
Read The Docs Documentation

Release 1.0

Eric Holscher, Charlie Leifer, Bobby Grace

August 19, 2016

1	Getting Started	3
1.1	Write Your Docs	3
1.2	Import Your Docs	4
2	Versions	5
2.1	How we envision versions working	5
2.2	Redirects on root URLs	5
3	Build Process	7
3.1	How we build documentation	7
3.2	Understanding what's going on	8
3.3	Builder Responsibility	8
3.4	Packages installed in the build environment	8
3.5	Writing your own builder	9
3.6	Deleting a stale or broken build environment	9
4	Read the Docs features	11
4.1	GitHub and Bitbucket Integration	11
4.2	Auto-updating	11
4.3	Internationalization	11
4.4	Canonical URLs	11
4.5	Versions	11
4.6	Version Control Support Matrix	12
4.7	PDF Generation	12
4.8	Search	12
4.9	Alternate Domains	12
5	Support	13
5.1	Usage Questions	13
5.2	Community Support	13
5.3	Commercial Support	14
6	Frequently Asked Questions	15
6.1	My project isn't building with autodoc	15
6.2	How do I change my slug (the URL your docs are served at)?	15
6.3	How do I change behavior for Read the Docs?	15
6.4	I get import errors on libraries that depend on C modules	16
6.5	Client Error 401 when building documentation	16
6.6	Deleting a stale or broken build environment	16

6.7	How do I host multiple projects on one CNAME?	17
6.8	Where do I need to put my docs for RTD to find it?	17
6.9	I want to use the Blue/Default Sphinx theme	17
6.10	I want to use the Read the Docs theme locally	17
6.11	Image scaling doesn't work in my documentation	17
6.12	I want comments in my docs	17
6.13	How do I support multiple languages of documentation?	18
6.14	Do I need to be whitelisted?	18
6.15	Does Read The Docs work well with "legible" docstrings?	18
6.16	Can I document a python package that is not at the root of my repository?	18
7	Read the Docs YAML Config	19
7.1	Supported Settings	19
8	Guides	21
8.1	Enabling Build Notifications	21
8.2	Enabling Google Analytics on your Project	21
9	Webhooks	23
9.1	GitHub	23
9.2	Bitbucket	23
9.3	Others	24
10	Badges	25
10.1	Status Badges	25
10.2	Project Pages	25
10.3	Style	25
11	Alternate Domains	27
11.1	Subdomain Support	27
11.2	CNAME Support	27
11.3	CNAME SSL	28
11.4	rtfd.org	28
12	Localization of Documentation	29
12.1	Single project in another language	29
12.2	Project with multiple translations	29
13	VCS Integration	31
13.1	GitHub	31
13.2	Bitbucket	31
14	Conda Support	33
14.1	Activating Conda	33
14.2	Custom Installs	33
15	Canonical URLs	35
15.1	Example	35
15.2	Enabling	35
15.3	Implementation	35
15.4	Links	35
16	Single Version Documentation	37
16.1	Enabling	37
16.2	Effects	37

17 Privacy Levels	39
17.1 Understanding the Privacy Levels	39
17.2 Project Objects	40
17.3 Version Objects	40
18 User-defined Redirects	41
18.1 Prefix Redirects	41
18.2 Page Redirects	41
18.3 Exact Redirects	42
18.4 Sphinx Redirects	42
18.5 Implementation	42
19 Automatic Redirects	43
19.1 Root URL	43
19.2 Supported Top-Level Redirects	43
19.3 Redirecting to a Page	44
20 Content Embedding	45
21 Read the Docs Team	47
21.1 Support Team	47
21.2 Operations Team	48
21.3 Development Team	48
22 Installation	51
22.1 What's available	52
23 Changelog	55
23.1 July 23, 2015	55
24 Contributing to Read the Docs	57
24.1 Contributing to development	57
24.2 Triaging tickets	57
24.3 Helping on translations	59
25 Testing	61
25.1 Continuous Integration	61
26 Architecture	63
26.1 Diagram	63
27 Development Standards	65
27.1 Front End Development	65
28 Build Environments	69
28.1 Setup	69
28.2 Configuration	69
29 How we use symlinks	71
29.1 Nginx	71
29.2 Subdomains	71
29.3 CNAMEs	72
30 Interesting Settings	73
30.1 SLUMBER_USERNAME	73
30.2 SLUMBER_PASSWORD	73

30.3	USE_SUBDOMAIN	73
30.4	PRODUCTION_DOMAIN	73
30.5	MULTIPLE_APP_SERVERS	73
30.6	DEFAULT_PRIVACY_LEVEL	74
30.7	INDEX_ONLY_LATEST	74
30.8	DOCUMENT_PYQUERY_PATH	74
30.9	USE_PIP_INSTALL	74
30.10	PUBLIC_DOMAIN	74
30.11	ALLOW_ADMIN	74
31	Internationalization	75
31.1	Making Strings Localizable	75
31.2	Strings in Templates	76
31.3	Strings in Python	77
32	Administrative Tasks	79
32.1	Updating Localization Files	79
32.2	Compiling to MO	79
32.3	Transifex Integration	79
33	Overview of issue labels	81
34	Read the Docs Business Features	83
34.1	Organizations	83
34.2	Sharing	84
34.3	Analytics	84
35	Info about custom installs	87
35.1	Customizing your install	87
35.2	Local VM Install	88
36	Designing Read the Docs	93
36.1	Style Catalog	93
36.2	Typekit Fonts	94
36.3	Readthedocs.org Changes	95
36.4	Sphinx Template Changes	95
36.5	Contributing	95
37	Read the Docs Theme	97
37.1	Contributing to the theme	98
37.2	How the Table of Contents builds	98
37.3	Other style notes	98
38	Read the Docs Open Source Philosophy	99
38.1	Official Support	99
38.2	Unsupported	99
38.3	Rationale	99
39	Sponsors of Read the Docs	101
39.1	Current sponsors	101
39.2	Past sponsors	101
39.3	Sponsorship Information	101
40	Talks about Read the Docs	103

41 Configuration of the production servers	105
41.1 Elastic Search Setup	105
HTTP Routing Table	107

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*
- *Operations Documentation*

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*.

1.1 Write Your Docs

You have two options for formatting your documentation:

- *In reStructuredText*
- *In Markdown*

1.1.1 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.

1.1.2 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.

1.2 Import Your Docs

Sign up for an account on RTD, then log in. Visit your [dashboard](#) and click [Import](#) to add your project to the site. Fill in the name and description, then specify where your repository is located. This is normally the URL or path name you'd use to checkout, clone, or branch your code. Some examples:

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

Add an optional homepage URL and some tags, then click “Create”.

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.

If you want to keep your code updated as you commit, configure your code repository to hit our [Post Commit Hooks](#). This will rebuild your docs every time you push your code.

We support 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.

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.

2.1 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.

2.2 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.

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.

3.1 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.

3.1.1 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. If you have a `README.rst`, it will be transformed into an `index.rst` automatically.

3.1.2 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 an `.md` extension. If you have a `README.md`, it will be transformed into an `index.md` automatically. As MkDocs doesn't support automatic PDF generation, Read the Docs cannot create a PDF version of your documentation with the *Mkdocs* option.

3.2 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_imported_docs(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)
```

3.3 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.

3.4 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

3.5 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 `api/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)
```

3.6 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.

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.

4.1 GitHub and Bitbucket 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 [VCS Integration](#) page.

4.2 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, and anywhere else we have a generic post-commit hook that allows you to POST to a URL to get your documentation built.

4.3 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.

4.4 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.

4.5 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.rtd.org/>).

4.6 Version Control Support Matrix

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

4.7 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.

4.8 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.

4.9 Alternate Domains

We provide support for CNAMEs, subdomains, and a shorturl for your project as well. This is outlined in the [Alternate Domains](#) section.

Support

5.1 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?”

5.2 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.

5.2.1 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.

5.2.2 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

5.3 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>.

Frequently Asked Questions

6.1 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.

6.2 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](#).

6.3 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 %}
```

6.4 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 Mock()

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 (`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.

6.5 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'
```

6.6 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.

6.7 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.

6.8 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.

6.9 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.

6.10 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.

6.11 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.

6.12 I want comments in my docs

RTD doesn't have explicit support for this. That said, a tool like [Disqus](#) can be used for this purpose on RTD.

6.13 How do I support multiple languages of documentation?

See the section on *Localization of Documentation*.

6.14 Do I need to be whitelisted?

No. Whitelisting has been removed as a concept in Read the Docs. You should have access to all of the features already.

6.15 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.

6.16 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.

Read the Docs YAML Config

Read the Docs now has support for configuring builds with a YAML file. The file, `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.

7.1 Supported Settings

7.1.1 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
```

7.1.2 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
```

7.1.3 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.

7.1.4 python

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

python.version

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

The version of Python to use when building your documentation.

```
python:
  version: 3
```

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
```

Guides

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

8.1 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!

8.2 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.

Webhooks

Webhooks are pretty amazing, and help to turn the web into a push instead of pull platform. We have support for hitting a URL whenever you commit to your project and we will try and rebuild your docs. This only rebuilds them if something has changed, so it is cheap on the server side. As anyone who has worked with push knows, pushing a doc update to your repo and watching it get updated within seconds is an awesome feeling.

9.1 GitHub

If your project is hosted on GitHub, you can easily add a hook that will rebuild your docs whenever you push updates:

- Go to the “Settings” page for your project
- Click “Webhooks & Services”
- In the “Services” section, click “Add service”
- In the list of available services, click “ReadTheDocs”
- Check “Active”
- Click “Add service”

Note: The GitHub URL in your Read the Docs project must match the URL on GitHub. The URL is case-sensitive.

If you ever need to manually set the webhook on GitHub, you can point it at <https://readthedocs.org/github>.

9.2 Bitbucket

If your project is hosted on Bitbucket, you can easily add a hook that will rebuild your docs whenever you push updates:

- Go to the “admin” page for your project
- Click “Services”
- In the available service hooks, select “Read the Docs”
- Click “Add service”

If you ever need to manually set the webhook on Bitbucket, you can point it at <https://readthedocs.org/bitbucket>.

9.3 Others

Your ReadTheDocs project detail page has your post-commit hook on it; it will look something along the lines of `http://readthedocs.org/build/<project_name>`. Regardless of which revision control system you use, you can just hit this URL to kick off a rebuild.

You could make this part of a hook using [Git](#), [Subversion](#), [Mercurial](#), or [Bazaar](#), perhaps through a simple script that accesses the build URL using `wget` or `curl`.

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.

10.1 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.

10.2 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/
```

10.3 Style

If you pass the `style` GET argument, we will pass it along to shields.io as is. This will allow you to have custom style badges.

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!

11.1 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.

11.2 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.

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

As an 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.

11.3 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;
    }
}
```

11.4 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`.

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

12.1 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 you 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.

12.2 Project with multiple translations

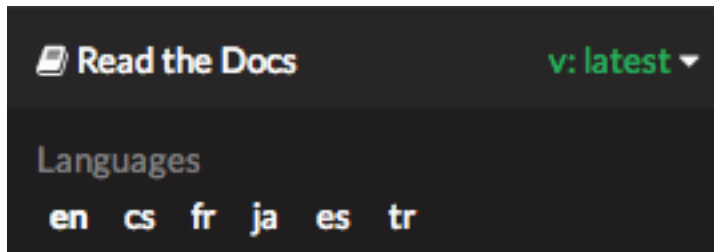
This situation is a bit more complicated. To support this, you will have one parent project and a number of projects marked as translations of that parent. Let's use `phpmyadmin` as an example.

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.

VCS Integration

13.1 GitHub

If you want to integrate GitHub editing into your own theme, the following variables are available in your custom templates:

- `github_user` - GitHub username
- `github_repo` - GitHub repo name
- `github_version` - GitHub blob
- `conf_py_path` - Path in the checkout to the docs root
- `pagename` - Sphinx variable representing the name of the page you're on.
- `display_github`

It can be used like this:

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

13.2 Bitbucket

If you want to integrate Bitbucket editing into your own theme, the following variables are available in your custom templates:

- `bitbucket_user` - Bitbucket username
- `bitbucket_repo` - Bitbucket repo name
- `bitbucket_version` - BitBucket version
- `conf_py_path` - Path in the checkout to the docs root
- `pagename` - Sphinx variable representing the name of the page you're on.
- `display_bitbucket`

It can be used like this:

```
{% if display_bitbucket %}
  <a href="https://bitbucket.org/{{ bitbucket_user }}/{{ bitbucket_repo }}/src/{{ bitbucket_version }}">
{% endif %}
```

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.

14.1 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 create`)
- Dependency Installation (Sphinx)
- User Package Installation (`conda env update`)

14.2 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.

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.

15.1 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`.

15.2 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.

15.3 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">
```

15.4 Links

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

<http://www.matcutts.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>

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>

16.1 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.

16.2 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.

Privacy Levels

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

17.1 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.

17.1.1 Public

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

17.1.2 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.

17.1.3 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.

17.2 Project Objects

17.2.1 Detail Views

- Project Detail (/projects/<slug>)
- API Detail (/api/v1/project/<slug>/)

17.2.2 List Views

- Home Page
- All Projects Page
- User Profile Page (/profiles/<user>/)
- Search

17.3 Version Objects

17.3.1 List Views

- Project Detail (/projects/<slug>)
- Version Selector on Home page
- Version Selector on Documentation page
- Search

User-defined Redirects

18.1 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.

18.2 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.

18.3 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.

18.4 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.

18.5 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.

Feature Requests

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.

19.1 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.

19.1.1 rtdf.org

Links to rtdf.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.

19.2 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.

19.3 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.

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'));
  }
);
```

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:

- *Support Team*
- *Operations Team*
- *Development Team*

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.

21.1 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

21.1.1 Members

- [Eric Holscher](#) (Pacific Time)
- [Anthony Johnson](#) (Pacific Time)
- [edunham](#)
- Your Name Here

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

21.1.2 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.

21.2 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.

21.2.1 Members

- [Eric Holscher](#) (Pacific Time)
- [Anthony Johnson](#) (Pacific Time)
- Your Name Here

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

21.2.2 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.

21.3 Development Team

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

21.3.1 Members

- Eric Holscher
- Anthony Johnson
- Gregor Müllegger
- Stein Magnus Jodal
- Your Name Here

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

21.3.2 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.

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.

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: 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 \
LDLAGS=-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
```

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

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
```

22.1 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](#).

From the dashboard you can import your existing docs provided that they are in a git or mercurial repo.

22.1.1 Creating new 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! We provide a simple editor and two sample pages whenever a new project is created. From there

its up to you to fill in the gaps - we'll build the docs, give you access to history on every revision of your files, and we plan on adding more features in the weeks and months to come.

22.1.2 Importing existing docs

The other side of readthedocs.org is hosting the docs you've already built. 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 on our site whenever you commit to your repo, effectively letting you 'set it and forget it'.

Changelog

This document will track major changes in the project.

Also note, this document is a Markdown file. This is mainly to keep parity with GitHub, and also because we can.

23.1 July 23, 2015

- Django 1.8 Support Merged

23.1.1 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](#) and [finer grained in the repo](#)
- 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/lukaszbdjango-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/psycpg/docs/news.html#what-s-new-in-psycpg-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

23.1.2 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.

23.1.3 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
```

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](#).

24.1 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 Bug](#). 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](#).

24.2 Triageing 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.

24.2.1 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.

24.2.2 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 Bug 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*.

24.2.3 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.

24.3 Helping on translations

If you wish to contribute translations, please do so on [Transifex](#).

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.
```

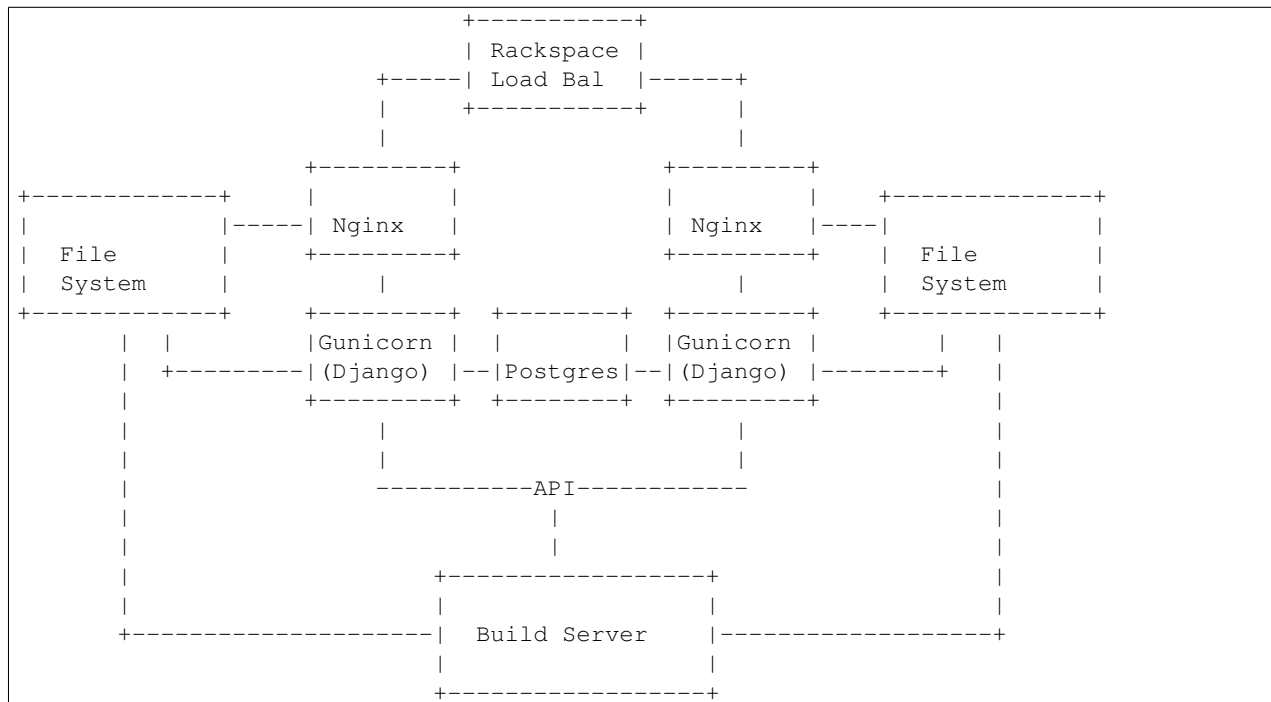
25.1 Continuous Integration

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>

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.

26.1 Diagram



Development Standards

27.1 Front End Development

27.1.1 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 vendored 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.

27.1.2 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`.

27.1.3 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.

27.1.4 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.

27.1.5 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.

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.

28.1 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 create the base image used for container creation. The base image is found in our [container images repo](#). It is the basic container image supported by our community site.

To get started, create the image using the `docker` command line tool. You can name the image whatever you like here, `rtfd-build` is the default name, but can be configured in your settings – see [Configuration](#):

```
docker build -t rtfd-build base/
```

When this process has completed, you should have a working image that Read the Docs can use to start containers.

28.2 Configuration

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

DOCKER_ENABLED 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.

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

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`

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

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.

29.1 Nginx

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

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

29.2 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.

29.3 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 `fabfile` 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 `fabfile`. When we build the `fabfile` docs, we create a symlink for all domains that have pointed at `fabfile` 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.

Interesting Settings

30.1 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.

30.2 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.

30.3 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`.

30.4 PRODUCTION_DOMAIN

Default: `readthedocs.org`

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.

30.5 MULTIPLE_APP_SERVERS

Default: `undefined`

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.

30.6 DEFAULT_PRIVACY_LEVEL

Default: `public`

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.

30.7 INDEX_ONLY_LATEST

Default: `False`

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

30.8 DOCUMENT_PYQUERY_PATH

Default: `div.document`

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.

30.9 USE_PIP_INSTALL

Default: `False`

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

30.10 PUBLIC_DOMAIN

Default: `settings.PRODUCTION_DOMAIN`

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

30.11 ALLOW_ADMIN

Default: `True`

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

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

31.1 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 `_()`.

31.1.1 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.

31.1.2 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')),  
)
```

31.1.3 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")
```

31.1.4 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.

31.2 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 %}
```

31.3 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 `gettext()` 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 gettext 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)
```

31.3.1 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

32.1 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.

32.2 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.

32.3 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](#).

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 Bug 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.

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.

Read the Docs Business Features

Note: These features are for our new business offering, readthedocs.com. We are currently in an invite-only beta, but will be opening up for more users soon.

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.

34.1 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.

34.1.1 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*.

34.1.2 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.

34.1.3 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.

34.2 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.

34.2.1 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.

34.2.2 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.

34.3 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.

34.3.1 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.

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.

35.1 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.

35.1.1 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.

35.1.2 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
    "%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](#).

35.2 Local VM Install

35.2.1 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”

35.2.2 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.

35.2.3 Tips

- If the installation doesn't work on VM using your login/LDAP credentials, try running the operations as root (su).


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 Docs' CSS and static assets.

To start, you should follow the [Installation](#) instructions to get a working copy of the Read the Docs repository locally.

36.1 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.


Read the Docs
Go
Dashboard
Log Out

Header 1.

Header 2.

Header 3.

Header 4.

Header 5.

Paragraph. Aside.

Paragraph with [link](#).

Paragraph with highlighted text.

Long form text. Read the Docs hosts documentation, making it fully *searchable* and easy to find. You can import your docs using any major version control system, including Mercurial, Git, Subversion, and Bazaar. We support [links](#) so your docs get built when you commit code. There's also support for versioning so you can build docs from tags and branches of your code in your repository. A [website](#) is available.

It's free and simple. Read the [Getting Started](#) guide to get going!

Table header	Table header 2
Table element.	Table element 2.
Table element.	Table element 2.

Form Paragraph.

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.

36.2 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.

36.3 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.

36.4 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.

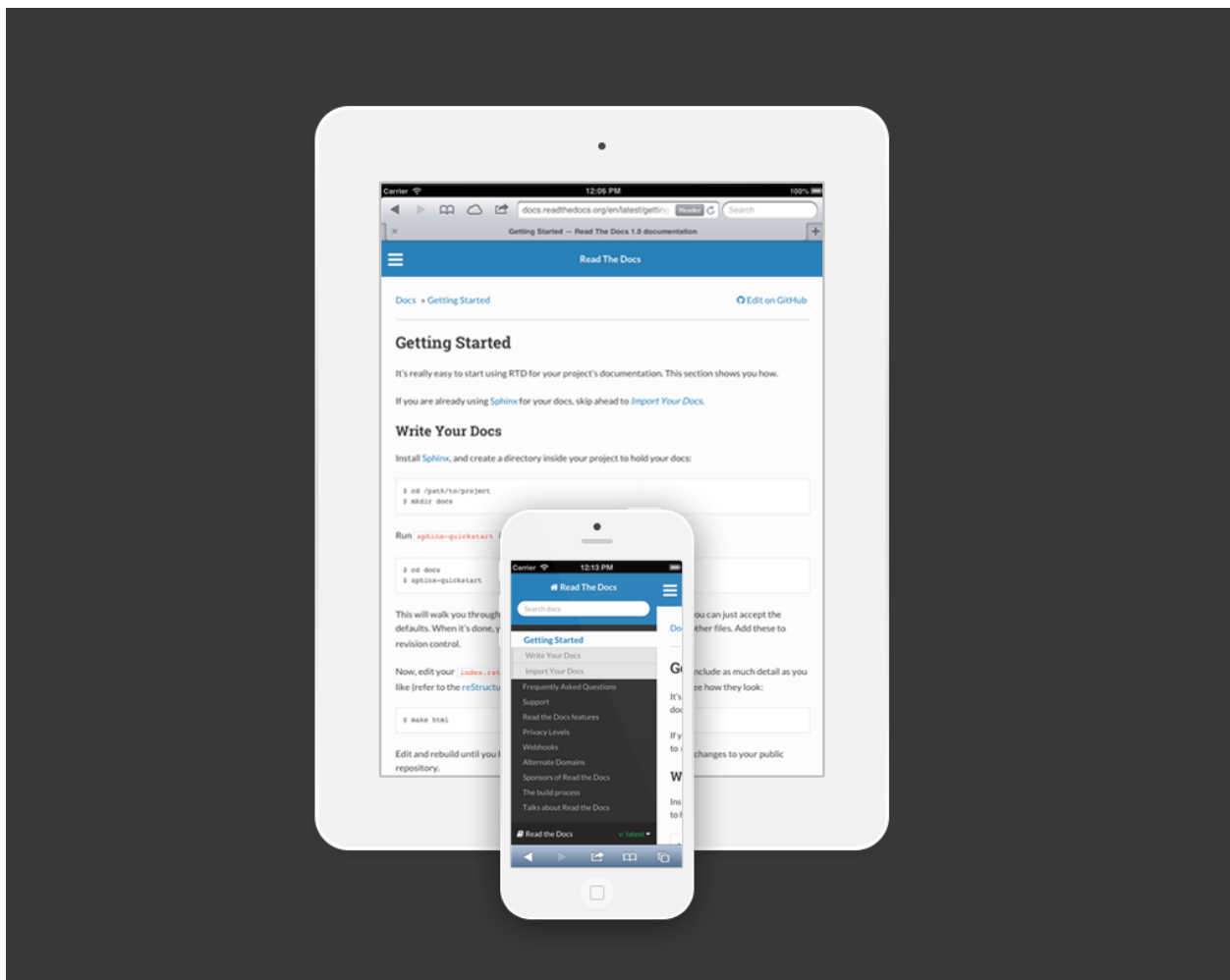
36.5 Contributing

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).

Read the Docs Theme

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



By default, Read the Docs will use its own custom sphinx theme unless you set one yourself in your `conf.py` file. Likewise, setting the theme to `default` will accomplish the same behavior. The theme can be found on [github here](#) and is meant to work independently of Read the Docs itself if you want to just use the theme locally.

This [blog post](#) provides some info about the design, but in short, the theme aims to solve the limitations of Sphinx's default navigation setup, where only a small portion of your docs were accessible in the sidebar. Our theme is also meant to work well on mobile and tablet devices.

37.1 Contributing to the theme

If you have issues or feedback, please [open an issue](#) on the theme's GitHub repository which itself is a submodule within the larger RTD codebase. That means any changes to the theme or the Read the Docs badge styling should be made there. The code is separate so that it can be used independent of Read the Docs as a regular Sphinx theme.

37.2 How the Table of Contents builds

Currently the left menu will build based upon any `toctree(s)` defined in your `index.rst` file. It outputs 2 levels of depth, which should give your visitors a high level of access to your docs. If no toctrees are set in your `index.rst` file the theme reverts to sphinx's usual local toctree which is based upon the heading set on your current page.

It's important to note that if you don't follow the same styling for your rST headers across your documents, the toctree will misbuild, and the resulting menu might not show the correct depth when it renders.

37.3 Other style notes

- As a responsive style, you should not set a height and width to your images.
- Wide tables will add a horizontal scroll bar to maintain the responsive layout.

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 having the code be open is a valuable learning tool, for people to see how a real large website is created.

38.1 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>

38.2 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

38.3 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.

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.

39.1 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)

39.2 Past sponsors

- [Python Software Foundation](#)
- [Revsys](#)
- [Mozilla Web Dev](#)
- [Django Software Foundation](#)
- [Lab305](#)
- [Twilio](#)

39.3 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.

39.3.1 Sponsor Us

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

Talks about Read the Docs

Note: This page is mainly just for showing a demo of updating docs, during a talk.

- PDX Python, May 2011
- OS Bridge, June 2011
- OSCON, July 2011
- Djangocon, July 2011
- OS Bridge, June 2014

Configuration of the production servers

This document is to help people who are involved in the production instance of Read the Docs running on readthedocs.org. It contains implementation details and useful hints for the people handling operations of the servers.

41.1 Elastic Search Setup

You need to install the ICU plugin to make ES work:

```
# Use the correct path to the plugin executable that ships with ES.  
/usr/share/elasticsearch/bin/plugin -install elasticsearch/elasticsearch-analysis-icu/2.3.0
```

```
from search.indexes import Index, PageIndex, ProjectIndex, SectionIndex  
  
# Create the index.  
index = Index()  
index_name = index.timestamped_index()  
index.create_index(index_name)  
index.update_aliases(index_name)  
# Update mapping  
proj = ProjectIndex()  
proj.put_mapping()  
page = PageIndex()  
page.put_mapping()  
sec = SectionIndex()  
sec.put_mapping()
```


/api

```
GET /api/v1/, ??
GET /api/v1/build/, ??
GET /api/v1/build/{id}/, ??
GET /api/v1/file/, ??
GET /api/v1/file/anchor/?q={search_term},
    ??
GET /api/v1/file/search/?q={search_term},
    ??
GET /api/v1/file/{id}/, ??
GET /api/v1/project/, ??
GET /api/v1/project/{id}, ??
GET /api/v1/user/, ??
GET /api/v1/user/{id}/, ??
GET /api/v1/version/, ??
GET /api/v1/version/{id}, ??
GET /api/v1/version/{id}/highest/, ??
GET /api/v1/version/{id}/highest/{version},
    ??
```


E

environment variable
 READTHEDOCS, [15](#)

R

READTHEDOCS, [15](#)