readthedocs-help-embed" data-section="How we envision versions␣
↪working">(Help)</a>
</div>
```

This will provide a pop-out for a user with the `How we envision versions working` section of the `versions` page. You can see this in action here:

Note: All Read the Docs pages already have the library loaded, so you can ignore the `link` and first `script` tags on all documentation.

Warning: We currently do not provide caching on this API. If the remote source you are including changes their page structure or deletes the content, your embed will break.

In Version 2 of this API we will provide a full-formed workflow that will stop this from happening.

Example API Response

Pure API use will return JSON:

```
{
    "content": [
```

```
        "<div class=\"section\" id=\"encoded-data\">\n<h2>Encoded Data?<a class=\
↪"headerlink\" href=\"/docs/requests/en/latest/community/faq.html#encoded-data\"␣
↪title=\"Permalink to this headline\">\u00b6</a></h2>\n<p>Requests automatically␣
↪decompresses gzip-encoded responses, and does\nits best to decode response content␣
↪to unicode when possible.</p>\n<p>You can get direct access to the raw response␣
↪(and even the socket),\nif needed as well.</p>\n</div>"
    ],
    "wrapped": [
        "\n<div class=\"readthedocs-embed-wrapper\">\n    <div class=\"readthedocs-
↪embed-content\">\n        <div class=\"section\" id=\"encoded-data\">\n<h2>Encoded␣
↪Data?<a class=\"headerlink\" href=\"/docs/requests/en/latest/community/faq.html
↪#encoded-data\" title=\"Permalink to this headline\">\u00b6</a></h2>\n<p>Requests␣
↪automatically decompresses gzip-encoded responses, and does\nits best to decode␣
↪response content to unicode when possible.</p>\n<p>You can get direct access to the␣
↪raw response (and even the socket),\nif needed as well.</p>\n</div>\n    </div>\n  ␣
↪ <div class=\"readthedocs-embed-badge\">\n        Embedded from <a href=\"/docs/
↪requests/en/latest/community/faq.html\">Read the Docs</a>\n    </div>\n</div>\n    "
    ],
    "meta": {
        "project": "requests",
        "doc": "community/faq",
        "section": "Encoded Data?",
        "version": "latest",
        "modified": "Wed, 04 Feb 2015 08:59:59 GMT"
    },
    "url": "/docs/requests/en/latest/community/faq.html",
    "headers": [
        {
            "Frequently Asked Questions": "#"
        },
        {
            "Encoded Data?": "#encoded-data"
        },
        {
            "Custom User-Agents?": "#custom-user-agents"
        },
        {
            "Why not Httplib2?": "#why-not-httplib2"
        },
        {
            "Python 3 Support?": "#python-3-support"
        },
        {
            "What are \u201chostname doesn\u2019t match\u201d errors?": "#what-are-
↪hostname-doesn-t-match-errors"
        }
    ]
}
```

CHAPTER 11

Contributing to Read the Docs

You are here to help on Read the Docs? Awesome, feel welcome and read the following sections in order to know what and how to work on something. If you get stuck at any point you can create a ticket on GitHub.

All members of our community are expected to follow our *Code of Conduct*. Please make sure you are welcoming and friendly in all of our spaces.

Contributing to development

If you want to deep dive and help out with development on Read the Docs, then first get the project installed locally according to the *Installation Guide*. After that is done we suggest you have a look at tickets in our issue tracker that are labelled Good First Issue. These are meant to be a great way to get a smooth start and won't put you in front of the most complex parts of the system.

If you are up to more challenging tasks with a bigger scope, then there are a set of tickets with a Feature Overview tag. These tickets have a general overview and description of the work required to finish. If you want to start somewhere, this would be a good place to start. That said, these aren't necessarily the easiest tickets. They are simply things that are explained. If you still didn't find something to work on, search for the Sprintable label. Those tickets are meant to be standalone and can be worked on ad-hoc.

When contributing code, then please follow the standard Contribution Guidelines set forth at contribution-guide.org.

We have a strict code style that is easy to follow since you just have to install pre-commit and it will automatically run different linting tools (autoflake, autopep8, docformatter, isort, prospector, unify and yapf) to check your changes before you commit them. `pre-commit` will let you know if there were any problems that is wasn't able to fix automatically.

To run the `pre-commit` command and check your changes:

```
$ pip install -U pre-commit
$ git add <your-modified-files>
$ pre-commit run
```

or to run against a specific file:

```
$ pre-commit run --files <file.py>
```

`pre-commit` can also be run as a git pre-commit hook. You can set this up with:

```
$ pre-commit install
```

After this installation, the next time you run `git commit` the `pre-commit run` command will be run immediately and will inform you of the changes and errors.

Note: Our code base is still maturing and the core team doesn't yet recommend running this as a pre-commit hook due to the number of changes this will cause while constructing a pull request. Independent pull requests with linting changes would be a great help to making this possible.

Triaging tickets

Here is a brief explanation on how we triage incoming tickets to get a better sense of what needs to be done on what end.

Note: You will need Triage permission on the project in order to do this. You can ask one of the members of the *Read the Docs Team* to give you access.

Initial triage

When sitting down to do some triaging work, we start with the list of untriaged tickets. We consider all tickets that do not have a label as untriaged. The first step is to categorize the ticket into one of the following categories and either close the ticket or assign an appropriate label. The reported issue ...

... is not valid If you think the ticket is invalid comment why you think it is invalid, then close the ticket. Tickets might be invalid if they were already fixed in the past or it was decided that the proposed feature will not be implemented because it does not conform with the overall goal of Read the Docs. Also if you happen to know that the problem was already reported, label the ticket with **Status: duplicate**, reference the other ticket that is already addressing the problem and close the duplicate.

Examples:

- *Builds fail when using matplotlib*: If the described issue was already fixed, then explain and instruct to re-trigger the build.
- *Provide way to upload arbitrary HTML files*: It was already decided that Read the Docs is not a dull hosting platform for HTML. So explain this and close the ticket.

... does not provide enough information Add the label **Needed: more information** if the reported issue does not contain enough information to decide if it is valid or not and ask on the ticket for the required information to go forward. We will re-triage all tickets that have the label **Needed: more information** assigned. If the original reporter left new information we can try to re-categorize the ticket. If the reporter did not come back to provide more required information after a long enough time, we will close the ticket (this will be roughly about two weeks).

Examples:

- *My builds stopped working. Please help!* Ask for a link to the build log and for which project is affected.

... is a valid enhancement proposal If the ticket contains an enhancement proposal that aligns with the goals of Read the Docs, then add the label **Enhancement**. If the proposal seems valid but requires further discussion between core contributors because there might be different possibilities on how to implement the enhancement, then also add the label **Needed: design decision**.

Examples:

- *Improve documentation about MKdocs integration*
- *Provide better integration with service XYZ*
- *Refactor module X for better readability*
- *Achieve world domination* (also needs the label **Needed: design decision**)

... is a valid problem within the code base: If it's a valid bug, then add the label **Bug**. Try to reference related issues if you come across any.

Examples:

- *Builds fail if conf.py contains non-ascii letters*

... is a currently valid problem with the infrastructure: Users might report about web server downtimes or that builds are not triggered. If the ticket needs investigation on the servers, then add the label **Operations**.

Examples:

- *Builds are not starting*

... is a question and needs answering: If the ticket contains a question about the Read the Docs platform or the code, then add the label **Support**.

Examples:

- *My account was set inactive. Why?*
- *How to use C modules with Sphinx autodoc?*
- *Why are my builds failing?*

... requires a one-time action on the server: Tasks that require a one time action on the server should be assigned the two labels **Support** and **Operations**.

Examples:

- *Please change my username*
- *Please set me as owner of this abandoned project*

After we finished the initial triaging of new tickets, no ticket should be left without a label.

Additional labels for categorization

Additionally to the labels already involved in the section above, we have a few more at hand to further categorize issues.

High Priority If the issue is urgent, assign this label. In the best case also go forward to resolve the ticket yourself as soon as possible.

Community Effort There are many valuable ideas in the issue tracker for future enhancements of Read the Docs. Unfortunately too many for the core developers to handle all of them. Therefore we assign the *Community Effort* label on all the issues that we see as valid for the project but that we currently do not have the resources to work on. We encourage community members to work on these tickets and to submit a pull request.

Good First Issue This label marks tickets that are easy to get started with. The ticket should be ideal for beginners to dive into the code base. Better is if the fix for the issue only involves touching one part of the code.

Sprintable Sprintable are all tickets that have the right amount of scope to be handled during a sprint. They are very focused and encapsulated.

Feature Overview If a feature is too big to be tackled in one ticket and should be split up, then we have a feature overview ticket explaining the overarching idea. Those tickets related to one feature should also be grouped by a milestone.

For a full list of available labels and their meanings, see *Overview of issue labels*.

Helpful links for triaging

Here is a list of links for contributors that look for work:

- Untriaged tickets: Go and triage them!
- Tickets labelled with Needed: more information: Come back to these tickets once in a while and close those that did not get any new information from the reporter. If new information is available, go and re-triage the ticket.
- Tickets labelled with Operations: These tickets are for contributors who have access to the servers.
- Tickets labelled with Support: Experienced contributors or community members with a broad knowledge about the project should handle those.
- Tickets labelled with Needed: design decision: Project leaders must take actions on these tickets. Otherwise no other contributor can go forward on them.

Helping on translations

If you wish to contribute translations, please do so on Transifex.

CHAPTER 12

Read the Docs Team

readthedocs.org is the largest open source documentation hosting service. It's provided as a free service to the open source community, and is worked on by a community of volunteers that we're hoping to expand! We currently serve over 20,000,000 pageviews a month, and we hope to maintain a reliable and stable hosting platform for years to come.

There are three major parts of this work:

Note: You may notice that a number of names appear on multiple teams. This is because we are lacking contributors. So please be bold and contact us, and we'll get you sorted into the right team.

Support Team

Read the Docs has thousands of users who depend on it everyday. Every day at least one of them has an issue that needs to be addressed by a site admin. This might include tasks like:

- Resetting a password
- Asking for a project name to be released
- Troubleshooting build errors

Members

- Eric Holscher (Pacific Time)

- Anthony Johnson (Mountain Time)
- Manuel Kaufmann (Central Time)
- Your Name Here

Feel free to ask any of us if you have questions or want to join!

Joining

The best place to start would be to start addressing some of the issues in our issue tracker. We have our support policies quite well documented in our *Contributing to Read the Docs*. **Be bold**. Start trying to reproduce issues that people have, or talk to them to get more information. After you get the hang of things, we'll happily give you the ability to tag and close issues by joining our Support Team.

Operations Team

readthedocs.org is a service that millions of people depend on each month. As part of operating the site, we maintain a 24/7 on-call rotation. This means that folks have to be available and have their phone in service.

Members

- Eric Holscher (Pacific Time)
- Anthony Johnson (Mountain Time)
- Matt Robenolt (Pacific Time)
- Your Name Here

Feel free to ask any of us if you have questions or want to join!

Joining

We are always looking for more people to share the on-call responsibility. If you are on-call for your job already, we'd love to piggy back on that duty as well.

You can email us at dev@readthedocs.org if you want to join our operations team. Because of the sensitive nature (API tokens, secret keys, SSL certs, etc.) of the work, we keep a private GitHub repository with the operations code & documentation.

The tools that we use are:

- Salt
- Nagios
- Graphite/Grafana
- Nginx
- Postgres
- Django
- Celery

It's fine if you aren't familiar with all of these things, but are willing to help with part of it!

Please reach out if you want to share the on-call responsibility. It really is an important job, and we'd love to have it be more geographically distributed.

Development Team

Also known as the "Core Team" in other projects. These folks have the ability to commit code to the project.

Members

- Eric Holscher
- Anthony Johnson
- Manuel Kaufmann
- David Fischer
- Your name here

Feel free to ask any of us if you have questions or want to join!

Joining

We try to be pretty flexible with who we allow on the development team. The best path is to send a few pull requests, and follow up to make sure they get merged successfully. You can check out our *Contributing to Read the Docs* to get more information, and find issues that need to be addressed. After that, feel free to ask for a commit bit.

CHAPTER 13

Google Summer of Code

Note: Thanks for your interest in Read the Docs! We are working hard to update the ideas list now that we are accepted in GSOC. Please give us a little while to work on things, and check back on this page for updates.

Read the Docs is participating in the Google Summer of Code in 2018. This page will contain all the information for students and anyone else interested in helping.

Skills

Incoming students will need the following skills:

- Intermediate Python & Django programming
- Familiarity with Markdown, reStructuredText, or some other plain text markup language
- Familiarity with git, or some other source control
- Ability to set up your own development environment for Read the Docs
- Basic understanding of web technologies (HTML/CSS/JS)
- An interest in documentation and improving open source documentation tools!

We're happy to help you get up to speed, but the more you are able to demonstrate ability in advance, the more likely we are to choose your application!

Mentors

Currently we have a few folks signed up:

- Eric Holscher
- Manuel Kaufmann

Warning: Please do not reach out directly to anyone about the Summer of Code. It will **not** increase your chances of being accepted!

Getting Started

The *Installation* doc is probably the best place to get going. It will walk you through getting a basic environment for Read the Docs setup.

Then you can look through our *Contributing to Read the Docs* doc for information on how to get started contributing to RTD.

People who has a history of submitting pull requests will be prioritized in our Summer of Code selection process.

Want to get involved?

If you're interested in participating in GSoC as a student, you can apply during the normal process provided by Google. We are currently overwhelmed with interest, so we are not able to respond individually to each person who is interested.

Project Ideas

We have written our some loose ideas for projects to work on here. We are also open to any other ideas that students might have.

Collections of Projects

This project involves building a user interface for groups of projects in Read the Docs (`Collections`). Users would be allowed to create, publish, and search a `Collection` of projects that they care about. We would also allow for automatic creation of `Collections` based on a project's `setup.py` or `requirements.txt`.

Once a user has a `Collection`, we would allow them to do a few sets of actions on them:

- Search across all the projects in the `Collection` with one search dialog
- Download all the project's documentation (PDF, HTMLZip, Epub) for offline viewing
- Build a landing page for the collection that lists out all the projects, and could even have a user-editable description, similar to our project listing page.

There is likely other ideas that could be done with `Collections` over time.

Support for additional build steps for linting, testing, and other useful things

Currently we only build documentation on Read the Docs, but we'd also like to add additional build steps that lets users perform more actions. This would likely take the form of wraping some of the existing Sphinx builders, and giving folks a nice way to use them inside Read the Docs.

It would be great to have wrappers for the following as a start:

- Link Check (http://www.sphinx-doc.org/en/stable/builders.html#sphinx.builders.linkcheck.CheckExternalLinksBuilder)
- Spell Check (https://pypi.python.org/pypi/sphinxcontrib-spelling/)
- Doctest (http://www.sphinx-doc.org/en/stable/ext/doctest.html#module-sphinx.ext.doctest)
- Coverage (http://www.sphinx-doc.org/en/stable/ext/coverage.html#module-sphinx.ext.coverage)

The goal would also be to make it quite easy for users to contribute third party build steps for Read the Docs, so that other useful parts of the Sphinx ecosystem could be tightly integrated with Read the Docs.

Refactor our search code

Currently we're using a homegrown library for Elastic Search. There is a new elasticsearch-dsl library that we should be using. This project will include:

- Improving our search indexing
- Refactoring how we "model" our search data to use elasticsearch-dsl Models
- Add additional search data into our indexes, like the programming languages, type of document (tutorial, api, etc.) and other data for users to filter by
- (Optional) Improve the UX of the search for users in various ways

Finish YAML config

Currently we have a basic *Read the Docs YAML Config* for Read the Docs. It's still considered beta, and there are a number of features that it doesn't support. We need to support everying users can currently do from our website dashboard inside the YAML file, and then plan a smooth transition path from the database UI to the YAML file.

This is a *large* project and will likely require a good deal of understanding of both Python as well as web technologies. We have a starting list of issues put together, but there will be much more work.

Integrated Redirects

Right now it's hard for users to rename files. We support redirects, but don't create them automatically on file rename, and our redirect code is brittle.

We should rebuild how we handle redirects across a number of cases:

- Detecting a file change in git/hg/svn and automatically creating a redirect
- Support redirecting an entire domain to another place
- Support redirecting versions

There will also be a good number of things that spawn from this, including version aliases and other related concepts, if this task doesn't take the whole summer.

API V3

We currently have a "v2" API that isn't well documented and doesn't allow users to write to it. We want to continue using Django REST Framework for this, but rethink how we're presenting our information to our users.

Currently we're showing everything as simple "models", and we want to start exposing "methods" on our data, similar to GitHub.

This is a large project and should only be done by someone who has done some basic API design previously.

Improve Translation Workflow

Currently we have our documentation & website translated on Transifex, but we don't have a management process for it. This means that translations will often sit for months before making it back into the site and being available to users.

This project would include puting together a workflow for translations:

- Communicate with existing translators and see what needs they have
- Help formalize the process that we have around Transifex to make it easier to contribute to
- Improve our tooling so that integrating new translations is easier

Additional Ideas

We have some medium sized projects sketched out in our issue tracker with the tag *Feature Overview*. Looking through these issues is a good place to start. You might also look through our milestones on GitHub, which provide outlines on the larger tasks that we're hoping to accomplish.

Thanks

This page was heavily inspired by Mailman's similar GSOC page. Thanks for the inspiration.

CHAPTER 14

Code of Conduct

Like the technical community as a whole, the Read the Docs team and community is made up of a mixture of professionals and volunteers from all over the world, working on every aspect of the mission - including mentorship, teaching, and connecting people.

Diversity is one of our huge strengths, but it can also lead to communication issues and unhappiness. To that end, we have a few ground rules that we ask people to adhere to. This code applies equally to founders, mentors and those seeking help and guidance.

This isn't an exhaustive list of things that you can't do. Rather, take it in the spirit in which it's intended - a guide to make it easier to enrich all of us and the technical communities in which we participate.

This code of conduct applies to all spaces managed by the Read the Docs project. This includes IRC, the mailing lists, the issue tracker, and any other forums created by the project team which the community uses for communication. In addition, violations of this code outside these spaces may affect a person's ability to participate within them.

If you believe someone is violating the code of conduct, we ask that you report it by emailing dev@readthedocs.org.

- **Be friendly and patient.**
- **Be welcoming.** We strive to be a community that welcomes and supports people of all backgrounds and identities. This includes, but is not limited to members of any race, ethnicity, culture, national origin, colour, immigration status, social and economic class, educational level, sex, sexual orientation, gender identity and expression, age, size, family status, political belief, religion, and mental and physical ability.
- **Be considerate.** Your work will be used by other people, and you in turn will depend on the work of others. Any decision you take will affect users and colleagues, and you should take those consequences into account when making decisions. Remember that we're a world-wide community, so you might not be communicating in someone else's primary language.
- **Be respectful.** Not all of us will agree all the time, but disagreement is no excuse for poor behavior and poor manners. We might all experience some frustration now and then, but we cannot allow that frustration to turn into a personal attack. It's important to remember that a community where people feel uncomfortable or threatened is not a productive one. Members of the Read the Docs community should be respectful when dealing with other members as well as with people outside the Read the Docs community.

- **Be careful in the words that you choose.** We are a community of professionals, and we conduct ourselves professionally. Be kind to others. Do not insult or put down other participants. Harassment and other exclusionary behavior aren't acceptable. This includes, but is not limited to:
  - Violent threats or language directed against another person.
  - Discriminatory jokes and language.
  - Posting sexually explicit or violent material.
  - Posting (or threatening to post) other people's personally identifying information ("doxing").
  - Personal insults, especially those using racist or sexist terms.
  - Unwelcome sexual attention.
  - Advocating for, or encouraging, any of the above behavior.
  - Repeated harassment of others. In general, if someone asks you to stop, then stop.
- **When we disagree, try to understand why.** Disagreements, both social and technical, happen all the time and Read the Docs is no exception. It is important that we resolve disagreements and differing views constructively. Remember that we're different. The strength of Read the Docs comes from its varied community, people from a wide range of backgrounds. Different people have different perspectives on issues. Being unable to understand why someone holds a viewpoint doesn't mean that they're wrong. Don't forget that it is human to err and blaming each other doesn't get us anywhere. Instead, focus on helping to resolve issues and learning from mistakes.

Original text courtesy of the Speak Up! project. This version was adopted from the Django Code of Conduct.

CHAPTER 15

Ethical Ads

Read the Docs is a large, free web service. There is one proven business model to support this kind of site: **Advertising**. We are building the advertising model we want to exist, and we're calling it **Ethical Ads**.

Ethical Ads respect users while providing value to advertisers. We don't track you, sell your data, or anything else. We simply show ads to users, based on the content of the pages you look at. We also give 10% of our ad space to community projects, as our way of saying thanks to the open source community.

We talk a bit below about *our worldview on advertising*, if you want to know more.

Important: Are you a marketer? Learn more about how you can connect with the millions of developers who Read the Docs each month.

Feedback

We're a community, and we value your feedback. If you ever want to reach out about this effort, feel free to shoot us an email.

You can *opt out* of having paid ads on your projects, or seeing paid ads if you want. You will still see community ads, which we run for free that promote community projects.

We have gone into more detail about our views in our blog post about this topic. Eric Holscher, one of our co-founders talks a bit more about funding open source this way on his blog.

Our worldview

We're building the advertising model we want to exist:

- We don't track you
- We don't sell your data

- We host everything ourselves, no third-party scripts or images

We're doing newspaper advertising, on the internet. For a hundred years, newspapers put an ad on the page, some folks would see it, and advertisers would pay for this. This is our model.

So much ad tech has been built to track users. Following them across the web, from site to site, showing the same ads and gathering data about them. Then retailers sell your purchase data to try and attribute sales to advertising. Now there is an industry in doing fake ad clicks and other scams, which leads the ad industry to track you even more intrusively to know more about you. The current advertising industry is in a vicious downward spiral.

As developers, we understand the massive downsides of the current advertising industry. This includes malware, slow site performance, and huge databases of your personal data being sold to the highest bidder.

The trend in advertising is to have larger and larger ads. They should run before your content, they should take over the page, the bigger, weirder, or flashier the better.

We opt out.

- We don't store personal information about you.
- We only keep track of views and clicks.
- We don't build a profile of your personality to sell ads against.
- We only show high quality ads from companies that are of interest to developers.

We are running a single, small, unobtrusive ad on documentation pages. The products should be interesting to you. The ads won't flash or move.

We run the ads we want to have on our site, in a way that makes us feel good.

Join us

We're building the advertising model we want to exist. We hope that others will join us in this mission:

- **If you're a developer**, talk to your marketing folks about using advertising that respects your privacy.
- **If you're a marketer**, vote with your dollars and support us in building the ad model we want to exist. Get more information on what we offer.

Community Ads

There are a large number of projects that we care about in the software ecosystem. A large number of them operate like we have for the past 6 years, with almost no income. Our new Community Ads program will highlight some of these projects each month.

We'll show 10% of our ad inventory each month to support an open source project that we care about. Send us an email to be considered for our Community Ads program.

Opting Out

We have added multiple ways to opt out of the advertising on Read the Docs.

Users can opt out of seeing paid advertisements on documentation pages:

- Go to the drop down user menu in the top right of the Read the Docs dashboard and clicking **Settings** (https://readthedocs.org/accounts/edit/).
- On the **Details** tab, you can deselect **See paid advertising**.

Project owners can also disable advertisements for their projects. You can change these options:

- Click on your **Project** page (`/projects/<slug>/`)
- Click the **Admin** dashboard link
- Choose the **Advertising** submenu on the left side
- Change your advertising settings

Project opt out options include:

- Supporting us financially with Read the Docs Gold. This will disable all ads from showing on your project's documentation.
- Supporting us with your time by contributing to the project.
- Moving to our paid product over at readthedocs.com.
- Opting out without doing any of the above. This will make us a little sad, but we understand not everyone has the means to contribute back.

CHAPTER 16

Sponsors of Read the Docs

Running Read the Docs isn't free, and the site wouldn't be where it is today without generous support of our sponsors. Below is a list of all the folks who have helped the site financially, in order of the date they first started supporting us.

Current sponsors

- Rackspace - They cover all of our hosting expenses every month. This is a pretty large sum of money, averaging around $3,000/mo, and we are really grateful to have them as a sponsor.
- Mozilla - Mozilla has given us a MOSS grant for building specific features, and have funded Eric Holscher to work on Read the Docs at $1,000/mo for 2016.
- You? (Email us at rev@readthedocs.org for more info)

Past sponsors

- Python Software Foundation
- Revsys
- Mozilla Web Dev
- Django Software Foundation
- Lab305
- Twilio

Sponsorship Information

As part of increasing sustainability, Read the Docs is testing out promoting sponsors on documentation pages. We have more information about this in our blog post about this effort.

Sponsor Us

Contact us at rev@readthedocs.org for more information on sponsoring Read the Docs.

CHAPTER 17

Read the Docs Open Source Philosophy

Read the Docs is Open Source software. We have licensed the code base as MIT, which provides almost no restrictions on the use of the code.

However, as a project there are things that we care about more than others. We built Read the Docs to support documentation in the Open Source community. The code is open for people to contribute to, so that they may build features into https://readthedocs.org that they want. We also believe sharing the code openly is a valuable learning tool, especially for demonsrating how to collaborate and maintain an enormous website.

Official Support

The time of the core developers of Read the Docs is limited. We provide official support for the following things:

- Local development on the Python code base
- Usage of https://readthedocs.org for Open Source projects
- Bug fixes in the code base, as it applies to running it on https://readthedocs.org

Unsupported

There are use cases that we don't support, because it doesn't further our goal of promoting documentation in the Open Source Community.

We do not support:

- Specific usage of Sphinx and Mkdocs, that don't affect our hosting
- Custom installations of Read the Docs at your company
- Installation of Read the Docs on other platforms
- Any installation issues outside of the Read the Docs Python Code

Rationale

Read the Docs was founded to improve documentation in the Open Source Community. We fully recognize and allow the code to be used for internal installs at companies, but we will not spend our time supporting it. Our time is limited, and we want to spend it on the mission that we set out to originally support.

If you feel strongly about installing Read the Docs internal to a company, we will happily link to third party resources on this topic. Please open an issue with a proposal if you want to take on this task.

CHAPTER 18

The Story of Read the Docs

Documenting projects is hard, hosting them shouldn't be. Read the Docs was created to make hosting documentation simple.

Read the Docs was started with a couple main goals in mind. The first goal was to encourage people to write documentation, by removing the barrier of entry to hosting. The other goal was to create a central platform for people to find documentation. Having a shared platform for all documentation allows for innovation at the platform level, allowing work to be done once and benefit everyone.

Documentation matters, but its often overlooked. We think that we can help a documentation culture flourish. Great projects, such as Django and SQLAlchemy, and projects from companies like Mozilla, are already using Read the Docs to serve their documentation to the world.

The site has grown quite a bit over the past year. Our look back at 2013 shows some numbers that show our progress. The job isn't anywhere near done yet, but it's a great honor to be able to have such an impact already.

We plan to keep building a great experience for people hosting their docs with us, and for users of the documentation that we host.

CHAPTER 19

Policy for Abandoned Projects

This policy describes the process by which a Read the Docs project name may be changed.

Rationale

Conflict between the current use of the name and a different suggested use of the same name occasionally arise. This document aims to provide general guidelines for solving the most typical cases of such conflicts.

Specification

The main idea behind this policy is that Read the Docs serves the community. Every user is invited to upload content under the Terms of Use, understanding that it is at the sole risk of the user.

While Read the Docs is not a backup service, the core team of Read the Docs does their best to keep that content accessible indefinitely in its published form. However, in certain edge cases the greater community's needs might outweigh the individual's expectation of ownership of a project name.

The use cases covered by this policy are:

Abandoned projects Renaming a project so that the original project name can be used by a different project

Active projects Resolving disputes over a name

Implementation

Reachability

The user of Read the Docs is solely responsible for being reachable by the core team for matters concerning projects that the user owns. In every case where contacting the user is necessary, the core team will try to do so at least three times, using the following means of contact:

- E-mail address on file in the user's profile
- E-mail addresses found in the given project's documentation
- E-mail address on the project's home page

The core team will stop trying to reach the user after six weeks and the user will be considered *unreachable*.

Abandoned projects

A project is considered *abandoned* when ALL of the following are met:

- Owner is unreachable (see *Reachability*)
- The project has no proper documentation being served (no successful builds) or does not have any releases within the past twelve months
- No activity from the owner on the project's home page (or no home page found).

All other projects are considered *active*.

Renaming of an abandoned project

Projects are never renamed solely on the basis of abandonment.

An *abandoned* project can be renamed (by appending `-abandoned` and a uniquifying integer if needed) for purposes of reusing the name when ALL of the following are met:

- The project has been determined *abandoned* by the rules described above
- The candidate is able to demonstrate their own failed attempts to contact the existing owner
- The candidate is able to demonstrate that the project suggested to reuse the name already exists and meets notability requirements
- The candidate is able to demonstrate why a fork under a different name is not an acceptable workaround
- The project has fewer than 100 monthly pageviews
- The core team does not have any additional reservations.

Name conflict resolution for active projects

The core team of Read the Docs are not arbiters in disputes around *active* projects. The core team recommends users to get in touch with each other and solve the issue by respectful communication.

Prior art

The Python Package Index (PyPI) policy for claiming abandoned packages (PEP-0541) heavily influenced this policy.

CHAPTER 20

DMCA Takedown Policy

These are the guidelines that Read the Docs follows when handling DMCA takedown requests and takedown counter requests. If you are a copyright holder wishing to submit a takedown request, or an author that has been notified of a takedown request, please familiarize yourself with *our process*. You will be asked to confirm that you have reviewed information if you submit a request or counter request.

We aim to keep this entire process as transparent as possible. Our process is modeled after GitHub's DMCA takedown process, which we appreciate for its focus on transparency and fairness. All requests and counter requests will be posted to this page below, in the *Request Archive*. These requests will be redacted to remove all identifying information, except for Read the Docs user and project names.

Takedown Process

Here are the steps the Read the Docs will follow in the takedown request process:

Copyright holder submits a request This request, if valid, will be posted publicly on this page, *down below*. The author affected by the takedown request will be notified with a link to the takedown request.

For more information on submitting a takedown request, see: *Submitting a Request*

Author is contacted The author of the content in question will be asked to make changes to the content specified in the takedown request. The author will have 24 hours to make these changes. The copyright holder will be notified if and when this process begins

Author acknowledges changes have been made The author must notify Read the Docs that changes have been made within 24 hours of receiving a takedown request. If the author does not respond to this request, the default action will be to disable the Read the Docs project and remove any hosted versions

Copyright holder review If the author has made changes, the copyright holder will be notified of these changes. If the changes are sufficient, no further action is required, though copyright holders are welcome to submit a formal retraction. If the changes are not sufficient, the author's changes can be rejected. If the takedown request requires alteration, a new request must be submitted. If Read the Docs does not receive a review response from the copyright holder within 2 weeks, the default action at this step is to assume the takedown request has been retracted.

Content may be disabled If the author does not respond to a request for changes, or if the copyright holder has rejected the author's changes during the review process, the documentation project in question will be disabled.

Author submits a counter request If the author believes their content was disabled as a result of a mistake, a counter request may be submitted. It would be advised that authors seek legal council before continuing. If the submitted counter request is sufficiently detailed, this counter will also be added to *this page*. The copyright holder will be notified, with a link to this counter request.

For more information on submitting a counter request, see: *Submitting a Counter*

Copyright holder may file legal action At this point, if the copyright holder wishes to keep the offending content disabled, the copyright holder must file for legal action ordering the author refrain from infringing activities on Read the Docs. The copyright holder will have 2 weeks to supply Read the Docs with a copy of a valid legal complaint against the author. The default action here, if the copyright holder does not respond to this request, is to re-enable the author's project.

Submitting a Request

Your request must:

Acknowledge this process You must first acknowledge you are familiar with our DMCA takedown request process. If you do not acknowledge that you are familiar with our process, you will be instructed to review this information.

Identify the infringing content You should list URLs to each piece of infringing content. If you allege that the entire project is infringing on copyrights you hold, please specify the entire project as infringing.

Identify infringement resolution You will need to specify what a user must do in order to avoid having the rest of their content disabled. Be as specific as possible with this. Specify if this means adding attribution, identify specific files or content that should be removed, or if you allege the entire project is infringing, your should be specific as to why it is infringing.

Include your contact information Include your name, email, physical address, and phone number.

Include your signature This can be a physical or electronic signature.

Requests can be submitted to: support@readthedocs.com

Submitting a Counter

Your counter request must:

Acknowledge this process You must first acknowledge you are familiar with our DMCA takedown request process. If you do not acknowledge that you are familiar with our process, you will be instructed to review this information.

Identify the infringing content that was removed Specify URLs in the original takedown request that you wish to challenge.

Include your contact information Include your name, email, physical address, and phone number.

Include your signature This can be a physical or electronic signature.

Requests can be submitted to: support@readthedocs.com

Request Archive

Currently, Read the Docs has not received any takedown requests.

CHAPTER 21

Webhooks

The primary method that Read the Docs uses to detect changes to your documentation is through the use of *webhooks*. Webhooks are configured with your repository provider, such as GitHub, Bitbucket or GitLab, and with each commit, merge, or other change to your repository, Read the Docs is notified. When we receive a webhook notification, we determine if the change is related to an active version for your project, and if it is, a build is triggered for that version.

Webhook Integrations

You'll find a list of configured webhook integrations on your project's admin dashboard, under **Integrations**. You can select any of these integrations to see the *integration detail page*. This page has additional configuration details and a list of HTTP exchanges that have taken place for the integration.

Webhook creation

If you import a project using a *connected account*, a webhook will be set up automatically for your repository. However, if your project was not imported through a connected account, you may need to manually configure a webhook for your project.

To manually set up a webhook, click **Add integration** on your project's **Integrations** admin dashboard page and select the integration type you'd like to add. After you have added the integration, you'll see a URL for the integration on the *integration detail page*. Use this URL when setting up a new webhook with your provider – these steps vary depending on the provider:

GitHub

- Go to the **Settings** page for your project
- Click **Webhooks** and then **Add webhook**
- For **Payload URL**, use the URL of the integration on Read the Docs, found on the *integration detail page* page

- For **Content type**, both *application/json* and *application/x-www-form-urlencoded* work
- Select **Just the push event**
- Finish by clicking **Add webhook**

Note: The webhook secret is not yet respected

Bitbucket

- Go to the **Settings** page for your project
- Click **Webhooks** and then **Add webhook**
- For **URL**, use the URL of the integration on Read the Docs, found on the *integration detail page* page
- Under **Triggers**, **Repository push** should be selected
- Finish by clicking **Save**

GitLab

- Go to the **Settings** page for your project
- Click **Integrations**
- For **URL**, use the URL of the integration on Read the Docs, found on the *integration detail page* page
- Leave the default **Push events** selected and mark **Tag push events** also
- Finish by clicking **Add Webhook**

Using the generic API integration

For repositories that are not hosted with a supported provider, we also offer a generic API endpoint for triggering project builds. Similar to webhook integrations, this integration has a specific URL, found on the *integration detail page*.

Token authentication is required to use the generic endpoint, you will find this token on the integration details page. The token should be passed in as a request parameter, either as form data or as part of JSON data input.

Parameters

This endpoint accepts the following arguments during an HTTP POST:

branches The names of the branches to trigger builds for. This can either be an array of branch name strings, or just a single branch name string.

Default: **latest**

token The integration token. You'll find this value on the *integration detail page* page.

For example, the cURL command to build the `dev` branch, using the token `1234`, would be:

```
curl -X POST -d "branches=dev" -d "token=1234" https://readthedocs.org/api/v2/webhook/
↪example-project/1/
```

A command like the one above could be called from a cron job or from a hook inside Git, Subversion, Mercurial, or Bazaar.

Authentication

This endpoint requires authentication. If authenticating with an integration token, a check will determine if the token is valid and matches the given project. If instead an authenticated user is used to make this request, a check will be performed to ensure the authenticated user is an owner of the project.

Debugging webhooks

If you are experiencing problems with an existing webhook, you may be able to use the integration detail page to help debug the issue. Each project integration, such as a webhook or the generic API endpoint, stores the HTTP exchange that takes place between Read the Docs and the external source. You'll find a list of these exchanges in any of the integration detail pages.

Resyncing webhooks

It might be necessary to re-establish a webhook if you are noticing problems. To resync a webhook from Read the Docs, visit the integration detail page and follow the directions for re-syncing your repository webhook.

CHAPTER 22

Badges

Badges let you show the state of your documentation to your users. They are great for embedding in your README, or putting inside your actual doc pages.

Status Badges

They will display in green for passing, red for failing, and yellow for unknown states.

Here are a few examples:

You can see it in action in the Read the Docs README. They will link back to your project's documentation page on Read the Docs.

Project Pages

You will now see badges embedded in your project page. The default badge will be pointed at the *default version* you have specified for your project. The badge URLs look like this:

```
https://readthedocs.org/projects/pip/badge/?version=latest
```

You can replace the version argument with any version that you want to show a badge for. If you click on the badge icon, you will be given snippets for RST, Markdown, and HTML; to make embedding it easier.

If you leave the version argument off, it will default to your latest version. This is probably best to include in your README, since it will stay up to date with your Read the Docs project:

```
https://readthedocs.org/projects/pip/badge/
```

CHAPTER 23

Alternate Domains

Read the Docs supports a number of custom domains for your convenience. Shorter URLs make everyone happy, and we like making people happy!

Subdomain Support

Every project has a subdomain that is available to serve its documentation. If you go to <slug>.readthedocs.io, it should show you the latest version of documentation. A good example is http://pip.readthedocs.io

Note: If you have an old project that has an underscore (_) in the name, it will use a subdomain with a hyphen (-). RFC 1035 has more information on valid subdomains.

CNAME Support

If you have your own domain, you can still host with us. This requires two steps:

- Add a CNAME record in your DNS that point to our servers `readthedocs.io`
- Add a Domain object in the **Project Admin > Domains** page for your project.

Note: The `Domain` that should be used is the actual subdomain that you want your docs served on. Generally it will be `docs.projectname.org`.

Using pip as an example, http://www.pip-installer.org resolves, but is hosted on our infrastructure.

As another example, fabric's dig record looks like this:

```
-> dig docs.fabfile.org
...
;; ANSWER SECTION:
docs.fabfile.org.   7200    IN  CNAME   readthedocs.io.
```

Note: We used to map your projects documentation from the subdomain that you pointed your CNAME to. This wasn't workable at scale, and now we require you to set the domain you want to resolve on your project.

CNAME SSL

We don't support SSL for CNAMEs on our side, but you can enable support if you have your own server. SSL requires having a secret key, and if we hosted the key for you, it would no longer be secret.

To enable SSL:

- Have a server listening on 443 that you control
- Add a domain that you wish to point at Read the Docs
- Enable proxying to us, with a custom `X-RTD-SLUG` header

An example nginx configuration for pip would look like:

```
server {
    server_name docs.pip-installer.org;
    location / {
        proxy_pass https://pip.readthedocs.io:443;
        proxy_set_header Host $http_host;
        proxy_set_header X-Forwarded-Proto https;
        proxy_set_header X-Real-IP $remote_addr;
        proxy_set_header X-Scheme $scheme;
        proxy_set_header X-RTD-SLUG pip;
        proxy_connect_timeout 10s;
        proxy_read_timeout 20s;
    }
}
```

rtfd.org

You can also use `rtfd.io` and `rtfd.org` for short URLs for Read the Docs. For example, http://pip.rtfd.io redirects to its documentation page. Any use of `rtfd.io` or `rtfd.org` will simply be redirected to `readthedocs.io`.

CHAPTER 24

Localization of Documentation

Note: This feature only applies to Sphinx documentation. We are working to bring it to our other documentation backends.

Read the Docs supports hosting your docs in multiple languages. There are two different things that we support:

- A single project written in another language
- A project with translations into multiple languages

Single project in another language

It is easy to set the *Language* of your project. On the project *Admin* page (or *Import* page), simply select your desired *Language* from the dropdown. This will tell Read the Docs that your project is in the language. The language will be represented in the URL for your project.

For example, a project that is in Spanish will have a default URL of `/es/latest/` instead of `/en/latest/`.

Note: You must commit the `.po` files for Read the Docs to translate your documentation.

Project with multiple translations

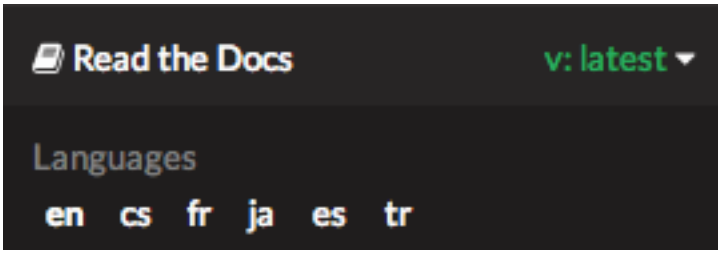
This situation is a bit more complicated. To support this, you will have one parent project and a number of projects marked as translations of that parent. Let's use `phpmyadmin` as an example.

The main `phpmyadmin` project is the parent for all translations. Then you must create a project for each translation, for example `phpmyadmin-spanish`. You will set the *Language* for `phpmyadmin-spanish` to `Spanish`. In the parent projects *Translations* page, you will say that `phpmyadmin-spanish` is a translation for your project.

This has the results of serving:

- `phpmyadmin` at `http://phpmyadmin.readthedocs.io/en/latest/`
- `phpmyadmin-spanish` at `http://phpmyadmin.readthedocs.io/es/latest/`

It also gets included in the Read the Docs flyout:



Note: The default language of any CNAME will be the language of the project the Domain object was set on. See *Alternate Domains* for more information.

Note: You can include multiple translations in the same repository, but each project must specify the language to build for those docs.

CHAPTER 25

Version Control System Integration

If you want to integrate editing into your own theme, you will have to declare few variables inside your configuration file `conf.py` in the `'html_context'` setting, for the template to use them.

More information can be found on Sphinx documentation.

GitHub

If you want to integrate GitHub, you can put the following snippet into your `conf.py`:

```
html_context = {
    "display_github": True, # Integrate GitHub
    "github_user": "MyUserName", # Username
    "github_repo": "MyDoc", # Repo name
    "github_version": "master", # Version
    "conf_py_path": "/source/", # Path in the checkout to the docs root
}
```

It can be used like this:

```
{% if display_github %}
    <li><a href="https://github.com/{{ github_user }}/{{ github_repo }}
    /blob/{{ github_version }}{{ conf_py_path }}{{ pagename }}.rst">
    Show on GitHub</a></li>
{% endif %}
```

Bitbucket

If you want to integrate Bitbucket, you can put the following snippet into your `conf.py`:

```
html_context = {
    "display_bitbucket": True, # Integrate Bitbucket
    "bitbucket_user": "MyUserName", # Username
    "bitbucket_repo": "MyDoc", # Repo name
    "bitbucket_version": "master", # Version
    "conf_py_path": "/source/", # Path in the checkout to the docs root
}
```

It can be used like this:

```
{% if display_bitbucket %}
    <a href="https://bitbucket.org/{{ bitbucket_user }}/{{ bitbucket_repo }}
    /src/{{ bitbucket_version}}{{ conf_py_path }}{{ pagename }}.rst'"
    class="icon icon-bitbucket"> Edit on Bitbucket</a>
{% endif %}
```

Gitlab

If you want to integrate Gitlab, you can put the following snippet into your `conf.py`:

```
html_context = {
    "display_gitlab": True, # Integrate Gitlab
    "gitlab_user": "MyUserName", # Username
    "gitlab_repo": "MyDoc", # Repo name
    "gitlab_version": "master", # Version
    "conf_py_path": "/source/", # Path in the checkout to the docs root
}
```

It can be used like this:

```
{% if display_gitlab %}
    <a href="https://{{ gitlab_host|default("gitlab.com") }}/
    {{ gitlab_user }}/{{ gitlab_repo }}/blob/{{ gitlab_version }}
    {{ conf_py_path }}{{ pagename }}{{ suffix }}" class="fa fa-gitlab">
    Edit on GitLab</a>
{% endif %}
```

Additional variables

- `'pagename'` - Sphinx variable representing the name of the page you're on.

CHAPTER 26

Subprojects

Projects can be configured in a nested manner, by configuring a project as a *subproject* of another project. This allows for documentation projects to share a search index and a namespace or custom domain, but still be maintained independently.

For example, a parent project, `Foo` is set up with a subproject, `Bar`. The documentation for `Foo` will be available at:

https://foo.readthedocs.io/en/latest/

The documentation for `Bar` will be available under this same path:

https://foo.readthedocs.io/projects/bar/en/latest/

Adding a Subproject

In the admin dashboard for your project, select "Subprojects" from the menu. From this page you can add a subproject by typing in the project slug.

Sharing a Custom Domain

Projects and subprojects can also be used to share a custom domain with a number of projects. To configure this, one project should be established as the parent project. This project will be configured with a custom domain. Projects can then be added as subprojects to this parent project.

If the example project `Foo` was set up with a custom domain, `docs.example.com`, the URLs for projects `Foo` and `Bar` would respectively be at: http://docs.example.com/en/latest/ and http://docs.example.com/projects/bar/en/latest/

Search

Projects that are configured as subprojects will share a search index with their parent and sibling projects. This is currently the only way to share search indexes between projects, we do not yet support sharing search indexes between

arbitrary projects.

CHAPTER 27

Conda Support

Warning: This feature is in a beta state. Please file an issue if you find anything wrong.

Read the Docs supports Conda as an environment management tool, along with Virtualenv. Conda support is useful for people who depend on C libraries, and need them installed when building their documentation.

This work was funded by Clinical Graphics – many thanks for their support of Open Source.

Activating Conda

Conda Support is the first feature enabled with *Read the Docs YAML Config*. You can enable it by creating a `readthedocs.yml` file in the root of your repository with the contents:

```
conda:
    file: environment.yml
```

This Conda environment will also have Sphinx and other build time dependencies installed. It will use the same order of operations that we support currently:

- Environment Creation (`conda env create`)
- Dependency Installation (Sphinx)

Custom Installs

If you are running a custom installation of Read the Docs, you will need the `conda` executable installed somewhere on your `PATH`. Because of the way `conda` works, we can't safely install it as a normal dependency into the normal Python virtualenv.

Warning: Installing conda into a virtualenv will override the `activate` script, making it so you can't properly activate that virtualenv anymore.

CHAPTER 28

Canonical URLs

Canonical URLs allow people to have consistent page URLs for domains. This is mainly useful for search engines, so that they can send people to the correct page.

Read the Docs uses these in two ways:

- We point all versions of your docs at the "latest" version as canonical
- We point at the user specified canonical URL, generally a custom domain for your docs.

Example

Fabric hosts their docs on Read the Docs. They mostly use their own domain for them `http://docs.fabfile.org`. This means that Google will index both `http://fabric-docs.readthedocs.io` and `http://docs.fabfile.org` for their documentation.

Fabric will want to set `http://docs.fabfile.org` as their canonical URL. This means that when Google indexes `http://fabric-docs.readthedocs.io`, it will know that it should really point at `http://docs.fabfile.org`.

Enabling

You can set the canonical URL for your project in the Project Admin page. Check your `Domains` tab for the domains that we know about.

Implementation

If you look at the source code for documentation built after you set your canonical URL, you should see a bit of HTML like this:

```
<link rel="canonical" href="http://pip.readthedocs.io/en/latest/installing.html">
```

Links

This is a good explanation of the usage of canonical URLs in search engines:

http://www.mattcutts.com/blog/seo-advice-url-canonicalization/

This is a good explanation for why canonical pages are good for SEO:

http://moz.com/blog/canonical-url-tag-the-most-important-advancement-in-seo-practices-since-sitemaps

CHAPTER 29

Single Version Documentation

Single Version Documentation lets you serve your docs at a root domain. By default, all documentation served by Read the Docs has a root of `/<language>/<version>/`. But, if you enable the "Single Version" option for a project, its documentation will instead be served at `/`.

Warning: This means you can't have translations or multiple versions for your documentation.

You can see a live example of this at http://www.contribution-guide.org

Enabling

You can toggle the "Single Version" option on or off for your project in the Project Admin page. Check your dashboard for a list of your projects.

Effects

Links generated on Read the Docs will now point to the proper URL. For example, if pip was set as a "Single Version" project, then links to its documentation would point to `http://pip.readthedocs.io/` rather than the default `http://pip.readthedocs.io/en/latest/`.

Documentation at `/<language>/<default_version>/` will still be served for backwards compatibility reasons. However, our usage of *Canonical URLs* should stop these from being indexed by Google.

CHAPTER 30

Privacy Levels

Read the Docs supports 3 different privacy levels on 2 different objects; Public, Protected, Private on Projects and Versions.

Understanding the Privacy Levels

Level	Detail	Listing	Search	Viewing
Private	No	No	No	Yes
Protected	Yes	No	No	Yes
Public	Yes	Yes	Yes	Yes

Note: With a URL to view the actual documentation, even private docs are viewable. This is because our architecture doesn't do any logic on documentation display, to increase availability.

Public

This is the easiest and most obvious. It is also the default. It means that everything is available to be seen by everyone.

Protected

Protected means that your object won't show up in Listing Pages, but Detail pages still work. For example, a Project that is Protected will not show on the homepage Recently Updated list, however, if you link directly to the project, you will get a 200 and the page will display.

Protected Versions are similar, they won't show up in your version listings, but will be available once linked to.

Private

Private objects are available only to people who have permissions to see them. They will not display on any list view, and will 404 when you link them to others.

CHAPTER 31

User-defined Redirects

You can set up redirects for a project in your project dashboard's Redirects page.

Quick Summary

- Log into your Readthedocs.com Admin account.
- From your dashboard, select the project on which you wish to add redirects.
- From the project's top navigation bar, select the Admin tab.
- From the left navigation menu, select Redirects.
- In the form box "Redirect Type" select the type of redirect you want. See below for detail.
- Depending on the redirect type you select, enter FROM and/or TO URL as needed.
- When finished, click the SUBMIT Button.

Your redirects will be effective immediately.

Redirect Types

Prefix Redirects

The most useful and requested feature of redirects was when migrating to Read the Docs from an old host. You would have your docs served at a previous URL, but that URL would break once you moved them. Read the Docs includes a language and version slug in your documentation, but not all documentation is hosted this way.

Say that you previously had your docs hosted at `http://docs.example.com/dev/`, you move `docs.example.com` to point at Read the Docs. So users will have a bookmark saved to a page at `http://docs.example.com/dev/install.html`.

You can now set a *Prefix Redirect* that will redirect all 404's with a prefix to a new place. The example configuration would be:

```
Type: Prefix Redirect
From URL: /dev/
```

Your users query would now redirect in the following manner:

```
docs.example.com/dev/install.html ->
docs.example.com/en/latest/install.html
```

Where `en` and `latest` are the default language and version values for your project.

Page Redirects

A more specific case is when you move a page around in your docs. The old page will start 404'ing, and your users will be confused. *Page Redirects* let you redirect a specific page.

Say you move the `example.html` page into a subdirectory of examples: `examples/intro.html`. You would set the following configuration:

```
Type: Page Redirect
From URL: /example.html
To URL: /examples/intro.html
```

Note that the `/` at the start doesn't count the `/en/latest`, but just the user-controlled section of the URL.

Exact Redirects

If you're redirecting from an old host AND you aren't maintaining old paths for your documents, a Prefix Redirect won't suffice and you'll need to create *Exact Redirects* to redirect from a specific URL, to a specific page.

Say you're moving `docs.example.com` to Read the Docs and want to redirect traffic from an old page at `http://docs.example.com/dev/install.html` to a new URL of `http://docs.example.com/en/latest/installing-your-site.html`.

The example configuration would be:

```
Type: Exact Redirect
From URL: /dev/install.html
To URL:   /en/latest/installing-your-site.html
```

Your users query would now redirect in the following manner:

```
docs.example.com/dev/install.html ->
docs.example.com/en/latest/installing-your-site.html
```

Note that you should insert the desired language for "en" and version for "latest" to achieve the desired redirect.

Sphinx Redirects

We also support redirects for changing the type of documentation Sphinx is building. If you switch between *HTMLDir* and *HTML*, your URL's will change. A page at `/en/latest/install.html` will be served at `/en/latest/install/`, or vice versa. The built in redirects for this will handle redirecting users appropriately.

Implementation

Since we serve documentation in a highly available way, we do not run any logic when we're serving documentation. This means that redirects will only happen in the case of a *404 File Not Found*.

In the future we might implement redirect logic in Javascript, but this first version is only implemented in the 404 handlers.

CHAPTER 32

Automatic Redirects

Read the Docs supports redirecting certain URLs automatically. This is an overview of the set of redirects that are fully supported and will work into the future.

Root URL

A link to the root of your documentation will redirect to the *default version*, as set in your project settings. For example:

```
pip.readthedocs.io -> pip.readthedocs.io/en/latest/
www.pip-installer.org -> www.pip-installer.org/en/latest
```

This only works for the root URL, not for internal pages. It's designed to redirect people from http://pip.readthedocs.io/ to the default version of your documentation, since serving up a 404 here would be a pretty terrible user experience. (If your "develop" branch was designated as your default version, then it would redirect to http://pip.readthedocs.io/en/develop.) But, it's not a universal redirecting solution. So, for example, a link to an internal page like http://pip.readthedocs.io/usage.html doesn't redirect to http://pip.readthedocs.io/en/latest/usage.html.

The reasoning behind this is that RTD organizes the URLs for docs so that multiple translations and multiple versions of your docs can be organized logically and consistently for all projects that RTD hosts. For the way that RTD views docs, http://pip.readthedocs.io/en/latest/ is the root directory for your default documentation in English, not http://pip.readthedocs.io/. Just like http://pip.readthedocs.io/en/develop/ is the root for your development documentation in English.

Among all the multiple versions of docs, you can choose which is the "default" version for RTD to display, which usually corresponds to the git branch of the most recent official release from your project.

rtfd.org

Links to rtfd.org are treated the same way as above. They redirect the root URL to the default version of the project. They are intended to be easy and short for people to type.

Supported Top-Level Redirects

Note: These "implicit" redirects are supported for legacy reasons. We will not be adding support for any more magic redirects. If you want additional redirects, they should live at a prefix like *Redirecting to a Page*

The main challenge of URL routing in Read the Docs is handling redirects correctly. Both in the interest of redirecting older URLs that are now obsolete, and in the interest of handling "logical-looking" URLs (leaving out the lang_slug or version_slug shouldn't result in a 404), the following redirects are supported:

```
/           -> /en/latest/
/en/        -> /en/latest/
/latest/    -> /en/latest/
```

The language redirect will work for any of the defined `LANGUAGE_CODES` we support. The version redirect will work for supported versions.

Redirecting to a Page

You can link to a specific page and have it redirect to your default version. This is done with the `/page/` URL. For example:

```
pip.readthedocs.io/page/quickstart.html -> pip.readthedocs.io/en/latest/quickstart.
↪html
www.pip-installer.org/page/quickstart.html -> www.pip-installer.org/en/latest/
↪quickstart.html
```

This allows you to create links that are always up to date.

Another way to handle this is the *latest* version. You can set your `latest` version to a specific version and just always link to latest.

CHAPTER 33

Content Embedding

Using the Read the Docs JavaScript Client, or with basic calls to our REST API, you can retrieve embeddable content for use on your own site. Content is embedded in an iframe element, primarily for isolation. To get example usage of the API, see the tools tab under an active project and select the page and section that you would like to test.

Note: The client library is still alpha quality. This guide is still lacking advanced usage of the library, and information on styling the iframe content. We plan to have more usage outlined as the library matures.

Example usage of the client library:

```
var embed = Embed();
embed.section(
    'read-the-docs', 'latest', 'features', 'Read the Docs features',
    function (section) {
        section.insertContent($('#help'));
    }
);
```

CHAPTER 34

Changelog

Version 2.1.6

- @davidfischer: Promo contrast improvements (#3549)
- @humitos: Refactor run command outside a Build and Environment (#3542)
- @AnatoliyURL: Project in the local read the docs don't see tags. (#3534)
- @malarzm: searchtools.js missing init() call (#3532)
- @johanneskoester: Build failed without details (#3531)
- @danielmitterdorfer: "Edit on Github" points to non-existing commit (#3530)
- @lk-geimfari: No such file or directory: 'docs/requirements.txt' (#3529)
- @stsewd: Fix Good First Issue link (#3522)
- @Blendify: Remove RTD Theme workaround (#3519)
- @stsewd: Move project description to the top (#3510)
- @davidfischer: Switch to universal analytics (#3495)
- @davidfischer: Convert default dev cache to local memory (#3477)
- @nlgranger: Github service: cannot unlink after deleting account (#3374)
- @andrewgodwin: "stable" appearing to track future release branches (#3268)
- @skddc: Add JSDoc to docs build environment (#3069)
- @chummels: RTD building old "stable" docs instead of "latest" when auto-triggered from recent push (#2351)
- @cajus: Builds get stuck in "Cloning" state (#2047)
- @gossi: Cannot delete subproject (#1341)
- @gigster99: extension problem (#1059)

Version 2.1.5

- @ericholscher: Add GSOC 2018 page (#3518)
- @stsewd: Move project description to the top (#3510)
- @RichardLitt: Docs: Rename "Good First Bug" to "Good First Issue" (#3505)
- @stsewd: Fix regex for getting project and user (#3501)
- @ericholscher: Check to make sure changes exist in BitBucket pushes (#3480)
- @andrewgodwin: "stable" appearing to track future release branches (#3268)
- @cdeil: No module named pip in conda build (#2827)
- @Yaseenh: building project does not generate new pdf with changes in it (#2758)
- @chummels: RTD building old "stable" docs instead of "latest" when auto-triggered from recent push (#2351)
- @KeithWoods: GitHub edit link is aggressively stripped (#1788)

Version 2.1.4

- @davidfischer: Add programming language to API/READTHEDOCS_DATA (#3499)
- @ericholscher: Remove our mkdocs search override (#3496)
- @humitos: Better style (#3494)
- @humitos: Update README.rst (#3492)
- @davidfischer: Small formatting change to the Alabaster footer (#3491)
- @matsen: Fixing "reseting" misspelling. (#3487)
- @ericholscher: Add David to dev team listing (#3485)
- @ericholscher: Check to make sure changes exist in BitBucket pushes (#3480)
- @ericholscher: Use semvar for readthedocs-build to make bumping easier (#3475)
- @davidfischer: Add programming languages (#3471)
- @humitos: Remove TEMPLATE_LOADERS since it's the default (#3469)
- @Code0x58: Minor virtualenv upgrade (#3463)
- @humitos: Remove invite only message (#3456)
- @maxirus: Adding to Install Docs (#3455)
- @stsewd: Fix a little typo (#3448)
- @stsewd: Better autogenerated index file (#3447)
- @stsewd: Better help text for privacy level (#3444)
- @msyriac: Broken link URL changed fixes #3442 (#3443)
- @ericholscher: Fix git (#3441)
- @ericholscher: Properly slugify the alias on Project Relationships. (#3440)
- @stsewd: Don't show "build ideas" to unprivileged users (#3439)
- @Blendify: Docs: Point Theme docs to new website (#3438)

- @humitos: Do not use double quotes on git command with –format option (#3437)
- @ericholscher: Hack in a fix for missing version slug deploy that went out a while back (#3433)
- @humitos: Check versions used to create the venv and auto-wipe (#3432)
- @ericholscher: Upgrade psycopg2 (#3429)
- @humitos: Fix "Edit in Github" link (#3427)
- @ericholscher: Add celery theme to supported ad options (#3425)
- @humitos: Link to version detail page from build detail page (#3418)
- @humitos: Move wipe button to version detail page (#3417)
- @humitos: Show/Hide "See paid advertising" checkbox depending on USE_PROMOS (#3412)
- @benjaoming: Strip well-known version component origin/ from remote version (#3377)
- @humitos: Remove warnings from code (#3372)
- @ericholscher: Add docker image from the YAML config integration (#3339)
- @humitos: Show proper error to user when conf.py is not found (#3326)
- @humitos: Simple task to finish inactive builds (#3312)
- @techtonik: Fix Edit links if version is referenced by annotated tag (#3302)
- @Riyuzakii: changed from html to css (#2699)

Version 2.1.3

date Dec 21, 2017

- @ericholscher: Upgrade psycopg2 (#3429)
- @humitos: Fix "Edit in Github" link (#3427)
- @ericholscher: Add celery theme to supported ad options (#3425)
- @ericholscher: Only build travis push builds on master. (#3421)
- @ericholscher: Add concept of dashboard analytics code (#3420)
- @humitos: Use default avatar for User/Orgs in OAuth services (#3419)
- @humitos: Link to version detail page from build detail page (#3418)
- @humitos: Move wipe button to version detail page (#3417)
- @bieagrathara: 019 497 8360 (#3416)
- @bieagrathara: rew (#3415)
- @tony: lint prospector task failing (#3414)
- @humitos: Remove extra 's' (#3413)
- @humitos: Show/Hide "See paid advertising" checkbox depending on USE_PROMOS (#3412)
- @accraze: Removing talks about RTD page (#3410)
- @humitos: Pin pylint to 1.7.5 and fix docstring styling (#3408)
- @agjohnson: Update style and copy on abandonment docs (#3406)

- @agjohnson: Update changelog more consistently (#3405)
- @agjohnson: Update prerelease invoke command to call with explicit path (#3404)
- @ericholscher: Fix changelog command (#3403)
- @agjohnson: Fix lint error (#3402)
- @julienmalard: Recent builds are missing translated languages links (#3401)
- @stsewd: Remove copyright application (#3400)
- @humitos: Show connect buttons for installed apps only (#3394)
- @agjohnson: Fix display of build advice (#3390)
- @agjohnson: Don't display the build suggestions div if there are no suggestions (#3389)
- @ericholscher: Pass more data into the redirects. (#3388)
- @ericholscher: Fix issue where you couldn't edit your canonical domain. (#3387)
- @benjaoming: Strip well-known version component origin/ from remote version (#3377)
- @humitos: Remove warnings from code (#3372)
- @JavaDevVictoria: Updated python.setup_py_install to be true (#3357)
- @humitos: Use default avatars for GitLab/GitHub/Bitbucket integrations (users/organizations) (#3353)
- @jonrkarr: Error in YAML configuration docs: default value for `python.setup_py_install` should be `true` (#3334)
- @humitos: Show proper error to user when conf.py is not found (#3326)
- @MikeHart85: Badges aren't updating due to being cached on GitHub. (#3323)
- @humitos: Simple task to finish inactive builds (#3312)
- @techtonik: Fix Edit links if version is referenced by annotated tag (#3302)
- @humitos: Remove/Update talks about RTD page (#3283)
- @gawel: Regain pyquery project ownership (#3281)
- @dialex: Build passed but I can't see the documentation (maze screen) (#3246)
- @makixx: Account is inactive (#3241)
- @agjohnson: Cleanup misreported failed builds (#3230)
- @cokelaer: links to github are broken (#3203)
- @agjohnson: Remove copyright application (#3199)
- @shacharoo: Unable to register after deleting my account (#3189)
- @gtalarico: 3 week old Build Stuck Cloning (#3126)
- @agjohnson: Regressions with conf.py and error reporting (#2963)
- @agjohnson: Can't edit canonical domain (#2922)
- @virtuald: Documentation stuck in 'cloning' state (#2795)
- @Riyuzakii: changed from html to css (#2699)
- @tjanez: Support specifying 'python setup.py build_sphinx' as an alternative build command (#1857)

- @bdarnell: Broken edit links (#1637)

Version 2.1.2

- @agjohnson: Update changelog more consistently (#3405)
- @agjohnson: Update prerelease invoke command to call with explicit path (#3404)
- @agjohnson: Fix lint error (#3402)
- @stsewd: Remove copyright application (#3400)
- @humitos: Show connect buttons for installed apps only (#3394)
- @agjohnson: Don't display the build suggestions div if there are no suggestions (#3389)
- @jonrkarr: Error in YAML configuration docs: default value for `python.setup_py_install` should be `true` (#3334)
- @humitos: Simple task to finish inactive builds (#3312)
- @agjohnson: Cleanup misreported failed builds (#3230)
- @agjohnson: Remove copyright application (#3199)

Version 2.1.1

Release information missing

Version 2.1.0

- @ericholscher: Revert "Merge pull request #3336 from rtfd/use-active-for-stable" (#3368)
- @agjohnson: Revert "Do not split before first argument (#3333)" (#3366)
- @ericholscher: Remove pitch from ethical ads page, point folks to actual pitch page. (#3365)
- @agjohnson: Add changelog and changelog automation (#3364)
- @ericholscher: Fix mkdocs search. (#3361)
- @ericholscher: Email sending: Allow kwargs for other options (#3355)
- @ericholscher: Try and get folks to put more tags. (#3350)
- @ericholscher: Suggest wiping your environment to folks with bad build outcomes. (#3347)
- @humitos: GitLab Integration (#3327)
- @jimfulton: Draft policy for claiming existing project names. (#3314)
- @agjohnson: More logic changes to error reporting, cleanup (#3310)
- @safwanrahman: [Fix #3182] Better user deletion (#3214)
- @ericholscher: Better User deletion (#3182)
- @RichardLitt: Add `Needed: replication` label (#3138)
- @josejrobles: Replaced usage of deprecated function get_fields_with_model with new ... (#3052)

- @ericholscher: Don't delete the subprojects directory on sync of superproject (#3042)
- @andrew: Pass query string when redirecting, fixes #2595 (#3001)
- @saily: Add GitLab repo sync and webhook support (#1870)
- @destroyerofbuilds: Setup GitLab Web Hook on Project Import (#1443)
- @takotuesday: Add GitLab Provider from django-allauth (#1441)

Version 2.0

- @ericholscher: Email sending: Allow kwargs for other options (#3355)
- @ericholscher: Try and get folks to put more tags. (#3350)
- @ericholscher: Small changes to email sending to enable from email (#3349)
- @dplanella: Duplicate TOC entries (#3345)
- @ericholscher: Small tweaks to ethical ads page (#3344)
- @agjohnson: Fix python usage around oauth pagination (#3342)
- @tony: Fix isort link (#3340)
- @ericholscher: Change stable version switching to respect `active` (#3336)
- @ericholscher: Allow superusers to pass admin & member tests for projects (#3335)
- @humitos: Do not split before first argument (#3333)
- @humitos: Update docs for pre-commit (auto linting) (#3332)
- @humitos: Take preferece of tags over branches when selecting the stable version (#3331)
- @humitos: Add prospector as a pre-commit hook (#3328)
- @andrewgodwin: "stable" appearing to track future release branches (#3268)
- @humitos: Config files for auto linting (#3264)
- @mekrip: Build is not working (#3223)
- @skddc: Add JSDoc to docs build environment (#3069)
- @jakirkham: Specifying conda version used (#2076)
- @agjohnson: Document code style guidelines (#1475)

Previous releases

Starting with version `2.0`, we will be incrementing the Read the Docs version based on semantic versioning principles, and will be automating the update of our changelog.

Below are some historical changes from when we have tried to add information here in the past

July 23, 2015

- Django 1.8 Support Merged

Code Notes

- Updated Django from `1.6.11` to `1.8.3`.
- Removed South and ported the South migrations to Django's migration framework.
- Updated django-celery from `3.0.23` to `3.1.26` as django-celery 3.0.x does not support Django 1.8.
- Updated Celery from `3.0.24` to `3.1.18` because we had to update django-celery. We need to test this extensively and might need to think about using the new Celery API directly and dropping django-celery. See release notes: http://docs.celeryproject.org/en/latest/whatsnew-3.1.html
- Updated tastypie from `0.11.1` to current master (commit `1e1aff3dd4dcd21669e9c68bd7681253b286b856`) as 0.11.x is not compatible with Django 1.8. No surprises expected but we should ask for a proper release, see release notes: https://github.com/django-tastypie/django-tastypie/blob/master/docs/release_notes/v0.12.0.rst
- Updated django-oauth from `0.16.1` to `0.21.0`. No surprises expected, see release notes [in the docs](https://django-allauth.readthedocs.org/en/latest/changelog.html) and [finer grained in the repo](https://github.com/pennersr/django-allauth/blob/9123223f167959e4e5c4074408db068f725559d1/ChangeLog#L1-169)
- Updated django-guardian from `1.2.0` to `1.3.0` to gain Django 1.8 support. No surprises expected, see release notes: https://github.com/lukaszb/django-guardian/blob/devel/CHANGES
- Using `django-formtools` instead of removed `django.contrib.formtools` now. Based on the Django release notes, these modules are the same except of the package name.
- Updated pytest-django from `2.6.2` to `2.8.0`. No tests required, but running the testsuite :smile:
- Updated psycopg2 from 2.4 to 2.4.6 as 2.4.5 is required by Django 1.8. No trouble expected as Django is the layer between us and psycopg2. Also it's only a minor version upgrade. Release notes: http://initd.org/psycopg/docs/news.html#what-s-new-in-psycopg-2-4-6
- Added `django.setup()` to `conf.py` to load django properly for doc builds.
- Added migrations for all apps with models in the `readthedocs/` directory

Deployment Notes

After you have updated the code and installed the new dependencies, you need to run these commands on the server:

```
python manage.py migrate contenttypes
python manage.py migrate projects 0002 --fake
python manage.py migrate --fake-initial
```

Locally I had trouble in a test environment that pip did not update to the specified commit of tastypie. It might be required to use `pip install -U -r requirements/deploy.txt` during deployment.

Development Update Notes

The readthedocs developers need to execute these commands when switching to this branch (or when this got merged into master):

- **Before updating** please make sure that all migrations are applied:

  ```
  python manage.py syncdb
  python manage.py migrate
  ```

- Update the codebase: `git pull`
- You need to update the requirements with `pip install -r requirements.txt`

- Now you need to fake the initial migrations:

```
python manage.py migrate contenttypes
python manage.py migrate projects 0002 --fake
python manage.py migrate --fake-initial
```

CHAPTER 35

Installation

Here is a step by step plan on how to install Read the Docs. It will get you to a point of having a local running instance.

Warning: Read the Docs does not itself run under Python 3 (though it does support building documentation for Python 3 projects). Please ensure the subsequent steps are performed using Python 2.7.

First, obtain Python 2.7 and virtualenv if you do not already have them. Using a virtual environment will make the installation easier, and will help to avoid clutter in your system-wide libraries. You will also need Git in order to clone the repository. If you plan to import Python 3 project to your RTD then you'll need to install Python 3 with virtualenv in your system as well.

Note: Requires Git version >=1.9

Note: If you are having trouble on OS X Mavericks (or possibly other versions of OS X) with building `lxml`, you probably might need to use Homebrew to `brew install libxml2`, and invoke the install with:

```
CFLAGS=-I/usr/local/opt/libxml2/include/libxml2 \
LDFLAGS=-L/usr/local/opt/libxml2/lib \
pip install -r requirements.txt
```

Note: Linux users may find they need to install a few additional packages in order to successfully execute `pip install -r requirements.txt`. For example, a clean install of Ubuntu 14.04 LTS will require the following packages:

```
sudo apt-get install build-essential
sudo apt-get install python-dev python-pip python-setuptools
sudo apt-get install libxml2-dev libxslt1-dev zlib1g-dev
```

CentOS/RHEL 7 will require:

```
sudo yum install python-devel python-pip libxml2-devel libxslt-devel
```

Users of other Linux distributions may need to install the equivalent packages, depending on their system configuration.

Note: If you want full support for searching inside your Read the Docs site you will need to install Elasticsearch.

Ubuntu users could install this package as following:

```
sudo apt-get install elasticsearch
```

Note: Besides the Python specific dependencies, you will also need Redis.

Ubuntu users could install this package as following:

```
sudo apt-get install redis-server
```

You will need to verify that your pip version is higher than 1.5 you can do this as such:

```
pip --version
```

If this is not the case please update your pip version before continuing:

```
pip install --upgrade pip
```

Once you have these, create a virtual environment somewhere on your disk, then activate it:

```
virtualenv rtd
cd rtd
source bin/activate
```

Create a folder in here, and clone the repository:

```
mkdir checkouts
cd checkouts
git clone https://github.com/rtfd/readthedocs.org.git
```

Next, install the dependencies using `pip` (included inside of virtualenv):

```
cd readthedocs.org
pip install -r requirements.txt
```

This may take a while, so go grab a beverage. When it's done, build your database:

```
python manage.py migrate
```

Then please create a superuser account for Django:

```
python manage.py createsuperuser
```

Now let's properly generate the static assets:

```
python manage.py collectstatic
```

By now, it is the right time to load in a couple users and a test project:

```
python manage.py loaddata test_data
```

Note: If you do not opt to install test data, you'll need to create an account for API use and set `SLUMBER_USERNAME` and `SLUMBER_PASSWORD` in order for everything to work properly.

Finally, you're ready to start the webserver:

```
python manage.py runserver
```

Visit http://127.0.0.1:8000/ in your browser to see how it looks; you can use the admin interface via http://127.0.0.1:8000/admin (logging in with the superuser account you just created).

For builds to properly kick off as expected, it is necessary the port you're serving on (i.e. `runserver 0.0.0.0:8080`) match the port defined in `PRODUCTION_DOMAIN`. You can utilize `local_settings.py` to modify this. (By default, it's `localhost:8000`)

While the webserver is running, you can build documentation for the latest version of a project called 'pip' with the `update_repos` command. You can replace 'pip' with the name of any added project:

```
python manage.py update_repos pip
```

What's available

After registering with the site (or creating yourself a superuser account), you will be able to log in and view the dashboard.

Importing your docs

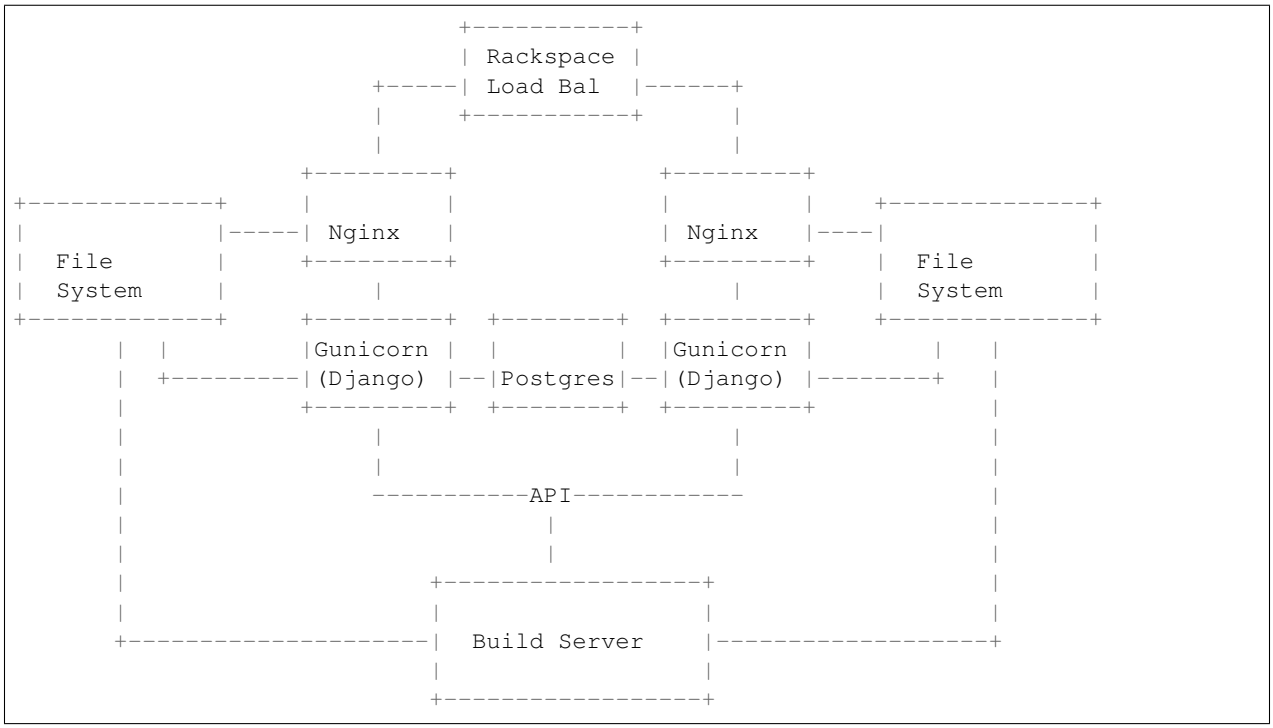
One of the goals of readthedocs.org is to make it easy for any open source developer to get high quality hosted docs with great visibility! Simply provide us with the clone URL to your repo, we'll pull your code, extract your docs, and build them! We make available a post-commit webhook that can be configured to update the docs whenever you commit to your repo. See our *Getting Started* page to learn more.

CHAPTER 36

Architecture

Read the Docs is architected to be highly available. A lot of projects host their documentation with us, so we have built the site so that it shouldn't go down. The load balancer is the only real single point of failure currently. This means mainly that if the network to the load balancer goes down, we have issues.

Diagram



CHAPTER 37

Testing

Before contributing to Read the Docs, make sure your patch passes our test suite and your code style passes our code linting suite.

Read the Docs uses Tox to execute testing and linting procedures. Tox is the only dependency you need to run linting or our test suite, the remainder of our requirements will be installed by Tox into environment specific virtualenv paths. Before testing, make sure you have Tox installed:

```
pip install tox
```

To run the full test and lint suite against your changes, simply run Tox. Tox should return without any errors. You can run Tox against all of our environments by running:

```
tox
```

To target a specific environment:

```
tox -e py27
```

The `tox` configuration has the following environments configured. You can target a single environment to limit the test suite:

```
py27
    Run our test suite using Python 2.7

lint
    Run code linting using `Prospector`_. This currently runs `pylint`_,
    `pyflakes`_, `pep8`_ and other linting tools.

docs
    Test documentation compilation with Sphinx.
```

Continuous Integration

The RTD test suite is exercised by Travis CI on every push to our repo at GitHub. You can check out the current build status: https://travis-ci.org/rtfd/readthedocs.org

CHAPTER 38

Building and Contributing to Documentation

As one might expect, the documentation for Read the Docs is built using Sphinx and hosted on Read the Docs. The docs are kept in the `docs/` directory at the top of the source tree.

You can build the docs by installing `Sphinx` and running:

```
# in the docs directory
make html
```

Please follow these guidelines when updating our docs. Let us know if you have any questions or something isn't clear.

The brand

We are called **Read the Docs**. The *the* is not capitalized.

We do however use the acronym **RTD**.

Titles

For page titles, or Heading1 as they are sometimes called, we use title-case.

If the page includes multiple sub-headings (H2, H3), we usually use sentence-case unless the titles include terminology that is supposed to be capitalized.

Content

- Do not break the content across multiple lines at 80 characters, but rather break them on semantic meaning (e.g. periods or commas). Read more about this here.
- If you are cross-referencing to a different page within our website, use the `doc` directive and not a hyperlink.

CHAPTER 39

Front End Development

Background

Note: Consider this the canonical resource for contributing Javascript and CSS. We are currently in the process of modernizing our front end development procedures. You will see a lot of different styles around the code base for front end JavaScript and CSS.

Our modern front end development stack includes the following tools:

- Gulp
- Bower
- Browserify
- Debowerify
- And soon, LESS

We use the following libraries:

- Knockout
- jQuery
- Several jQuery plugins

Previously, JavaScript development has been done in monolithic files or inside templates. jQuery was added as a global object via an include in the base template to an external source. There are no standards currently to JavaScript libraries, this aims to solve that.

The requirements for modernizing our front end code are:

- Code should be modular and testable. One-off chunks of JavaScript in templates or in large monolithic files are not easily testable. We currently have no JavaScript tests.
- Reduce code duplication.

- Easy JavaScript dependency management.

Modularizing code with Browserify is a good first step. In this development workflow, major dependencies commonly used across JavaScript includes are installed with Bower for testing, and vendorized as standalone libraries via Gulp and Browserify. This way, we can easily test our JavaScript libraries against jQuery/etc, and have the flexibility of modularizing our code. See *JavaScript Bundles* for more information on what and how we are bundling.

To ease deployment and contributions, bundled JavaScript is checked into the repository for now. This ensures new contributors don't need an additional front end stack just for making changes to our Python code base. In the future, this may change, so that assets are compiled before deployment, however as our front end assets are in a state of flux, it's easier to keep absolute sources checked in.

Getting Started

You will need a working version of Node and NPM to get started. We won't cover that here, as it varies from platform to platform.

To install these tools and dependencies:

```
npm install
```

This will install locally to the project, not globally. You can install globally if you wish, otherwise make sure `node_modules/.bin` is in your PATH.

Next, install front end dependencies:

```
bower install
```

The sources for our bundles are found in the per-application path `static-src`, which has the same directory structure as `static`. Files in `static-src` are compiled to `static` for static file collection in Django. Don't edit files in `static` directly, unless you are sure there isn't a source file that will compile over your changes.

To test changes while developing, which will watch source files for changes and compile as necessary, you can run Gulp with our development target:

```
gulp dev
```

Once you are satisfied with your changes, finalize the bundles (this will minify library sources):

```
gulp build
```

If you updated any of our vendor libraries, compile those:

```
gulp vendor
```

Make sure to check in both files under `static` and `static-src`.

Making Changes

If you are creating a new library, or a new library entry point, make sure to define the application source file in `gulpfile.js`, this is not handled automatically right now.

If you are bringing in a new vendor library, make sure to define the bundles you are going to create in `gulpfile.js` as well.

Tests should be included per-application, in a path called `tests`, under the `static-src/js` path you are working in. Currently, we still need a test runner that accumulates these files.

Deployment

If merging several branches with JavaScript changes, it's important to do a final post-merge bundle. Follow the steps above to rebundle the libraries, and check in any changed libraries.

JavaScript Bundles

There are several components to our bundling scheme:

Vendor library We repackage these using Browserify, Bower, and Debowerify to make these libraries available by a `require` statement. Vendor libraries are packaged separately from our JavaScript libraries, because we use the vendor libraries in multiple locations. Libraries bundled this way with Browserify are available to our libraries via `require` and will back down to finding the object on the global `window` scope.

Vendor libraries should only include libraries we are commonly reusing. This currently includes `jQuery` and `Knockout`. These modules will be excluded from libraries by special includes in our `gulpfile.js`.

Minor third party libraries These libraries are maybe used in one or two locations. They are installed via Bower and included in the output library file. Because we aren't reusing them commonly, they don't require a separate bundle or separate include. Examples here would include jQuery plugins used on one off forms, such as jQuery Payments.

Our libraries These libraries are bundled up excluding vendor libraries ignored by rules in our `gulpfile.js`. These files should be organized by function and can be split up into multiple files per application.

Entry points to libraries must be defined in `gulpfile.js` for now. We don't have a defined directory structure that would make it easy to imply the entry point to an application library.

CHAPTER 40

Build Environments

Read the Docs uses container virtualization to encapsulate documentation build processes. Each build spins up a new virtual machine using our base image, which is an image with the minimum necessary components required to build documentation. Virtual machines are limiting in CPU time and memory, which aims to reduce excessive usage of build resources.

Setup

Build environments use Docker to handle container virtualization. To perform any development on the Docker build system, you will need to set up Docker on your host system. Setup of Docker will vary by system, and so is out of the scope of this documentation.

Once you have Docker set up, you will need to pull down our build image. These images are found on our Docker Hub repository, the source comes from our container image repo.

To get started using Docker for build environments, you'll need to pull down at least one build image. For example, to pull down our latest image:

```
docker pull readthedocs/build:latest
```

The default image used by our build servers is `readthedocs/build:2.0`. This would be a good place to start testing as the `latest` version could operate differently. See `DOCKER_IMAGE` below for setting this configuration option.

After this image is downloaded, you can update your settings to use the new image – see *Configuration*.

Configuration

There are several settings used to configure usage of virtual machines:

DOCKER_ENABLE True/False value used to enable the Docker build environment.

Default: `False`

DOCKER_LIMITS A dictionary of limits to virtual machines. These limits include:

time An integer representing the total allowed time limit (in seconds) of build processes. This time limit affects the parent process to the virtual machine and will force a virtual machine to die if a build is still running after the allotted time expires.

memory The maximum memory allocated to the virtual machine. If this limit is hit, build processes will be automatically killed. Examples: '200m' for 200MB of total memory, or '2g' for 2GB of total memory.

Default: `None`

DOCKER_IMAGE Tag of a Docker image to use as a base image.

Default: `readthedocs/build:2.0`

DOCKER_SOCKET URI of the socket to connect to the Docker daemon. Examples include: `unix:///var/run/docker.sock` and `tcp://127.0.0.1:2375`.

Default: `None`

DOCKER_VERSION Version of the API to use for the Docker API client.

Default: `None`

CHAPTER 41

How we use symlinks

Read the Docs stays highly available by serving all documentation pages out of nginx. This means that they never hit our Python layer, meaning that they never hit our database. This reduces the total number of servers to serve a request to 1, each of which is redundant.

Nginx

We handle a couple of different types of requests in nginx:

- Requests to a readthedocs.org subdomain
- Requests to a CNAME

Subdomains

For subdomains this is a simple lookup. This doesn't require symlinks, but it shows the basic logic that we need to replicate.

When a user navigates to `http://pip.readthedocs.org/en/latest/`, we know that they want the pip documentation. So we simply serve them the documentation:

```
location ~ ^/en/(.+)/(.*) {
    alias /home/docs/checkouts/readthedocs.org/user_builds/$domain/rtd-builds/$1/$2;
    error_page 404 = @fallback;
    error_page 500 = @fallback;
}

location @fallback {
    proxy_pass http://127.0.0.1:8888;
    proxy_set_header Host $host;
    proxy_set_header X-Real-IP $remote_addr;
    proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
```

```
    add_header X-Deity Asgard;
}
```

Note: The `@fallback` directive is hit when we don't find the proper file. This will cause things to hit the Python backend, so that proper action can be taken.

CNAMEs

CNAMEs add a bit of difficulty, because at the nginx layer we don't know what documentation to serve. When someone requests `http://docs.fabfile.org/en/latest/`, we can't look at the URL to know to serve the `fabric` docs.

This is where symlinks come in. When someone requests `http://docs.fabfile.org/en/latest/` the first time, it hits the Python layer. In that Python layer we record that `docs.fabfile.org` points at `fabric`. When we build the `fabric` docs, we create a symlink for all domains that have pointed at `fabric` before.

So, when we get a request for `docs.fabfile.org` in the future, we will be able to serve it directly from nginx. In this example, $host would be `docs.fabfile.org`:

```
location ~ ^/en/(?P<doc_version>.+)/(?P<path>.*) {
    alias /home/docs/checkouts/readthedocs.org/cnames/$host/$doc_version/$path;
    error_page 404 = @fallback;
    error_page 500 = @fallback;
}
```

Notice that nowhere in the above path is the project's slug mentioned. It is simply there in the symlink in the cnames directory, and the docs are served from there.

CHAPTER 42

Interesting Settings

SLUMBER_USERNAME

Default: `test`

The username to use when connecting to the Read the Docs API. Used for hitting the API while building the docs.

SLUMBER_PASSWORD

Default: `test`

The password to use when connecting to the Read the Docs API. Used for hitting the API while building the docs.

USE_SUBDOMAIN

Default: `False`

Whether to use subdomains in URLs on the site, or the Django-served content. When used in production, this should be `True`, as Nginx will serve this content. During development and other possible deployments, this might be `False`.

PRODUCTION_DOMAIN

Default: `localhost:8000`

This is the domain that gets linked to throughout the site when used in production. It depends on `USE_SUBDOMAIN`, otherwise it isn't used.

MULTIPLE_APP_SERVERS

Default: `None`

This is a list of application servers that built documentation is copied to. This allows you to run an independent build server, and then have it rsync your built documentation across multiple front end documentation/app servers.

DEFAULT_PRIVACY_LEVEL

Default: `None`

What privacy projects default to having. Generally set to `public`. Also acts as a proxy setting for blocking certain historically insecure options, like serving generated artifacts directly from the media server.

INDEX_ONLY_LATEST

Default: `None`

In search, only index the `latest` version of a Project.

DOCUMENT_PYQUERY_PATH

Default: `None`

The Pyquery path to an HTML element that is the root of your document. This is used for making sure we are only searching the main content of a document.

USE_PIP_INSTALL

Default: `None`

Whether to use `pip install .` or `python setup.py install` when installing packages into the Virtualenv. Default is to use `python setup.py install`.

PUBLIC_DOMAIN

Default: `None`

A special domain for serving public documentation. If set, public docs will be linked here instead of the `PRODUCTION_DOMAIN`.

ALLOW_ADMIN

Default: `None`

Whether to include `django.contrib.admin` in the URL's.

CHAPTER 43

Internationalization

This document covers the details regarding internationalization and localization that are applied in Read the Docs. The guidelines described are mostly based on Kitsune's localization documentation.

As with most of the Django applications out there, Read the Docs' i18n/l10n framework is based on GNU gettext. Crowd-sourced localization is optionally available at Transifex.

For more information about the general ideas, look at this document: http://www.gnu.org/software/gettext/manual/html_node/Concepts.html

Making Strings Localizable

Making strings in templates localizable is exceptionally easy. Making strings in Python localizable is a little more complicated. The short answer, though, is to just wrap the string in `_()`.

Interpolation

A string is often a combination of a fixed string and something changing, for example, `Welcome, James` is a combination of the fixed part `Welcome,`, and the changing part `James`. The naive solution is to localize the first part and then follow it with the name:

```
_('Welcome, ') + username
```

This is **wrong!**

In some locales, the word order may be different. Use Python string formatting to interpolate the changing part into the string:

```
_('Welcome, {name}').format(name=username)
```

Python gives you a lot of ways to interpolate strings. The best way is to use Py3k formatting and kwargs. That's the clearest for localizers.

Localization Comments

Sometimes, it can help localizers to describe where a string comes from, particularly if it can be difficult to find in the interface, or is not very self-descriptive (e.g. very short strings). If you immediately precede the string with a comment that starts with `Translators:`, the comment will be added to the PO file, and visible to localizers.

Example:

```
DEFAULT_THEME_CHOICES = (
    # Translators: This is a name of a Sphinx theme.
    (THEME_DEFAULT, _('Default')),
    # Translators: This is a name of a Sphinx theme.
    (THEME_SPHINX, _('Sphinx Docs')),
    # Translators: This is a name of a Sphinx theme.
    (THEME_TRADITIONAL, _('Traditional')),
    # Translators: This is a name of a Sphinx theme.
    (THEME_NATURE, _('Nature')),
    # Translators: This is a name of a Sphinx theme.
    (THEME_HAIKU, _('Haiku')),
)
```

Adding Context with msgctxt

Strings may be the same in English, but different in other languages. English, for example, has no grammatical gender, and sometimes the noun and verb forms of a word are identical.

To make it possible to localize these correctly, we can add "context" (known in gettext as *msgctxt*) to differentiate two otherwise identical strings. Django provides a `pgettext()` function for this.

For example, the string *Search* may be a noun or a verb in English. In a heading, it may be considered a noun, but on a button, it may be a verb. It's appropriate to add a context (like *button*) to one of them.

Generally, we should only add context if we are sure the strings aren't used in the same way, or if localizers ask us to.

Example:

```
from django.utils.translation import pgettext

month = pgettext("text for the search button on the form", "Search")
```

Plurals

You have 1 new messages grates on discerning ears. Fortunately, gettext gives us a way to fix that in English *and* other locales, the `ngettext()` function:

```
ngettext('singular sentence', 'plural sentence', count)
```

A more realistic example might be:

```
ngettext('Found {count} result.',
         'Found {count} results',
         len(results)).format(count=len(results))
```

This method takes three arguments because English only needs three, i.e., zero is considered "plural" for English. Other languages may have different plural rules, and require different phrases for, say 0, 1, 2-3, 4-10, >10. That's absolutely fine, and gettext makes it possible.

Strings in Templates

When putting new text into a template, all you need to do is wrap it in a `{% trans %}` template tag:

```
<h1>{% trans "Heading" %}</h1>
```

Context can be added, too:

```
<h1>{% trans "Heading" context "section name" %}</h1>
```

Comments for translators need to precede the internationalized text and must start with the `Translators:` keyword.:

```
{# Translators: This heading is displayed in the user's profile page #}
<h1>{% trans "Heading" %}</h1>
```

To interpolate, you need to use the alternative and more verbose `{% blocktrans %}` template tag — it's actually a block:

```
{% blocktrans %}Welcome, {{ name }}!{% endblocktrans %}
```

Note that the `{{ name }}` variable needs to exist in the template context.

In some situations, it's desirable to evaluate template expressions such as filters or accessing object attributes. You can't do that within the `{% blocktrans %}` block, so you need to bind the expression to a local variable first:

```
{% blocktrans with revision.created_date|timesince as timesince %}
{{ revision }} {{ timesince }} ago
{% endblocktrans %}

{% blocktrans with project.name as name %}Delete {{ name }}?{% endblocktrans %}
```

`{% blocktrans %}` also provides pluralization. For that you need to bind a counter with the name `count` and provide a plural translation after the `{% plural %}` tag:

```
{% blocktrans with amount=article.price count years=i.length %}
That will cost $ {{ amount }} per year.
{% plural %}
That will cost $ {{ amount }} per {{ years }} years.
{% endblocktrans %}
```

Strings in Python

Note: Whenever you are adding a string in Python, ask yourself if it really needs to be there, or if it should be in the template. Keep logic and presentation separate!

Strings in Python are more complex for two reasons:

1. We need to make sure we're always using Unicode strings and the Unicode-friendly versions of the functions.
2. If you use the `ugettext()` function in the wrong place, the string may end up in the wrong locale!

Here's how you might localize a string in a view:

```
from django.utils.translation import ugettext as _

def my_view(request):
    if request.user.is_superuser:
        msg = _(u'Oh hi, staff!')
    else:
        msg = _(u'You are not staff!')
```

Interpolation is done through normal Python string formatting:

```
msg = _(u'Oh, hi, {user}').format(user=request.user.username)
```

Context information can be supplied by using the `pgettext()` function:

```
msg = pgettext('the context', 'Search')
```

Translator comments are normal one-line Python comments:

```
# Translators: A message to users.
msg = _(u'Oh, hi there!')
```

If you need to use plurals, import the `ungettext()` function:

```
from django.utils.translation import ungettext

n = len(results)
msg = ungettext('Found {0} result', 'Found {0} results', n).format(n)
```

Lazily Translated Strings

You can use `ugettext()` or `ungettext()` only in views or functions called from views. If the function will be evaluated when the module is loaded, then the string may end up in English or the locale of the last request!

Examples include strings in module-level code, arguments to functions in class definitions, strings in functions called from outside the context of a view. To internationalize these strings, you need to use the `_lazy` versions of the above methods, `ugettext_lazy()` and `ungettext_lazy()`. The result doesn't get translated until it is evaluated as a string, for example by being output or passed to `unicode()`:

```
from django.utils.translation import ugettext_lazy as _

class UserProfileForm(forms.ModelForm):
    first_name = CharField(label=_('First name'), required=False)
    last_name = CharField(label=_('Last name'), required=False)
```

In case you want to provide context to a lazily-evaluated gettext string, you will need to use `pgettext_lazy()`.

Administrative Tasks

Updating Localization Files

To update the translation source files (eg if you changed or added translatable strings in the templates or Python code) you should run `python manage.py makemessages -l <language>` in the project's root directory (substitute `<language>` with a valid language code).

The updated files can now be localized in a PO editor or crowd-sourced online translation tool.

Compiling to MO

Gettext doesn't parse any text files, it reads a binary format for faster performance. To compile the latest PO files in the repository, Django provides the `compilemessages` management command. For example, to compile all the available localizations, just run:

```
$ python manage.py compilemessages -a
```

You will need to do this every time you want to push updated translations to the live site.

Also, note that it's not a good idea to track MO files in version control, since they would need to be updated at the same pace PO files are updated, so it's silly and not worth it. They are ignored by `.gitignore`, but please make sure you don't forcibly add them to the repository.

Transifex Integration

To push updated translation source files to Transifex, run `tx push -s` (for English) or `tx push -t <language>` (for non-English).

To pull changes from Transifex, run `tx pull -a`. Note that Transifex does not compile the translation files, so you have to do this after the pull (see the *Compiling to MO* section).

For more information about the `tx` command, read the Transifex client's help pages.

CHAPTER 44

Overview of issue labels

Here is a full list of labels that we use in the GitHub issue tracker and what they stand for.

Bug An issue describing unexpected or malicious behaviour of the readthedocs.org software.

Community Effort Tickets with this label are valid issues that the core team thinks are worth to fix or implement in the future. However the core team's resources are too scarce to address these issues. Tickets marked with this label are issues that the core team will **not** work on, but contributions from the community are very welcome.

Design Issues related to the UI of the readthedocs.org website.

Enhancement Feature requests and ideas about how to make readthedocs.org better in general will have this label.

Feature Overview Features that are too big to be tackled in one ticket are split up into multiple tickets. A feature overview ticket is then explaining the overarching idea. See the milestone of the feature overview ticket for all related tickets.

Good First Issue This label marks tickets that are easy to get started with. The ticket should be ideal for beginners to dive into the code base.

High Priority Tickets with this label should be resolved as quickly as possible.

Mkdocs Tickets that are related to the Mkdocs builder.

Needed: design decision Tickets that need a design decision are blocked for development until a project leader clarifies the way in which the issue should be approached.

Needed: documentation If an issue involves creating or refining documentation, this label will be assigned.

Needed: more information This label indicates that the issue has not enough information in order to decide on how to go forward. See the documentation about our *triage process* for more information.

Needed: patch This label indicates that a patch is required in order to resolve the ticket. A fix should be proposed via a pull request on GitHub.

Needed: tests This label indicates that a better test coverage is required to resolve the ticket. New tests should be proposed via a pull request on GitHub.

Needed: replication This label indicates that a bug has been reported, but has not been successfully replicated by another user or contributor yet.

Operations Tickets that require changes in the server infrastructure.

PR: ready for review Pull Requests that are considered complete. A review by at least one core developer is required prior to merging it.

PR: work in progress Pull Requests that are not complete yet. A final review is not possible yet, but every Pull Request is open for discussion.

Sprintable Sprintable are all tickets that have the right amount of scope to be handled during a sprint. They are very focused and encapsulated.

Status: blocked The ticket cannot be resolved until some other ticket has been closed. See the ticket's log for which ticket is blocking this ticket.

Status: duplicate See the ticket's log to find the original ticket that this one is a duplicate of.

Status: invalid A ticket is invalid if the reported issue cannot be reproduced.

Status: rejected A ticket gets rejected if the proposed enhancement is not in line with the overall goals of readthedocs.org.

Support Questions that needs answering but do not require code changes or issues that only require a one time action on the server will have this label. See the documentation about our *triage process* for more information.

CHAPTER 45

API

This is the Read The Docs API documentation, autogenerated from the source code.

readthedocs.bookmarks

readthedocs.bookmarks.admin

Django admin interface for `Bookmark`.

readthedocs.bookmarks.models

Models for the bookmarks app.

class `readthedocs.bookmarks.models.`**Bookmark**(**args*, ***kwargs*)
A user's bookmark of a `Project`, `Version`, and page.

readthedocs.bookmarks.urls

URL config for the bookmarks app.

readthedocs.bookmarks.views

Views for the bookmarks app.

class `readthedocs.bookmarks.views.`**BookmarkAddView**(***kwargs*)
Adds bookmarks in response to POST requests.

post(*request*, **args*, ***kwargs*)
Add a new bookmark for the current user.

Points at `project`, `version`, `page`, and `url`.

class readthedocs.bookmarks.views.**BookmarkListView**(***kwargs*)
Displays all of a logged-in user's bookmarks.

model
alias of Bookmark

class readthedocs.bookmarks.views.**BookmarkRemoveView**(***kwargs*)
Deletes a user's bookmark in response to a POST request.

Renders a delete? confirmation page in response to a GET request.

post(*request*, **args*, ***kwargs*)
Delete a bookmark.

Uses the primary key from the URL or JSON data from the request.

readthedocs.builds

readthedocs.builds.admin

Django admin interface for Build and related models.

readthedocs.builds.models

Models for the builds app.

class readthedocs.builds.models.**APIVersion**(**args*, ***kwargs*)
Version proxy model for API data deserialization.

This replaces the pattern where API data was deserialized into a mocked **:py:cls:'Version'** object. This pattern was confusing, as it was not explicit as to what form of object you were working with – API backed or database backed.

This model preserves the Version model methods, allowing for overrides on model field differences. This model pattern will generally only be used on builder instances, where we are interacting solely with API data.

class readthedocs.builds.models.**Build**(**args*, ***kwargs*)
Build data.

finished
Return if build has a finished state.

class readthedocs.builds.models.**BuildCommandResult**(**args*, ***kwargs*)
Build command for a Build.

run_time
Total command runtime in seconds.

class readthedocs.builds.models.**BuildCommandResultMixin**
Mixin for common command result methods/properties.

Shared methods between the database model *BuildCommandResult* and non-model respresentations of build command results from the API

failed
Did the command exit with a failing exit code.

Helper for inverse of *successful()*

successful
Did the command exit with a successful exit code.

class readthedocs.builds.models.**Version**(**args*, ***kwargs*)
Version of a `Project`.

clean_build_path()
Clean build path for project version.

Ensure build path is clean for project version. Used to ensure stale build checkouts for each project version are removed.

commit_name
Return the branch name, the tag name or the revision identifier.

The result could be used as ref in a git repo, e.g. for linking to GitHub, Bitbucket or GitLab.

get_build_path()
Return version build path if path exists, otherwise `None`.

get_github_url(*docroot*, *filename*, *source_suffix=u'.rst'*, *action=u'view'*)
Return a GitHub URL for a given filename.

Parameters

- **docroot** – Location of documentation in repository
- **filename** – Name of file
- **source_suffix** – File suffix of documentation format
- **action** – `view` (default) or `edit`

identifier = None
The identifier is the ID for the revision this is version is for. This might be the revision number (e.g. in SVN), or the commit hash (e.g. in Git). If the this version is pointing to a branch, then `identifier` will contain the branch name.

identifier_friendly
Return display friendly identifier.

save(**args*, ***kwargs*)
Add permissions to the Version for all owners on save.

slug = None
The slug is the slugified version of `verbose_name` that can be used in the URL to identify this version in a project. It's also used in the filesystem to determine how the paths for this version are called. It must not be used for any other identifying purposes.

verbose_name = None
This is the actual name that we got for the commit stored in `identifier`. This might be the tag or branch name like `"v1.0.4"`. However this might also hold special version names like `"latest"` and `"stable"`.

class readthedocs.builds.models.**VersionAlias**(**args*, ***kwargs*)
Alias for a `Version`.

readthedocs.builds.urls

URL configuration for builds app.

readthedocs.builds.views

Views for builds app.

readthedocs.doc_builder

readthedocs.doc_builder.base

Base classes for Builders.

class readthedocs.doc_builder.base.**BaseBuilder**(*build_env*, *python_env*, *force=False*)
The Base for all Builders. Defines the API for subclasses.

Expects subclasses to define `old_artifact_path`, which points at the directory where artifacts should be copied from.

build()
Do the actual building of the documentation.

clean(**__)
Clean the path where documentation will be built.

create_index(*extension=u'md'*, **__)
Create an index file if it needs it.

docs_dir(*docs_dir=None*, **__)
Handle creating a custom docs_dir if it doesn't exist.

force(**__)
An optional step to force a build even when nothing has changed.

move(**__)
Move the generated documentation to its artifact directory.

run(**args*, ***kwargs*)
Proxy run to build environment.

readthedocs.doc_builder.environments

Documentation Builder Environments.

class readthedocs.doc_builder.environments.**BuildCommand**(*command*, *cwd=None*, *shell=False*, *environment=None*, *combine_output=True*, *input_data=None*, *build_env=None*, *bin_path=None*, *description=None*, *record_as_success=False*)
Wrap command execution for execution in build environments.

This wraps subprocess commands with some logic to handle exceptions, logging, and setting up the env for the build command.

This acts a mapping of sorts to the API representation of the *readthedocs.builds.models.BuildCommandResult* model.

Parameters

- **command** – string or array of command parameters
- **cwd** – current working path for the command
- **shell** – execute command in shell, default=False
- **environment** (*dict*) – environment variables to add to environment
- **combine_output** – combine stdout/stderr, default=True
- **input_data** (*str*) – data to pass in on stdin
- **build_env** – build environment to use to execute commands
- **bin_path** – binary path to add to PATH resolution
- **description** – a more grokable description of the command being run

get_command()
Flatten command.

run()
Set up subprocess and execute command.

Parameters

- **cmd_input** (*str*) – input to pass to command in STDIN
- **combine_output** – combine STDERR into STDOUT

save()
Save this command and result via the API.

class readthedocs.doc_builder.environments.**DockerBuildCommand**(*command*, *cwd=None*, *shell=False*, *environment=None*, *combine_output=True*, *input_data=None*, *build_env=None*, *bin_path=None*, *description=None*, *record_as_success=False*)
Create a docker container and run a command inside the container.

Build command to execute in docker container

get_wrapped_command()
Escape special bash characters in command to wrap in shell.

In order to set the current working path inside a docker container, we need to wrap the command in a shell call manually. Some characters will be interpreted as shell characters without escaping, such as: `pip install requests<0.8`. This escapes a good majority of those characters.

run()
Execute command in existing Docker container.

Parameters

- **cmd_input** (*str*) – input to pass to command in STDIN
- **combine_output** – combine STDERR into STDOUT

class readthedocs.doc_builder.environments.**LocalBuildEnvironment**(*project=None*, *version=None*, *build=None*, *config=None*, *record=True*, *environment=None*, *update_on_success=True*)

Local execution build environment.

command_class
alias of *BuildCommand*

class readthedocs.doc_builder.environments.**DockerBuildEnvironment**(**args*, ***kwargs*)

Docker build environment, uses docker to contain builds.

If `settings.DOCKER_ENABLE` is true, build documentation inside a docker container, instead of the host system, using this build environment class. The build command creates a docker container from a pre-built image, defined by `settings.DOCKER_IMAGE`. This container is started with a mount to the project's build path under `user_builds` on the host machine, walling off project builds from reading/writing other projects' data.

Parameters **docker_socket** – Override to Docker socket URI

command_class
alias of *DockerBuildCommand*

container_id
Return id of container if it is valid.

container_state()
Get container state.

create_container()
Create docker container.

get_client()
Create Docker client connection.

get_container_host_config()
Create the `host_config` settings for the container.

It mainly generates the proper path bindings between the Docker container and the Host by mounting them with the proper permissions. Besides, it mounts the `GLOBAL_PIP_CACHE` if it's set and we are under `DEBUG`.

The object returned is passed to Docker function `client.create_container`.

update_build_from_container_state()
Update buildenv state from container state.

In the case of the parent command exiting before the exec commands finish and the container is destroyed, or in the case of OOM on the container, set a failure state and error message explaining the failure on the buildenv.

readthedocs.doc_builder.backends

readthedocs.doc_builder.backends.sphinx

Sphinx backend for building docs.

class readthedocs.doc_builder.backends.sphinx.**BaseSphinx**(**args*, ***kwargs*)
The parent for most sphinx builders.

append_conf(**__)
Modify given `conf.py` file from a whitelisted user's project.

get_config_params()
Get configuration parameters to be rendered into the conf file.

class readthedocs.doc_builder.backends.sphinx.**DockerLatexBuildCommand**(*command*, *cwd=None*, *shell=False*, *environment=None*, *combine_output=True*, *input_data=None*, *build_env=None*, *bin_path=None*, *description=None*, *record_as_success=False*)
Ignore LaTeX exit code if there was file output.

class readthedocs.doc_builder.backends.sphinx.**LatexBuildCommand**(*command*, *cwd=None*, *shell=False*, *environment=None*, *combine_output=True*, *input_data=None*, *build_env=None*, *bin_path=None*, *description=None*, *record_as_success=False*)
Ignore LaTeX exit code if there was file output.

class readthedocs.doc_builder.backends.sphinx.**PdfBuilder**(**args*, ***kwargs*)
Builder to generate PDF documentation.

readthedocs.core

readthedocs.core.admin

Django admin interface for core models.

class readthedocs.core.admin.**UserAdminExtra**(*model*, *admin_site*)
Admin configuration for User.

class readthedocs.core.admin.**UserProjectFilter**(*request*, *params*, *model*, *model_admin*)
Filter users based on project properties.

queryset(*request*, *queryset*)
Add filters to queryset filter.

PROJECT_ACTIVE and PROJECT_BUILT look for versions on projects, PROJECT_RECENT looks for projects with builds in the last year

readthedocs.core.forms

Forms for core app.

class readthedocs.core.forms.**FacetField**(*choices=()*, *required=True*, *widget=None*, *label=None*, *initial=None*, *help_text=u''*, **args*, ***kwargs*)
For filtering searches on a facet.

Has validation for the format of facet values.

valid_value(*value*)
Although this is a choice field, no choices need to be supplied.

Instead, we just validate that the value is in the correct format for facet filtering (facet_name:value)

class readthedocs.core.forms.**FacetedSearchForm**(**args*, ***kwargs*)
Supports fetching faceted results with a corresponding query.

facets A list of facet names for which to get facet counts

models Limit the search to one or more models

readthedocs.core.middleware

Middleware for core app.

class readthedocs.core.middleware.**FooterNoSessionMiddleware**
Middleware that doesn't create a session on logged out doc views.

This will reduce the size of our session table drastically.

class readthedocs.core.middleware.**ProxyMiddleware**
Middleware that sets REMOTE_ADDR based on HTTP_X_FORWARDED_FOR, if the.

latter is set. This is useful if you're sitting behind a reverse proxy that causes each request's REMOTE_ADDR to be set to 127.0.0.1. Note that this does NOT validate HTTP_X_FORWARDED_FOR. If you're not behind a reverse proxy that sets HTTP_X_FORWARDED_FOR automatically, do not use this middleware. Anybody can spoof the value of HTTP_X_FORWARDED_FOR, and because this sets REMOTE_ADDR based on HTTP_X_FORWARDED_FOR, that means anybody can "fake" their IP address. Only use this when you can absolutely trust the value of HTTP_X_FORWARDED_FOR.

class readthedocs.core.middleware.**SingleVersionMiddleware**
Reset urlconf for requests for 'single_version' docs.

In settings.MIDDLEWARE_CLASSES, SingleVersionMiddleware must follow after SubdomainMiddleware.

class readthedocs.core.middleware.**SubdomainMiddleware**
Middleware to display docs for non-dashboard domains.

process_request(*request*)
Process requests for unhandled domains.

If the request is not for our PUBLIC_DOMAIN, or if PUBLIC_DOMAIN is not set and the request is for a subdomain on PRODUCTION_DOMAIN, process the request as a request a documentation project.

readthedocs.core.models

Models for the core app.

class readthedocs.core.models.**UserProfile**(**args*, ***kwargs*)
Additional information about a User.

get_contribution_details()
Get the line to put into commits to attribute the author.

Returns a tuple (name, email)

readthedocs.core.views

Core views, including the main homepage,

documentation and header rendering, and server errors.

readthedocs.core.views.**server_error_404**(*request*, *exception*, *template_name='404.html'*)
A simple 404 handler so we get media.

readthedocs.core.views.**server_error_500**(*request*, *template_name='500.html'*)
A simple 500 handler so we get media.

readthedocs.core.management.commands

This is where custom manage.py commands are defined. Rebuild documentation for all projects Clean up stable build paths per project version Import a project's programming language from GitHub

This builds a basic management command that will set a projects language to the most used one in GitHub.

Requires a GITHUB_AUTH_TOKEN to be set in the environment, which should contain a proper GitHub Oauth Token for rate limiting. Resync GitHub project for user Trigger build for project slug Reindex Elastic Search indexes Generate metadata for all projects Update symlinks for projects Build documentation using the API and not hitting a database.

Usage:

```
./manage.py update_api <slug>
```

Custom management command to rebuild documentation for all projects.

Invoked via ./manage.py update_repos.

class readthedocs.core.management.commands.update_repos.**Command**(*stdout=None*, *stderr=None*, *no_color=False*)
Management command for rebuilding documentation on projects

Rebuild documentation for all projects

readthedocs.projects

readthedocs.projects.admin

Django administration interface for `projects.models`

class `readthedocs.projects.admin.`**ImportedFileAdmin**(*model*, *admin_site*)
Admin view for `ImportedFile`

class `readthedocs.projects.admin.`**ProjectAdmin**(*model*, *admin_site*)
Project model admin view.

ban_owner(*request*, *queryset*)
Ban project owner.

This will only ban single owners, because a malicious user could add a user as a co-owner of the project. We don't want to induce and collatoral damage when flagging users.

delete_selected_and_artifacts(*request*, *queryset*)
Remove HTML/etc artifacts from application instances.

Prior to the query delete, broadcast tasks to delete HTML artifacts from application instances.

class `readthedocs.projects.admin.`**ProjectOwnerBannedFilter**(*request*, *params*, *model*, *model_admin*)
Filter for projects with banned owners.

There are problems adding `users__profile__banned` to the `list_filter` attribute, so we'll create a basic filter to capture banned owners.

class `readthedocs.projects.admin.`**ProjectRelationshipInline**(*parent_model*, *admin_site*)
Project inline relationship view for *ProjectAdmin*

model
alias of `ProjectRelationship`

class `readthedocs.projects.admin.`**RedirectInline**(*parent_model*, *admin_site*)
Redirect inline relationship view for *ProjectAdmin*

model
alias of `Redirect`

class `readthedocs.projects.admin.`**VersionInline**(*parent_model*, *admin_site*)
Version inline relationship view for *ProjectAdmin*

model
alias of `Version`

readthedocs.projects.constants

Project constants.

Default values and other various configuration for projects, including available theme names and repository types.

readthedocs.projects.forms

Project forms.

class readthedocs.projects.forms.**BaseVersionsForm**(*data=None*, *files=None*, *auto_id=u'id_%s'*, *prefix=None*, *initial=None*, *error_class=<class 'django.forms.utils.ErrorList'>*, *label_suffix=None*, *empty_permitted=False*, *field_order=None*)

Form for versions page.

save_version(*version*)

Save version if there has been a change, trigger a rebuild.

class readthedocs.projects.forms.**DomainForm**(**args*, ***kwargs*)

Form to configure a custom domain name for a project.

class readthedocs.projects.forms.**DualCheckboxWidget**(*version*, *attrs=None*, *check_test=<type 'bool'>*)

Checkbox with link to the version's built documentation.

class readthedocs.projects.forms.**EmailHookForm**(**args*, ***kwargs*)

Project email notification form.

class readthedocs.projects.forms.**FeatureForm**(**args*, ***kwargs*)

Project feature form for dynamic admin choices.

This form converts the CharField into a ChoiceField on display. The underlying driver won't attempt to do validation on the choices, and so we can dynamically populate this list.

class readthedocs.projects.forms.**IntegrationForm**(**args*, ***kwargs*)

Form to add an integration.

This limits the choices of the integration type to webhook integration types

class readthedocs.projects.forms.**ProjectAdvancedForm**(**args*, ***kwargs*)

Advanced project option form.

class readthedocs.projects.forms.**ProjectAdvertisingForm**(**args*, ***kwargs*)

Project promotion opt-out form.

class readthedocs.projects.forms.**ProjectBackendForm**(*data=None*, *files=None*, *auto_id=u'id_%s'*, *prefix=None*, *initial=None*, *error_class=<class 'django.forms.utils.ErrorList'>*, *label_suffix=None*, *empty_permitted=False*, *field_order=None*)

Get the import backend.

class readthedocs.projects.forms.**ProjectBasicsForm**(**args*, ***kwargs*)

Form for basic project fields.

save(*commit=True*)

Add remote repository relationship to the project instance.

class readthedocs.projects.forms.**ProjectExtraForm**(**args*, ***kwargs*)

Additional project information form.

class readthedocs.projects.forms.**ProjectForm**(**args*, ***kwargs*)

Project form.

Parameters **user** – If provided, add this user as a project user on save

class readthedocs.projects.forms.**ProjectRelationshipForm**(**args*, ***kwargs*)
Form to add/update project relationships.

get_subproject_queryset()
Return scrubbed subproject choice queryset.

This removes projects that are either already a subproject of another project, or are a superproject, as neither case is supported.

class readthedocs.projects.forms.**ProjectTriggerBuildMixin**
Mixin to trigger build on form save.

This should be replaced with signals instead of calling trigger_build explicitly.

save(*commit=True*)
Trigger build on commit save.

class readthedocs.projects.forms.**RedirectForm**(**args*, ***kwargs*)
Form for project redirects.

class readthedocs.projects.forms.**TranslationForm**(**args*, ***kwargs*)
Project translation form.

class readthedocs.projects.forms.**UserForm**(**args*, ***kwargs*)
Project user association form.

class readthedocs.projects.forms.**WebHookForm**(**args*, ***kwargs*)
Project webhook form.

readthedocs.projects.forms.**build_upload_html_form**(*project*)
Upload HTML form with list of versions to upload HTML for.

readthedocs.projects.forms.**build_versions_form**(*project*)
Versions form with a list of versions and version privacy levels.

readthedocs.projects.models

Project models.

class readthedocs.projects.models.**APIProject**(**args*, ***kwargs*)
Project proxy model for API data deserialization.

This replaces the pattern where API data was deserialized into a mocked **:py:cls:‘Project‘** object. This pattern was confusing, as it was not explicit as to what form of object you were working with – API backed or database backed.

This model preserves the Project model methods, allowing for overrides on model field differences. This model pattern will generally only be used on builder instances, where we are interacting solely with API data.

class readthedocs.projects.models.**Domain**(**args*, ***kwargs*)
A custom domain name for a project.

class readthedocs.projects.models.**EmailHook**(*id*, *project*, *email*)

class readthedocs.projects.models.**Feature**(**args*, ***kwargs*)
Project feature flags.

Features should generally be added here as choices, however features may also be added dynamically from a signal in other packages. Features can be added by external packages with the use of signals:

```
@receiver(pre_init, sender=Feature)
def add_features(sender, **kwargs):
    sender.FEATURES += (('blah', 'BLAH'),)
```

The FeatureForm will grab the updated list on instantiation.

get_feature_display()
Implement display name field for fake ChoiceField.

Because the field is not a ChoiceField here, we need to manually implement this behavior.

class readthedocs.projects.models.**ImportedFile**(**args*, ***kwargs*)
Imported files model.

This tracks files that are output from documentation builds, useful for things like CDN invalidation.

class readthedocs.projects.models.**Project**(**args*, ***kwargs*)
Project model.

add_comment(*version_slug*, *page*, *content_hash*, *commit*, *user*, *text*)
Add comment to node.

Parameters

- **version_slug** – Version slug to use for node lookup
- **page** – Page to attach comment to
- **content_hash** – Hash of content to apply comment to
- **commit** – Commit that updated comment
- **user** – `User` instance that created comment
- **text** – Comment text

add_node(*content_hash*, *page*, *version*, *commit*)
Add comment node.

Parameters

- **content_hash** – Hash of node content
- **page** – Doc page for node
- **version** (*str*) – Slug for project version to apply node to
- **commit** (*str*) – Commit that node was updated in

all_active_versions()
Get queryset with all active versions.

Note: This is a temporary workaround for activate_versions filtering out things that were active, but failed to build

---

Returns `Version` queryset

artifact_path(*type_*, *version='latest'*)
The path to the build html docs in the project.

conf_file(*version='latest'*)
Find a `conf.py` file in the project checkout.

find(*filename*, *version*)
Find files inside the project's `doc` path.

Parameters

- **filename** – Filename to search for in project checkout
- **version** – Version instance to set version checkout path

full_build_path(*version='latest'*)
The path to the build html docs in the project.

full_dash_path(*version='latest'*)
The path to the build dash docs in the project.

full_doc_path(*version='latest'*)
The path to the documentation root in the project.

full_epub_path(*version='latest'*)
The path to the build epub docs in the project.

full_find(*filename*, *version*)
Find files inside a project's checkout path.

Parameters

- **filename** – Filename to search for in project checkout
- **version** – Version instance to set version checkout path

full_json_path(*version='latest'*)
The path to the build json docs in the project.

full_latex_path(*version='latest'*)
The path to the build LaTeX docs in the project.

full_man_path(*version='latest'*)
The path to the build man docs in the project.

full_singlehtml_path(*version='latest'*)
The path to the build singlehtml docs in the project.

get_default_branch()
Get the version representing 'latest'.

get_default_version()
Get the default version (slug).

Returns self.default_version if the version with that slug actually exists (is built and published). Otherwise returns 'latest'.

get_docs_url(*version_slug=None*, *lang_slug=None*, *private=None*)
Return a URL for the docs.

Always use http for now, to avoid content warnings.

get_feature_value(*feature*, *positive*, *negative*)
Look up project feature, return corresponding value.

If a project has a feature, return `positive`, otherwise return `negative`

get_latest_build(*finished=True*)
Get latest build for project.

Parameters **finished** – Return only builds that are in a finished state

get_production_media_path(*type_*, *version_slug*, *include_file=True*)
Used to see if these files exist so we can offer them for download.

Parameters

- **type** – Media content type, ie - 'pdf', 'zip'
- **version_slug** – Project version slug for lookup
- **include_file** (*bool*) – Include file name in return

Returns Full path to media file or path

get_production_media_url(*type_*, *version_slug*, *full_path=True*)
Get the URL for downloading a specific media file.

get_subproject_urls()
List subproject URLs.

This is used in search result linking

has_feature(*feature_id*)
Does project have existing feature flag.

If the feature has a historical True value before the feature was added, we consider the project to have the flag. This is used for deprecating a feature or changing behavior for new projects

is_type_mkdocs
Is project type Mkdocs.

is_type_sphinx
Is project type Sphinx.

pip_cache_path
Path to pip cache.

rtd_build_path(*version='latest'*)
The destination path where the built docs are copied.

static_metadata_path()
The path to the static metadata JSON settings file.

subdomain()
Get project subdomain from resolver.

supported_versions()
Get the list of supported versions.

Returns List of version strings.

translations_symlink_path(*language=None*)
Path in the doc_path that we symlink translations.

update_stable_version()
Returns the version that was promoted to be the new stable version.

Return `None` if no update was mode or if there is no version on the project that can be considered stable.

vcs_repo(*version='latest'*, *environment=None*)
Return a Backend object for this project able to handle VCS commands.

Parameters

- **environment** (*doc_builder.environments.BuildEnvironment*) – environment to run the commands

- **version** (*str*) – version slug for the backend (LATEST by default)

class readthedocs.projects.models.**ProjectRelationship**(**args*, ***kwargs*)
Project to project relationship.

This is used for subprojects

class readthedocs.projects.models.**WebHook**(*id*, *project*, *url*)

readthedocs.projects.search_indexes

Project search indexes.

class readthedocs.projects.search_indexes.**ImportedFileIndex**
Search index for imported files.

index_queryset(*using=None*)
Used when the entire index for model is updated.

prepare_text(*obj*)
Prepare the text of the html file.

This only works on machines that have the html files for the projects checked out.

class readthedocs.projects.search_indexes.**ProjectIndex**
Project search index.

index_queryset(*using=None*)
Used when the entire index for model is updated.

readthedocs.projects.tasks

Tasks related to projects.

This includes fetching repository code, cleaning conf.py files, and rebuilding documentation.

class readthedocs.projects.tasks.**SyncRepositoryMixin**
Mixin that handles the VCS sync/update.

static get_version(*project=None*, *version_pk=None*)
Retrieve version data from the API.

Parameters

- **project** (projects.models.Project) – project object to sync
- **version_pk** (*int*) – version pk to sync

Returns a data-complete version object

Return type *builds.models.APIVersion*

sync_repo()
Update the project's repository and hit sync_versions API.

class readthedocs.projects.tasks.**SyncRepositoryTask**
Entry point to synchronize the VCS documentation.

run(*version_pk*)
Run the VCS synchronization.

Parameters **version_pk** (*int*) – version pk to sync

Returns whether or not the task ended successfully

Return type bool

class readthedocs.projects.tasks.**UpdateDocsTask**(*build_env=None*, *python_env=None*, *config=None*, *force=False*, *search=True*, *localmedia=True*, *build=None*, *project=None*, *version=None*)

The main entry point for updating documentation.

It handles all of the logic around whether a project is imported, we created it or a webhook is received. Then it will sync the repository and build the html docs if needed.

build_docs()

Wrapper to all build functions.

Executes the necessary builds for this task and returns whether the build was successful or not.

Returns Build outcomes with keys for html, search, localmedia, pdf, and epub

Return type dict

build_docs_class(*builder_class*)

Build docs with additional doc backends.

These steps are not necessarily required for the build to halt, so we only raise a warning exception here. A hard error will halt the build process.

build_docs_epub()

Build ePub docs.

build_docs_html()

Build HTML docs.

build_docs_localmedia()

Get local media files with separate build.

build_docs_pdf()

Build PDF docs.

build_docs_search()

Build search data with separate build.

static get_build(*build_pk*)

Retrieve build object from API.

Parameters **build_pk** – Build primary key

get_env_vars()

Get bash environment variables used for all builder commands.

static get_project(*project_pk*)

Get project from API.

run(*pk*, *version_pk=None*, *build_pk=None*, *record=True*, *docker=False*, *search=True*, *force=False*, *localmedia=True*, ***__)

Run a documentation sync n' build.

This is fully wrapped in exception handling to account for a number of failure cases. We first run a few commands in a local build environment, but do not report on environment success. This avoids a flicker on the build output page where the build is marked as finished in between the local environment steps and the docker build steps.

If a failure is raised, or the build is not successful, return `False`, otherwise, `True`.

Unhandled exceptions raise a generic user facing error, which directs the user to bug us. It is therefore a benefit to have as few unhandled errors as possible.

Parameters

- **int** (*build_pk*) – Project id
- **int** – Project Version id (latest if None)
- **int** – Build id (if None, commands are not recorded)
- **bool** (*localmedia*) – record a build object in the database
- **bool** – use docker to build the project
- **bool** – update search
- **bool** – force Sphinx build
- **bool** – update localmedia

Returns whether build was successful or not

Return type bool

run_build(*docker=False*, *record=True*)
Build the docs in an environment.

If `docker` is True, or Docker is enabled by the settings.DOCKER_ENABLE setting, then build in a Docker environment. Otherwise build locally.

run_setup(*record=True*)
Run setup in the local environment.

Return True if successful.

send_notifications()
Send notifications on build failure.

set_valid_clone()
Mark on the project that it has been cloned properly.

setup_python_environment()
Build the virtualenv and install the project into it.

Always build projects with a virtualenv.

Parameters **build_env** – Build environment to pass commands and execution through.

setup_vcs()
Update the checkout of the repo to make sure it's the latest.

This also syncs versions in the DB.

Parameters **build_env** – Build environment

update_app_instances(*html=False*, *localmedia=False*, *search=False*, *pdf=False*, *epub=False*)
Update application instances with build artifacts.

This triggers updates across application instances for html, pdf, epub, downloads, and search. Tasks are broadcast to all web servers from here.

update_documentation_type()
Force Sphinx for 'auto' documentation type.

This used to determine the type and automatically set the documentation type to Sphinx for rST and Mkdocs for markdown. It now just forces Sphinx, due to markdown support.

readthedocs.projects.tasks.**email_notification**(*version*, *build*, *email*)
Send email notifications for build failure.

Parameters

- **version** – Version instance that failed
- **build** – Build instance that failed
- **email** – Email recipient address

readthedocs.projects.tasks.**webhook_notification**(*version*, *build*, *hook_url*)
Send webhook notification for project webhook.

Parameters

- **version** – Version instance to send hook for
- **build** – Build instance that failed
- **hook_url** – Hook URL to send to

readthedocs.projects.utils

Utility functions used by projects.

readthedocs.projects.utils.**find_file**(*filename*)
Recursively find matching file from the current working path.

Parameters **file** – Filename to match

Returns A list of matching filenames.

readthedocs.projects.utils.**run**(**commands*)
Run one or more commands.

Each argument in `commands` can be passed as a string or as a list. Passing as a list is the preferred method, as space escaping is more explicit and it avoids the need for executing anything in a shell.

If more than one command is given, then this is equivalent to chaining them together with `&&`; if all commands succeed, then `(status, out, err)` will represent the last successful command. If one command failed, then `(status, out, err)` will represent the failed command.

Returns `(status, out, err)`

readthedocs.projects.utils.**safe_write**(*filename*, *contents*)
Normalize and write to filename.

Write `contents` to the given `filename`. If the filename's directory does not exist, it is created. Contents are written as UTF-8, ignoring any characters that cannot be encoded as UTF-8.

Parameters

- **filename** – Filename to write to
- **contents** – File contents to write to file

readthedocs.projects.views

readthedocs.projects.views.public

Public project views.

class readthedocs.projects.views.public.**ProjectDetailView**(***kwargs*)
Display project onboard steps.

model
alias of Project

class readthedocs.projects.views.public.**ProjectIndex**(***kwargs*)
List view of public Project instances.

model
alias of Project

readthedocs.projects.views.public.**elastic_project_search**(*request*, *project_slug*)
Use elastic search to search in a project.

readthedocs.projects.views.public.**file_autocomplete**(*request*, *project_slug*)
Return a json list of file names.

readthedocs.projects.views.public.**project_analytics**(*request*, *project_slug*)
Have a analytics API placeholder.

readthedocs.projects.views.public.**project_badge**(*request*, **args*, ***kwargs*)
Return a sweet badge for the project.

readthedocs.projects.views.public.**project_download_media**(*request*, *project_slug*, *type_*, *version_slug*)
Download a specific piece of media.

Perform an auth check if serving in private mode.

> **Warning:** This is linked directly from the HTML pages. It should only care about the Version permissions, not the actual Project permissions.

readthedocs.projects.views.public.**project_downloads**(*request*, *project_slug*)
A detail view for a project with various dataz.

readthedocs.projects.views.public.**project_embed**(*request*, *project_slug*)
Have a content API placeholder.

readthedocs.projects.views.public.**project_index**(*request*, **args*, ***kwargs*)
List view of public Project instances.

readthedocs.projects.views.public.**project_versions**(*request*, *project_slug*)
Project version list view.

Shows the available versions and lets the user choose which ones to build.

readthedocs.projects.views.public.**search_autocomplete**(*request*)
Return a json list of project names.

readthedocs.projects.views.public.**version_autocomplete**(*request*, *project_slug*)
Return a json list of version names.

readthedocs.projects.views.private

Project views for authenticated users.

class readthedocs.projects.views.private.**ImportDemoView**(***kwargs*)
View to pass request on to import form to import demo project.

form_class
alias of ProjectBasicsForm

get(*request*, **args*, ***kwargs*)
Process link request as a form post to the project import form.

get_form_data()
Get form data to post to import form.

get_form_kwargs()
Form kwargs passed in during instantiation.

class readthedocs.projects.views.private.**ImportView**(***kwargs*)
On GET, show the source an import view, on POST, mock out a wizard.

If we are accepting POST data, use the fields to seed the initial data in *ImportWizardView*. The import templates will redirect the form to /dashboard/import

get(*request*, **args*, ***kwargs*)
Display list of repositories to import.

Adds a warning to the listing if any of the accounts connected for the user are not supported accounts.

wizard_class
alias of *ImportWizardView*

class readthedocs.projects.views.private.**ImportWizardView**(***kwargs*)
Project import wizard.

done(*form_list*, ***kwargs*)
Save form data as object instance.

Don't save form data directly, instead bypass documentation building and other side effects for now, by signalling a save without commit. Then, finish by added the members to the project and saving.

get_form_kwargs(*step=None*)
Get args to pass into form instantiation.

get_template_names()
Return template names based on step name.

is_advanced()
Determine if the user selected the show advanced field.

class readthedocs.projects.views.private.**IntegrationMixin**
Project external service mixin for listing webhook objects.

form_class
alias of IntegrationForm

get_integration()
Return project integration determined by url kwarg.

model
alias of Integration

class readthedocs.projects.views.private.**IntegrationWebhookSync**(***kwargs*)
Resync a project webhook.

The signal will add a success/failure message on the request.

class readthedocs.projects.views.private.**ProjectDashboard**(***kwargs*)
Project dashboard.

model
alias of Project

readthedocs.projects.views.private.**edit_alias**(*request*, **args*, ***kwargs*)
Edit project alias form view.

readthedocs.projects.views.private.**project_delete**(*request*, **args*, ***kwargs*)
Project delete confirmation view.

Make a project as deleted on POST, otherwise show a form asking for confirmation of delete.

readthedocs.projects.views.private.**project_manage**(*request*, **args*, ***kwargs*)
Project management view.

Where you will have links to edit the projects' configuration, edit the files associated with that project, etc.

Now redirects to the normal /projects/<slug> view.

readthedocs.projects.views.private.**project_notifications**(*request*, **args*, ***kwargs*)
Project notification view and form view.

readthedocs.projects.views.private.**project_notifications_delete**(*request*, **args*, ***kwargs*)
Project notifications delete confirmation view.

readthedocs.projects.views.private.**project_redirects**(*request*, **args*, ***kwargs*)
Project redirects view and form view.

readthedocs.projects.views.private.**project_redirects_delete**(*request*, **args*, ***kwargs*)
Project redirect delete view.

readthedocs.projects.views.private.**project_translations**(*request*, **args*, ***kwargs*)
Project translations view and form view.

readthedocs.projects.views.private.**project_users**(*request*, **args*, ***kwargs*)
Project users view and form view.

readthedocs.projects.views.private.**project_version_delete_html**(*request*, **args*, ***kwargs*)
Project version 'delete' HTML.

This marks a version as not built

readthedocs.projects.views.private.**project_version_detail**(*request*, **args*, ***kwargs*)
Project version detail page.

readthedocs.projects.views.private.**project_versions**(*request*, **args*, ***kwargs*)
Project versions view.

Shows the available versions and lets the user choose which ones he would like to have built.

readthedocs.vcs_support

readthedocs.vcs_support.base

Base classes for VCS backends.

class readthedocs.vcs_support.base.**BaseVCS**(*project*, *version_slug*, *environment=None*, ***kwargs*)

Base for VCS Classes.

VCS commands are ran inside a LocalEnvironment.

branches
Returns a list of VCSVersion objects.

See VCSVersion for more information.

checkout(*identifier=None*)
Set the state to the given identifier.

If identifier is None, checkout to the latest revision.

The type and format of identifier may change from VCS to VCS, so each backend is responsible to understand it's identifiers.

commit
Returns a string representing the current commit.

make_clean_working_dir()
Ensures that the working dir exists and is empty

tags
Returns a list of VCSVersion objects.

See VCSVersion for more information.

update()
Update a local copy of the repository in self.working_dir.

If self.working_dir is already a valid local copy of the repository, update the repository, else create a new local copy of the repository.

class readthedocs.vcs_support.base.**VCSVersion**(*repository*, *identifier*, *verbose_name*)
Represents a Version (tag or branch) in a VCS.

This class should only be instantiated in BaseVCS subclasses.

It can act as a context manager to temporarily switch to this tag (eg to build docs for this tag).

CHAPTER 46

Read the Docs Business Features

Note: These features are for our new business offering, readthedocs.com.

All of the other features outlined in these docs work on both sites. Things inside this section are specific to our business offering.

The largest feature that is different is that documentation on readthedocs.com is **private**. If you have private code that you want documentation for, this is our solution.

Organizations

Organizations allow you to segment who has access to what projects in your company. Your company will be represented as an Organization, let's use ACME Corporation as our example.

ACME has a few people inside their organization, some who need full access and some who just need access to one project.

Member Types

- **Owners** – Get full access to both view and edit the Organization and all Projects
- **Members** – Get access to a subset of the Organization projects
- **Teams** – Where you give members access to a set of projects.

The best way to think about this relationship is:

Owners will create *Teams* to assign permissions to all *Members*.

Team Types

You can create two types of Teams:

- **Admins** – These teams have full access to administer the projects in the team. They are allowed to change all of the settings, set notifications, and perform any action under the **Admin** tab.
- **Read Only** – These teams are only able to read and search inside the documents.

Example

ACME would set up *Owners* of their organization, for example Frank Roadrunner would be an owner. He has full access to the organization and all projects.

Wile E. Coyote is a contractor, and will just have access to the new project Road Builder.

Roadrunner would set up a *Team* called *Contractors*. That team would have *Read Only* access to the *Road Builder* project. Then he would add *Wile E. Coyote* to the team. This would give him access to just this one project inside the organization.

Sharing

Note: This feature only exists on our Business offering at readthedocs.com.

You can share your project with users outside of your company. This works by sending them a link, which will allow them to view a specific project inside your company.

Enabling

- Go into your *Project Admin* page and to the *Sharing* link.
- Under the *Add Token* heading, add a *Description* so you remember who you're sharing it with.
- Click *Share* to create.
- Copy the link that is generated, and give that to the person who you want to give access.

Note: You can always revoke access in the same panel.

Effects

Once the person you send the link to clicks the link, they will have access to view your project. It will only work for the specific browser that they click the link from.

Warning: They will be able to share this token with other people, so only share with people you trust. We only let sharing links be activated **five** times to prevent abuse.

Analytics

Note: These features are still being developed, and aren't deployed yet.

Analytics lets you see *who* is viewing *which* documents. This allows you to understand how your documentation is being used, so you can focus on expanding and updating parts people are reading most.

Viewing

Each project page has a listing of the number of views that it has seen. You can click through here to inspect more information about who is viewing, and when they are looking at things.

You can also view your Analytics data in your documentation pages. There is a button in the Read the Docs flyout what will overlay analytics information. This will let you understand how users are using docs, in context of the actual documentation.

CHAPTER 47

Info about custom installs

Read the Docs is open source, which means you can run your own version of it. There are many reasons to do this, the main one being if you want a private instance. If you have to keep everything behind a firewall or VPN, this is for you.

Warning: Read the Docs developers do not support custom installs of our software. These documents are maintained by the community, and might not be up to date.

Customizing your install

Read the Docs has a lot of *Interesting Settings* that help customize your install. This document will outline some of the more useful ways that these can be combined.

Have a local settings file

If you put a file named `local_settings.py` in the `readthedocs/settings` directory, it will override settings available in the base install.

Adding your own title to pages

This requires 2 parts of setup. First, you need to add a custom `TEMPLATE_DIRS` setting that points at your template overrides. Then, in those template overrides you have to insert your logo where the normal RTD logo goes.

Note: This works for any setting you wish to change.

Example `local_settings.py`:

```
import os

# Directory that the project lives in, aka ../..
SITE_ROOT = '/'.join(os.path.dirname(__file__).split('/')[0:-2])

TEMPLATE_DIRS = (
    "%s/var/custom_templates/" % SITE_ROOT, # Your custom template directory, before␣
↪the RTD one to override it.
    '%s/readthedocs/templates/' % SITE_ROOT, # Default RTD template dir
)
```

Example `base.html` in your template overrides:

```
{% extends "/home/docs/checkouts/readthedocs.org/readthedocs/templates/base.html" %}
{% load i18n %}

{% block branding %}{% trans "My sweet site" %} {% endblock %}
```

You can of course override any block in the template. If there is something that you would like to be able to customize, but isn't currently in a block, please submit an issue.

Local VM Install

Assumptions and Prerequisites

- Debian VM provisioned with python 2.7.x
- All python dependencies and setup tools are installed

```
$ sudo apt-get install python-setuptools
$ sudo apt-get install build-essential
$ sudo apt-get install python-dev
$ sudo apt-get install libevent-dev
$ sudo easy_install pip
```

- Git

```
$ sudo apt-get install git
```

- Git repo is `git.corp.company.com:git/docs/documentation.git`
- Source documents are in `../docs/source`
- Sphinx

```
$ sudo pip install sphinx
```

Note: Not using sudo may prevent access. "error: could not create '/usr/local/lib/python2.7/dist-packages/markupsafe': Permission denied"

Local RTD Setup

1. Install RTD.

To host your documentation on a local RTD installation, set it up in your VM.

```
$ mkdir checkouts
$ cd checkouts
$ git clone https://github.com/rtfd/readthedocs.org.git
$ cd readthedocs.org
$ sudo pip install -r requirements.txt
```

Possible Error and Resolution

Error: `error: command 'gcc' failed with exit status 1`

Resolution: Run the following commands.

```
$ sudo apt-get update
$ sudo apt-get install python2.7-dev tk8.5 tcl8.5 tk8.5-dev tcl8.5-dev libxml2-devel␣
↪libxslt-devel
$ sudo apt-get build-dep python-imaging --fix-missing
```

On Debian 8 (jessie) the command is slightly different

```
$ sudo apt-get update
$ sudo apt-get install python2.7-dev tk8.5 tcl8.5 tk8.5-dev tcl8.5-dev libxml2-dev␣
↪libxslt-dev
$ sudo apt-get build-dep python-imaging --fix-missing
```

Also don't forget to re-run the dependency installation

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

2. Configure the RTD Server and Superuser.

1. Run the following commands.

   ```
   $ ./manage.py migrate
   $ ./manage.py createsuperuser
   ```

2. This will prompt you to create a superuser account for Django. Enter appropriate details. For example:

   ```
   Username: monami.b
   Email address: monami.b@email.com
   Password: pa$$word
   ```

3. RTD Server Administration.

Navigate to the `../checkouts/readthedocs.org` folder in your VM and run the following command.

```
$ ./manage.py runserver [VM IP ADDRESS]:8000
$ curl -i http://[VM IP ADDRESS]:8000
```

You should now be able to log into the admin interface from any PC in your LAN at `http://[VM IP ADDRESS]:8000/admin` using the superuser account created in django.

Go to the dashboard at `http://[VM IP ADDRESS]:8000/dashboard` and follow these steps:

1. Point the repository to your corporate Git project where the documentation source is checked in. Example: git.corp.company.com:/git/docs/documentation.git

2. Clone the documentation sources from Git in the VM.

3. Navigate to the root path for documentation.

4. Run the following Sphinx commands.

```
$ make html
```

This generates the HTML documentation site using the default Sphinx theme. Verify the output in your local documentation folder under `../build/html`

Possible Error and Resolution

Error: Couldn't access Git Corp from VM.

Resolution: The primary access may be set from your base PC/laptop. You will need to configure your RSA keys in the VM.

Workaround-1

1. In your machine, navigate to the `.ssh` folder.

```
$ cd .ssh/
$ cat id_rsa
```

2. Copy the entire Private Key.

3. Now, SSH to the VM.

4. Open the `id_rsa` file in the VM.

```
$ vim /home/<username>/.ssh/id_rsa
```

5. Paste the RSA key copied from your machine and save file (`Esc. :wq!`).

Workaround 2

SSH to the VM using the `-A` directive.

```
$ ssh document-vm -A
```

This provides all permissions for that particular remote session, which are revoked when you logout.

4. Build Documentation on Local RTD Instance.

Log into `http://[VM IP ADDRESS]:[PORT]` using the django superuser creds and follow these steps.

For a new project

1. Select **<username> > Add Project** from the user menu.
2. Click **Manually Import Project**.
3. Provide the following information in the **Project Details** page:
   - **Name**: Appropriate name for the documentation project. For example – API Docs Project
   - **Repository URL**: URL to the documentation project. For example - git.corp.company.com:/git/docs/documentation.git
   - **Repository Type**: Git
4. Select the **Edit advanced project options** checkbox.
5. Click **Next**.

For an existing project

1. Select **<username> > Projects** from the user menu.
2. Select the relevant project from the **Projects** list.
3. Select latest from the **Build a version** dropdown.
4. Click **Build**. This will take you to the Builds tab where the progress status is displayed. This may take some time.

Tips

- If the installation doesn't work on VM using your login/LDAP credentials, try running the operations as root (su).

CHAPTER 48

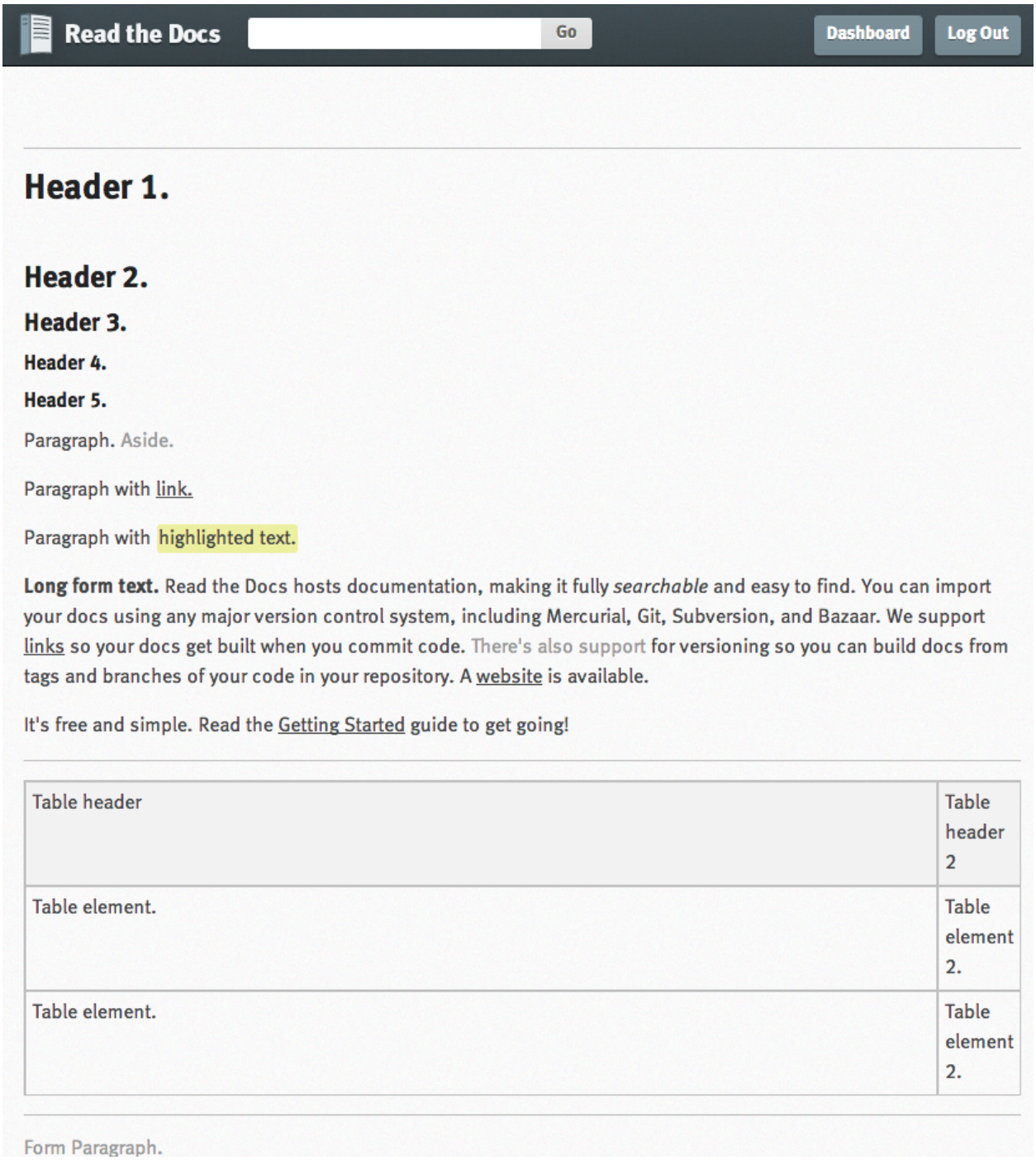
Designing Read the Docs

So you're thinking of contributing some of your time and design skills to Read the Docs? That's **awesome**. This document will lead you through a few features available to ease the process of working with Read the Doc's CSS and static assets.

To start, you should follow the *Installation* instructions to get a working copy of the Read the Docs repository locally.

Style Catalog

Once you have RTD running locally, you can open `http://localhost:8000/style-catalog/` for a quick overview of the currently available styles.



This way you can quickly get started writing HTML – or if you're modifying existing styles you can get a quick idea of how things will change site-wide.

Typekit Fonts

RTD uses FF Meta via TypeKit to render most display and body text.

To make this work locally, you can register a free TypeKit account and create a site profile for `localhost:8000`

that includes the linked font.

Readthedocs.org Changes

Styles for the primary RTD site are located in `media/css` directory.

These styles only affect the primary site – **not** any of the generated documentation using the default RTD style.

Sphinx Template Changes

Styles for generated documentation are located in `readthedocs/templates/sphinx/_static/rtd.css`

Of note, projects will retain the version of that file they were last built with – so if you're editing that file and not seeing any changes to your local built documentation, you need to rebuild your example project.

Contributing

Contributions should follow the *Contributing to Read the Docs* guidelines where applicable – ideally you'll create a pull request against the Read the Docs GitHub project from your forked repo and include a brief description of what you added / removed / changed, as well as an attached image (you can just take a screenshot and drop it into the PR creation form) of the effects of your changes.

There's not a hard browser range, but your design changes should work reasonably well across all major browsers, IE8+ – that's not to say it needs to be pixel-perfect in older browsers! Just avoid making changes that render older browsers utterly unusable (or provide a sane fallback).

HTTP Routing Table

/api

Python Module Index

r

Index

D

E

F

G

H

I

L

M

P

Q

R

S

T

U

V

W