
Read The Docs Documentation

Release 1.0

Eric Holscher, Charlie Leifer, Bobby Grace

August 24, 2014

1	User Documentation	3
1.1	Getting Started	3
1.2	Build Process	4
1.3	Read the Docs features	5
1.4	Support	7
1.5	Frequently Asked Questions	7
1.6	Features	10
2	Developer Documentation	19
2.1	Installation	19
2.2	Contributing to Read the Docs	21
2.3	Running tests	22
2.4	Architecture	22
2.5	How we use symlinks	23
2.6	Interesting Settings	24
2.7	Internationalization	25
2.8	Administrative Tasks	28
2.9	Read the Docs Public API	29
2.10	API	40
3	Designer Documentation	49
3.1	Designing Read the Docs	49
3.2	Read the Docs Theme	52
4	About Read the Docs	55
4.1	Sponsors of Read the Docs	55
4.2	Talks about Read the Docs	55
5	Operations Documentation	57
5.1	Configuration of the production servers	57
	Python Module Index	61

Read the Docs hosts documentation for the open source community. It supports [Sphinx](#) docs written with [reStructuredText](#), and can pull from your [Subversion](#), [Bazaar](#), [Git](#), and [Mercurial](#) repositories. The code is open source, and available on [github](#).

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

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

Information about development and running your own instance is also available:

- *Developer Documentation*
- *Designer Documentation*
- *Operations Documentation*

User Documentation

1.1 Getting Started

This document will show you how to get up and running with Read the Docs.

If you are already using [Sphinx](#) for your docs, skip ahead to *Import Your Docs*.

There is a [screencast](#) that will help you get started if you prefer.

1.1.1 Write Your Docs

Install [Sphinx](#), and 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 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
```

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 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 minutes 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.

If you have any more trouble, don’t hesitate to reach out to us. The [Support](#) page has more information on getting in touch.

1.2 Build Process

1.2.1 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 *Use Virtualenv* 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(project, version)
if project.use_virtualenv:
    run('python setup.py install')
    if project.requirements_file:
        run('pip install -r %s' % project.requirements_file)
build_docs(version=version, pdf=pdf, man=man, epub=epub, dash=dash)
copy_files(artifact_dir)
```

1.2.2 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.

1.2.3 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:

- Latex (texlive-full)
- libevent (libevent-dev)
- dvisvgm
- graphviz
- libxslt1.1
- libxml2-dev

1.2.4 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 *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)
```

1.2.5 Deleting a stale or broken build environment

RTD doesn't expose this in the UI, but it is possible to remove the build directory of your project. If you want to remove a build environment for your project, hit http://readthedocs.org/wipe/<project_slug>/<version_slug>/. You must be logged in to do this.

1.3 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.

1.3.1 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.

1.3.2 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.

1.3.3 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.

1.3.4 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.

1.3.5 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/>).

1.3.6 Version Control Support Matrix

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

1.3.7 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.

1.3.8 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.

1.3.9 Alternate Domains

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

1.4 Support

1.4.1 Getting Help

The easiest way to get help with the project is to join the `#readthedocs` channel on Freenode. We hang out there and you can get real-time help with your projects. The other good way is to open an issue on [Github](#).

The mailing list at <https://groups.google.com/forum/#!forum/read-the-docs> is also available for support.

1.4.2 Backwards Incompatible Changes

Backwards Incompatible Changes will be emailed to the [mailing list](#). They will be prefixed with “Backwards Incompatible Changes”. We are thinking about having some kind of Backwards Incompatible Changes policy, much like the 1.0 of a code base, once we define the redirects and interfaces that we wish to expose permanently.

1.5 Frequently Asked Questions

1.5.1 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.

1.5.2 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's `conf.py` file, you can vary the behavior based on this. For example:

```
import os
on_rtd = os.environ.get('READTHEDOCS', None) == '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 %}
```

1.5.3 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 (ie `pip install mock`) and replace the above import:

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

1.5.4 Can I make search engines only see one version of my docs?

You can do this for Google at least with a canonical link tag. It should look like:

```
<link rel="canonical" href="http://ericholscher.com/
{%- for word in pagename.split('/') -%}
  {%- if word != 'index' -%}
    {%- if word != '' -%}
      {{ word }}/
    {%- endif -%}
  {%- endif -%}
{%- endfor -%}
{% if builder == "dirhtml" %}/{% else %}.html{% endif %}
">
```

1.5.5 Deleting a stale or broken build environment

RTD doesn't expose this in the UI, but it is possible to remove the build directory of your project. If you want to remove a build environment for your project, hit http://readthedocs.org/wipe/<project_slug>/<version_slug>/. You must be logged in to do this.

1.5.6 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.org` domain:

<http://celery.readthedocs.org/projects/kombu/en/latest/>

This also works the same for CNAME's:

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

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

1.5.7 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.

1.5.8 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.

1.5.9 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.

1.5.10 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.

1.5.11 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.

1.5.12 How do I support multiple languages of documentation?

See the section on *Localization of Documentation*.

1.5.13 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.

1.5.14 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.

1.6 Features

1.6.1 Webhooks

Web hooks 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.

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”

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 “Hooks”
- In the available service hooks, select “Read the Docs”
- Click “Add hook”

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/<pk>`. 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`.

1.6.2 Badges

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

Status Badges

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

Here are a few examples:

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

Project Pages

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

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

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

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

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

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.

1.6.3 Alternate Domains

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

Subdomain Support

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

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

CNAME Support

If you have your own domain, you can still host with us. If you point a CNAME record in your DNS to the subdomain for your project, it should magically serve your latest documentation on the custom domain. Using pip as another 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  fabric-docs.readthedocs.org.
```

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 http://pip.readthedocs.org:80;
        proxy_set_header Host $http_host;
        proxy_set_header X-Forwarded-Proto http;
        proxy_set_header X-Real-IP $remote_addr;
        proxy_set_header X-Scheme $scheme;
        proxy_set_header X-RTD-SLUG pip;
        proxy_connect_timeout 10s;
        proxy_read_timeout 20s;
    }
}
```

rtfd.org

You can also use `rtfd.org` as a short URL for Read the Docs. For example, <http://pip.rtfd.org> redirects to its documentation page. Any use of `rtfd.org` will simply be redirected to `readthedocs.org`.

1.6.4 Localization of Documentation

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

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

Single project in another language

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

Project with multiple translations

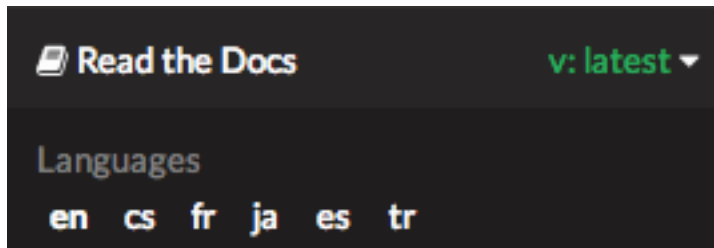
This situation is a bit more complicated. To support this, you will have one parent project and a number of projects marked as translations of that parent. Let's use `phpmyadmin` as an example.

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

This has the results of serving:

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

It also gets included in the Read the Docs flyout:



1.6.5 VCS Integration

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 %}
```

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 %}
```

1.6.6 Canonical URLs

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

Read the Docs uses these in two ways:

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

Example

Fabric hosts their docs on Read the Docs. They mostly use their own domain for them `http://docs.fabfile.org`. This means that Google will index both `http://fabric-docs.readthedocs.org` 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.org`, it will know that it should really point at `http://docs.fabfile.org`.

Enabling

You can set the canonical URL for your project in the Project Admin page. Check your [dashboard](#) for a list of your projects.

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.org/en/latest/installing.html">
```

Links

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

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

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

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

1.6.7 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`.

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](#).

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.

Redirects on root URLs

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

1.6.8 Single Version Documentation

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

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

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

Enabling

You can toggle the “Single Version” option on or off for your project in the Project Admin page. Check your [dashboard](#) for a list of your projects.

Effects

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

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

1.6.9 Privacy Levels

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

Understanding the Privacy Levels

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

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

Public

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

Protected

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

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

Private

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

Project Objects

Detail Views

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

List Views

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

Version Objects

List Views

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

Developer Documentation

2.1 Installation

Installing RTD is pretty simple. Here is a step by step plan on how to do it.

First, obtain [Python](#) 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.

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

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

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
```

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 with [virtualenv](#)):

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

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

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

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

```
cd readthedocs
./manage.py syncdb
```

This will prompt you to create a superuser account for Django. Do that. Then:

```
./manage.py migrate
```

Go ahead and load in a couple users and a test projects:

```
./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:

```
./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).

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:

```
./manage.py update_repos pip
```

2.1.1 Solr (Search) Setup

Apache Solr is used to index and search documents. This is an optional requirement, and only necessary if you want to develop or use search.

Additional python requirements necessary to use Solr:

```
pip install pysolr
pip install pyquery
```

Fetch and unpack Solr:

```
curl -O http://archive.apache.org/dist/lucene/solr/3.5.0/apache-solr-3.5.0.tgz
tar xvfz apache-solr-3.5.0.tgz && SOLR_PATH=`pwd`/apache-solr-3.5.0/example
```

Generate the schema.xml file:

```
./manage.py build_solr_schema > $SOLR_PATH/solr/conf/schema.xml
```

Start the server:

```
cd $SOLR_PATH && java -jar start.jar
```

Index the data:

```
./manage.py build_files # creates database objects referencing project files
./manage.py update_index
```

Note: For production environments, you'll want to run Solr in a more permanent servlet container, such as Tomcat or Jetty. Ubuntu distributions include prepackaged Solr installations. Try `aptitude install solr-tomcat` or `aptitude install solr-jetty`.

See `/etc/[solr|tomcat6|jetty]` for configuration options. The `schema.xml` file must be replaced with the version built by `django-haystack`.

2.1.2 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.

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.

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'.

2.1.3 Installation with Vagrant

It is also possible to run RTD using [Vagrant](#), using Vagrant v1.1+ and the [Salt plugin](#) for [Vagrant](#), by running the following commands:

```
vagrant plugin install vagrant-salt
vagrant up
```

The [Vagrant](#) virtual machine will take a while to create and provision, and will leave a virtual machine running an instance of RTD with the following settings:

URL <http://localhost:8000>

Username docs

Password docs

Note: The hostname `localhost` is used here, though it is possible to test RTD and subdomains by adding entries in `/etc/hosts` for `readthedocs.org` and your subdomains on `readthedocs.org`, pointing to `127.0.0.1` on the host system. The site will be available at <http://readthedocs.org:8000> with the proper records set up.

The repository is shared with the host file system, so edits can be made outside the virtual environment.

2.2 Contributing to Read the Docs

Read the Docs follows the standard Contribution Guidelines set forth at [contribution-guide.org](#). Please read that site and follow the instructions to make sure your patches will be accepted.

2.2.1 Tickets

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.

2.2.2 Translations

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

2.3 Running tests

Currently RTD isn't well tested. This is a problem, and it is being worked on. However, we do have a basic test suite. To run the tests, you need simply need to run:

```
./manage.py test rtd_tests
```

This should spit out a bunch of info, build a couple projects, and eventually pass.

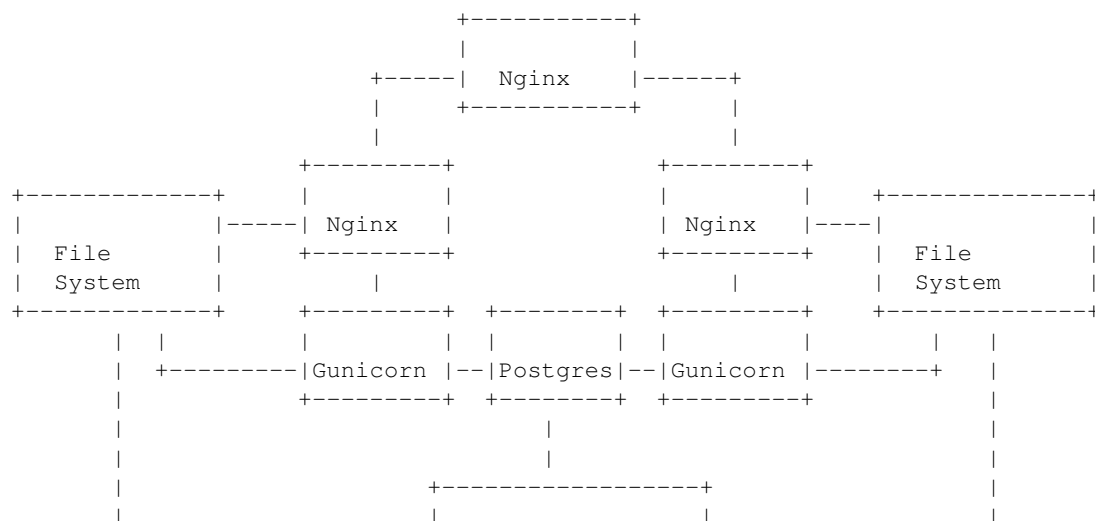
2.3.1 Continuous Integration

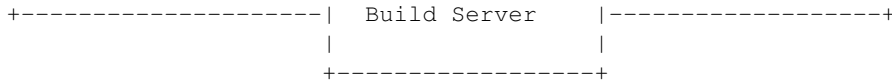
The RTD test suite is exercised by Travis CI on every push to our repo at GitHub. You can check out the current build status: <https://travis-ci.org/rtd/readthedocs.org>

2.4 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.

2.4.1 Diagram





2.5 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 out database. This reduces the total number of servers to serve a request to 1, each of which is redundant.

2.5.1 Nginx

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

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

2.5.2 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.

2.5.3 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 projects slug mentioned. It is simply there in the symlink in the `cnames` directory, and the docs are served from there.

2.6 Interesting Settings

2.6.1 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.

2.6.2 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.

2.6.3 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`.

2.6.4 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.

2.6.5 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.

2.6.6 INDEX_ONLY_LATEST

Default: `False`

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

2.6.7 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.

2.6.8 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`.

2.7 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

2.7.1 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 `_()`.

Interpolation

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

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

This is **wrong!**

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

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

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

Localization Comments

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

Example:

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

Adding Context with msgctxt

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

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

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

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

Example:

```
from django.utils.translation import pgettext  
  
month = pgettext("text for the search button on the form", "Search")
```

Plurals

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

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

A more realistic example might be:

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

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

2.7.2 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 %}
```

2.7.3 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)
```

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()`.

2.8 Administrative Tasks

2.8.1 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 `readthedocs/` 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.

2.8.2 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.

2.8.3 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](#).

2.9 Read the Docs Public API

We have a limited public API that is available for you to get data out of the site. This page will only show a few of the basic parts, please file a ticket or ping us on IRC (#readthedocs on [Freenode \(chat.freenode.net\)](#)) if you have feature requests.

This document covers the read-only API provided. We have plans to create a read/write API, so that you can easily automate interactions with your project.

The API is written in Tastypie, which provides a nice ability to browse the API from your browser. If you go to <http://readthedocs.org/api/v1/?format=json> and just poke around, you should be able to figure out what is going on.

2.9.1 A basic API client using slumber

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

```
import slumber
import json

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

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

#val = api.build(49252).get()
#val = api.build.get(project__slug='read-the-docs')

#val = api.user.get(username='eric')

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

#val = api.version('pip').highest.get()
#val = api.version('pip').highest('0.8').get()
```

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

2.9.2 Example of adding a user to a project

```
import slumber
```

```
USERNAME = 'eric'
PASSWORD = 'test'
```

```
user_to_add = 'coleifer'
project_slug = 'read-the-docs'
```

```
api = slumber.API(base_url='http://readthedocs.org/api/v1/', authentication={'name': USERNAME, 'passw
```

```
project = api.project.get(slug=project_slug)
user = api.user.get(username=user_to_add)
project_objects = project['objects'][0]
user_objects = user['objects'][0]
```

```
data = {'users': project_objects['users'][:]}
data['users'].append(user_objects['resource_uri'])
```

```
print "Adding %s to %s" % (user_objects['username'], project_objects['slug'])
api.project(project_objects['id']).put(data)
```

```
project2 = api.project.get(slug=project_slug)
project2_objects = project2['objects'][0]
print "Before users: %s" % project_objects['users']
print "After users: %s" % project2_objects['users']
```

2.9.3 API Endpoints

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

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

2.9.4 Root

GET **/api/v1/**

Retrieve a list of resources.

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

```

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

```

Data

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

2.9.5 Builds

GET `/api/v1/build/`

Retrieve a list of Builds.

```

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

```

Data

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

2.9.6 Build

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

Path arguments **id** – A Build id.

Retrieve a single Build.

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

Data

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

2.9.7 Files

GET **/api/v1/file/**

Retrieve a list of Files.

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

```
"objects": [FILES]
}
```

Data

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

2.9.8 File

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

Path arguments `id` – A File id.

Retrieve a single File.

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

Data

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

2.9.9 Projects

GET `/api/v1/project/`

Retrieve a list of Projects.

```
{
  "meta": {
    "limit": 20,
    "next": "/api/v1/project/?limit=20&offset=20",

```

```
    "offset": 0,
    "previous": null,
    "total_count": 2067
  },
  "objects": [PROJECTS]
}
```

Data

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

2.9.10 Project

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

Path arguments `id` – A Project id.

Retrieve a single Project.

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

Data

- **absolute_url** (*string*) – URI for project (not the Project object from the API.)
- **analytics_code** (*string*) – Analytics tracking code.
- **copyright** (*string*) – Copyright
- **crate_url** (*string*) – Crate.io URI.
- **default_branch** (*string*) – Default branch.
- **default_version** (*string*) – Default version.
- **description** (*string*) – Description of project.
- **django_packages_url** (*string*) – Djangopackages.com URI.
- **documentation_type** (*string*) – Either “sphinx” or “sphinx_html”.
- **id** (*string*) – Project id.
- **modified_date** (*string*) – Last modified date.
- **name** (*string*) – Project name.
- **project_url** (*string*) – Project homepage.
- **pub_date** (*string*) – Last published date.
- **repo** (*string*) – URI for VCS repository.
- **repo_type** (*string*) – Type of VCS repository.
- **requirements_file** (*string*) – Pip requirements file for packages needed for building docs.
- **resource_uri** (*string*) – URI for Project.
- **slug** (*string*) – Slug.
- **subdomain** (*string*) – Subdomain.
- **suffix** (*string*) – File suffix of docfiles. (Usually “.rst”.)
- **theme** (*string*) – Sphinx theme.
- **use_virtualenv** (*boolean*) – Build project in a virtualenv? (True or False)
- **users** (*array*) – Array of readthedocs.org user URIs for administrators of Project.
- **version** (*string*) – DEPRECATED.

2.9.11 Users

GET **/api/v1/user/**

Retrieve List of Users

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

```

```
"objects": [USERS]
}
```

Data

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

2.9.12 User

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

Path arguments `id` – A User id.

Retrieve a single User

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

Data

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

2.9.13 Versions

GET `/api/v1/version/`

Retrieve a list of Versions.

```
{
  "meta": {
    "limit": 20,
    "next": "/api/v1/version/?limit=20&offset=20",

```



```

    "offset": 0,
    "previous": null,
    "total_count": 16437
  },
  "objects": [VERSIONS]
}

```

Data

- **limit** (*integer*) – Number of Versions returned.
- **next** (*string*) – URI for next set of Versions.
- **offset** (*integer*) – Current offset used for pagination.
- **previous** (*string*) – URI for previous set of Versions.
- **total_count** (*integer*) – Total number of Versions.
- **objects** (*array*) – Array of [Version](#) objects.

2.9.14 Version

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

Path arguments `id` – A Version id.

Retrieve a single Version.

```

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

```

Data

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

2.9.15 Filtering Examples

Find Highest Version

`http://readthedocs.org/api/v1/version/pip/highest/?format=json`

GET `/api/v1/version/{id}/highest/`

Path arguments `id` – A Version id.

Retrieve highest version.

```
{
  "is_highest": true,
  "project": "Version 1.0.1 of pip (5476)",
  "slug": [
    "1.0.1"
  ],
  "url": "/docs/pip/en/1.0.1/",
  "version": "1.0.1"
}
```

Compare Highest Version

This will allow you to compare whether a certain version is the highest version of a specific project. The below query should return a `'is_highest': false` in the returned dictionary.

`http://readthedocs.org/api/v1/version/pip/highest/0.8/?format=json`

GET `/api/v1/version/{id}/highest/{version}`

Path arguments

- `id` – A Version id.
- `version` – A Version number or string.

Retrieve highest version.

```
{
  "is_highest": false,
  "project": "Version 1.0.1 of pip (5476)",
  "slug": [
    "1.0.1"
  ],
  "url": "/docs/pip/en/1.0.1/",
  "version": "1.0.1"
}
```

File Search

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

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

Path arguments `search_term` – Perform search with this term.

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

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

Anchor Search

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

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

Path arguments `search_term` – Perform search of files containing anchor text with this term.

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

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

2.10 API

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

2.10.1 bookmarks

`bookmarks.admin`

Django admin interface for `Bookmark`.

`bookmarks.models`

```
class bookmarks.models.Bookmark(*args, **kwargs)
    Bookmark(id, project_id, user_id, date, url, desc)
```

`bookmarks.urls`

`bookmarks.views`

2.10.2 builds

`builds.admin`

Django admin interface for `Build` and related models.

`builds.models`

```
class builds.models.Build(*args, **kwargs)
    Build(id, project_id, version_id, type, state, date, success, setup, setup_error, output, error, exit_code, commit)

class builds.models.Version(*args, **kwargs)
    Version(id, project_id, type, identifier, verbose_name, slug, supported, active, built, uploaded, privacy_level)

    get_build_path()
        Return version build path if path exists, otherwise None

    save(*args, **kwargs)
        Add permissions to the Version for all owners on save.

class builds.models.VersionAlias(*args, **kwargs)
    VersionAlias(id, project_id, from_slug, to_slug, largest)
```

`builds.urls`

`builds.views`

2.10.3 doc_builder

`doc_builder.base`

class `doc_builder.base.BaseBuilder` (*version*, *force=False*)

The Base for all Builders. Defines the API for subclasses.

Expects subclasses to define `old_artifact_path`, which points at the directory where artifacts should be copied from.

build (*id=None*, ***kwargs*)

Do the actual building of the documentation.

clean (***kwargs*)

Clean the path where documentation will be built

force (***kwargs*)

An optional step to force a build even when nothing has changed.

move (***kwargs*)

Move the documentation from it's generated place to its artifact directory.

`doc_builder.backends`

`doc_builder.backends.sphinx`

class `doc_builder.backends.sphinx.BaseSphinx` (**args*, ***kwargs*)

The parent for most sphinx builders.

append_conf (***kwargs*)

Modify the given `conf.py` file from a whitelisted user's project.

2.10.4 core

`core.admin`

Django admin interface for core models.

`core.forms`

class `core.forms.FacetField` (*choices=()*, *required=True*, *widget=None*, *label=None*, *initial=None*, *help_text=u''*, **args*, ***kwargs*)

For filtering searches on a facet, with validation for the format of facet values.

valid_value (*value*)

Although this is a choice field, no choices need to be supplied. Instead, we just validate that the value is in the correct format for facet filtering (`facet_name:value`)

class `core.forms.FacetedSearchForm` (**args*, ***kwargs*)

Supports fetching faceted results with a corresponding query.

facets A list of facet names for which to get facet counts

models Limit the search to one or more models

`core.middleware`

class `core.middleware.SingleVersionMiddleware`

Reset urlconf for requests for 'single_version' docs.

In settings.MIDDLEWARE_CLASSES, SingleVersionMiddleware must follow after SubdomainMiddleware.

`core.models`

class `core.models.UserProfile` (*args, **kwargs)

Additional information about a User.

get_contribution_details ()

Gets the line to put into commits to attribute the author.

Returns a tuple (name, email)

`core.search_sites`

`core.views`

Core views, including the main homepage, post-commit build hook, documentation and header rendering, and server errors.

core.views.default_docs_kwargs (request, project_slug=None)

Return kwargs used to reverse lookup a project's default docs URL.

Determining which URL to redirect to is done based on the kwargs passed to reverse(serve_docs, kwargs). This function populates kwargs for the default docs for a project, and sets appropriate keys depending on whether request is for a subdomain URL, or a non-subdomain URL.

core.views.get_suggestion (project_slug, lang_slug, version_slug, pagename, user)

| project | version | language | What to show |

1 | 0 | 0 | 0 | Error message |

2 | 0 | 0 | 1 | Error message (Can't happen) |

3 | 0 | 1 | 0 | Error message (Can't happen) |

4 | 0 | 1 | 1 | Error message (Can't happen) |

5 | 1 | 0 | 0 | A link to top-level page of default version |

6 | 1 | 0 | 1 | Available versions on the translation project |

7 | 1 | 1 | 0 | Available translations of requested version |

8 | 1 | 1 | 1 | A link to top-level page of requested version |

core.views.github_build (*args, **kwargs)

A post-commit hook for github.

core.views.redirect_lang_slug (request, lang_slug, project_slug=None)

Redirect /en/ to /en/latest/.

core.views.redirect_page_with_filename (request, filename, project_slug=None)

Redirect /page/file.html to /en/latest/file.html.

```
core.views.redirect_project_slug(request, project_slug=None)
    Redirect / to /en/latest/.
```

```
core.views.redirect_version_slug(request, version_slug, project_slug=None)
    Redirect /latest/ to /en/latest/.
```

```
core.views.server_error(request, template_name='500.html')
    A simple 500 handler so we get media
```

```
core.views.server_error_404(request, template_name='404.html')
    A simple 404 handler so we get media
```

`core.management.commands`

This is where custom `manage.py` commands are defined.

```
class core.management.commands.update_repos.Command
    Custom management command to rebuild documentation for all projects on the site.    Invoked via
    ./manage.py update_repos.
```

2.10.5 projects

`projects.admin`

Django administration interface for `Project` and related models.

`projects.constants`

Default values and other various configuration for projects, including available theme names and repository types.

`projects.forms`

`projects.models`

```
class projects.models.EmailHook(*args, **kwargs)
    EmailHook(id, project_id, email)
```

```
class projects.models.ImportedFile(*args, **kwargs)
    ImportedFile(id, project_id, version_id, name, slug, path, md5)
```

```
class projects.models.Project(*args, **kwargs)
    Project(id, pub_date, modified_date, name, slug, description, repo, repo_type, project_url, canon-
    ical_url, version, copyright, theme, suffix, single_version, default_version, default_branch, require-
    ments_file, documentation_type, analytics_code, path, conf_py_file, featured, skip, mirror, use_virtualenv,
    python_interpreter, use_system_packages, django_packages_url, privacy_level, version_privacy_level, lan-
    guage, main_language_project_id, num_major, num_minor, num_point)
```

```
    all_active_versions()
```

A temporary workaround for `active_versions` filtering out things that were active, but failed to build

```
    artifact_path(type, version='latest')
```

The path to the build html docs in the project.

cnames_symlink_path (*domain*)

Path in the doc_path that we symlink cnames

This has to be at the top-level because Nginx doesn't know the projects slug.

find (*file, version*)

A balla API to find files inside of a projects dir.

full_build_path (*version='latest'*)

The path to the build html docs in the project.

full_dash_path (*version='latest'*)

The path to the build dash docs in the project.

full_doc_path (*version='latest'*)

The path to the documentation root in the project.

full_epub_path (*version='latest'*)

The path to the build epub docs in the project.

full_find (*file, version*)

A balla API to find files inside of a projects dir.

full_json_path (*version='latest'*)

The path to the build json docs in the project.

full_latex_path (*version='latest'*)

The path to the build latex docs in the project.

full_man_path (*version='latest'*)

The path to the build man docs in the project.

full_singlehtml_path (*version='latest'*)

The path to the build singlehtml docs in the project.

get_default_branch ()

Get the version representing "latest"

get_default_version ()

Get the default version (slug).

Returns self.default_version if the version with that slug actually exists (is built and published). Otherwise returns 'latest'.

get_docs_url (*version_slug=None, lang_slug=None*)

Return a url for the docs. Always use http for now, to avoid content warnings.

rtd_build_path (*version='latest'*)

The destination path where the built docs are copied.

single_version_symlink_path ()

Path in the doc_path for the single_version symlink.

static_metadata_path ()

The path to the static metadata JSON settings file

subprojects_symlink_path (*project*)

Path in the doc_path that we symlink subprojects

supported_versions (*flat=True*)

Get the list of supported versions. Returns a list of version strings.

translations_symlink_path (*language=None*)

Path in the doc_path that we symlink translations


```
class projects.models.ProjectRelationship(*args, **kwargs)
    ProjectRelationship(id, parent_id, child_id)
```

```
class projects.models.WebHook(*args, **kwargs)
    WebHook(id, project_id, url)
```

projects.search_indexes

projects.tasks

Tasks related to projects, including fetching repository code, cleaning `conf.py` files, and rebuilding documentation.

```
projects.tasks.create_build(version, api, record)
```

Create the build object. If we're recording it, save it to the DB. Otherwise just use an empty hash.

```
projects.tasks.docker_build(version_pk, pdf=True, man=True, epub=True, dash=True,
                             search=True, force=False, intersphinx=True, localmedia=True)
```

The code that executes inside of docker

```
projects.tasks.ensure_version(api, project, version_pk)
```

Ensure we're using a sane version. This also creates the "latest" version if it doesn't exist.

```
projects.tasks.fileify(version_pk)
```

Create ImportedFile objects for all of a version's files.

This is a prereq for indexing the docs for search. It also causes celery-haystack to kick off an index of the file.

```
projects.tasks.finish_build(version, build, results)
```

Build Finished, do house keeping bits

```
projects.tasks.record_build(api, record, build, results, state)
```

Record a build by hitting the API.

Returns nothing

```
projects.tasks.setup_environment(version)
```

Build the virtualenv and install the project into it.

```
projects.tasks.setup_vcs(version, build, api)
```

Update the checkout of the repo to make sure it's the latest. This also syncs versions in the DB.

projects.utils

Utility functions used by projects.

```
projects.utils.find_file(file)
```

Find matching filenames in the current directory and its subdirectories, and return a list of matching filenames.

```
projects.utils.run(*commands, **kwargs)
```

Run one or more commands, and return `(status, out, err)`. If more than one command is given, then this is equivalent to chaining them together with `&&`; if all commands succeed, then `(status, out, err)` will represent the last successful command. If one command failed, then `(status, out, err)` will represent the failed command.

```
projects.utils.safe_write(filename, contents)
```

Write `contents` to the given `filename`. If the `filename`'s directory does not exist, it is created. Contents are written as UTF-8, ignoring any characters that cannot be encoded as UTF-8.

```
projects.utils.update_static_metadata(project_pk)
```

This is here to avoid circular imports in `models.py`

`projects.views`

`projects.views.public`

`projects.views.public.project_badge(request, project_slug, redirect=False)`

Return a sweet badge for the project

`projects.views.public.project_detail(request, project_slug)`

A detail view for a project with various dataz

`projects.views.public.project_downloads(request, project_slug)`

A detail view for a project with various dataz

`projects.views.public.search_autocomplete(request)`

return a json list of project names

`projects.views.public.version_autocomplete(request, project_slug)`

return a json list of version names

`projects.views.private`

`class projects.views.private.ProjectDashboard(**kwargs)`

A dashboard! If you aint know what that means you aint need to. Essentially we show you an overview of your content.

`model`

alias of `Project`

`projects.views.private.project_advanced(request, *args, **kwargs)`

Edit an existing project - depending on what type of project is being edited (created or imported) a different form will be displayed

`projects.views.private.project_delete(request, *args, **kwargs)`

Make a project as deleted on POST, otherwise show a form asking for confirmation of delete.

`projects.views.private.project_edit(request, *args, **kwargs)`

Edit an existing project - depending on what type of project is being edited (created or imported) a different form will be displayed

`projects.views.private.project_import(request, *args, **kwargs)`

Import docs from an repo

`projects.views.private.project_manage(request, *args, **kwargs)`

The management view for a project, where you will have links to edit the projects' configuration, edit the files associated with that project, etc.

Now redirects to the normal `/projects/<slug>` view.

`projects.views.private.project_versions(request, *args, **kwargs)`

Shows the available versions and lets the user choose which ones he would like to have built.

2.10.6 `vcs_support`

`vcs_support.base`

`class vcs_support.base.BaseCLI`

Helper class for CLI-heavy classes.

run (*args)

Parameters **bits** – list of command and args. See `subprocess` docs

class `vcs_support.base.BaseContributionBackend(repo)`

Base class for contribution backends.

The main purpose of this base class is to define the API.

classmethod **accepts** (url)

Classmethod that checks if a given repository URL is supported by this backend.

get_branch_file (branch, filename)

Returns the contents of a file as it is in the specified branch.

push_branch (branch, title='', comment='')

Pushes a branch upstream.

set_branch_file (branch, filename, contents, comment='')

Saves the file in the specified branch.

class `vcs_support.base.BaseVCS(project, version)`

Base for VCS Classes. Built on top of the BaseCLI.

branches

Returns a list of VCSVersion objects. See VCSVersion for more information.

checkout (identifier=None)

Set the state to the given identifier.

If identifier is None, checkout to the latest revision.

The type and format of identifier may change from VCS to VCS, so each backend is responsible to understand it's identifiers.

commit

Returns a string representing the current commit.

get_contribution_backend ()

Returns a contribution backend or None for this repository. The backend is detected via the repository URL.

make_clean_working_dir ()

Ensures that the working dir exists and is empty

tags

Returns a list of VCSVersion objects. See VCSVersion for more information.

update ()

If self.working_dir is already a valid local copy of the repository, update the repository, else create a new local copy of the repository.

class `vcs_support.base.VCSProject`

Transient object to encapsulate a projects stuff

class `vcs_support.base.VCSVersion(repository, identifier, verbose_name)`

Represents a Version (tag or branch) in a VCS.

This class should only be instantiated in BaseVCS subclasses.

It can act as a context manager to temporarily switch to this tag (eg to build docs for this tag).

Designer Documentation


3.1 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.

3.1.1 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.

3.1.2 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.

3.1.3 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.

3.1.4 Sphinx Template Changes

Styles for generated documentation are located in `readthedocs/templates/sphinx/_static/rtd.css`

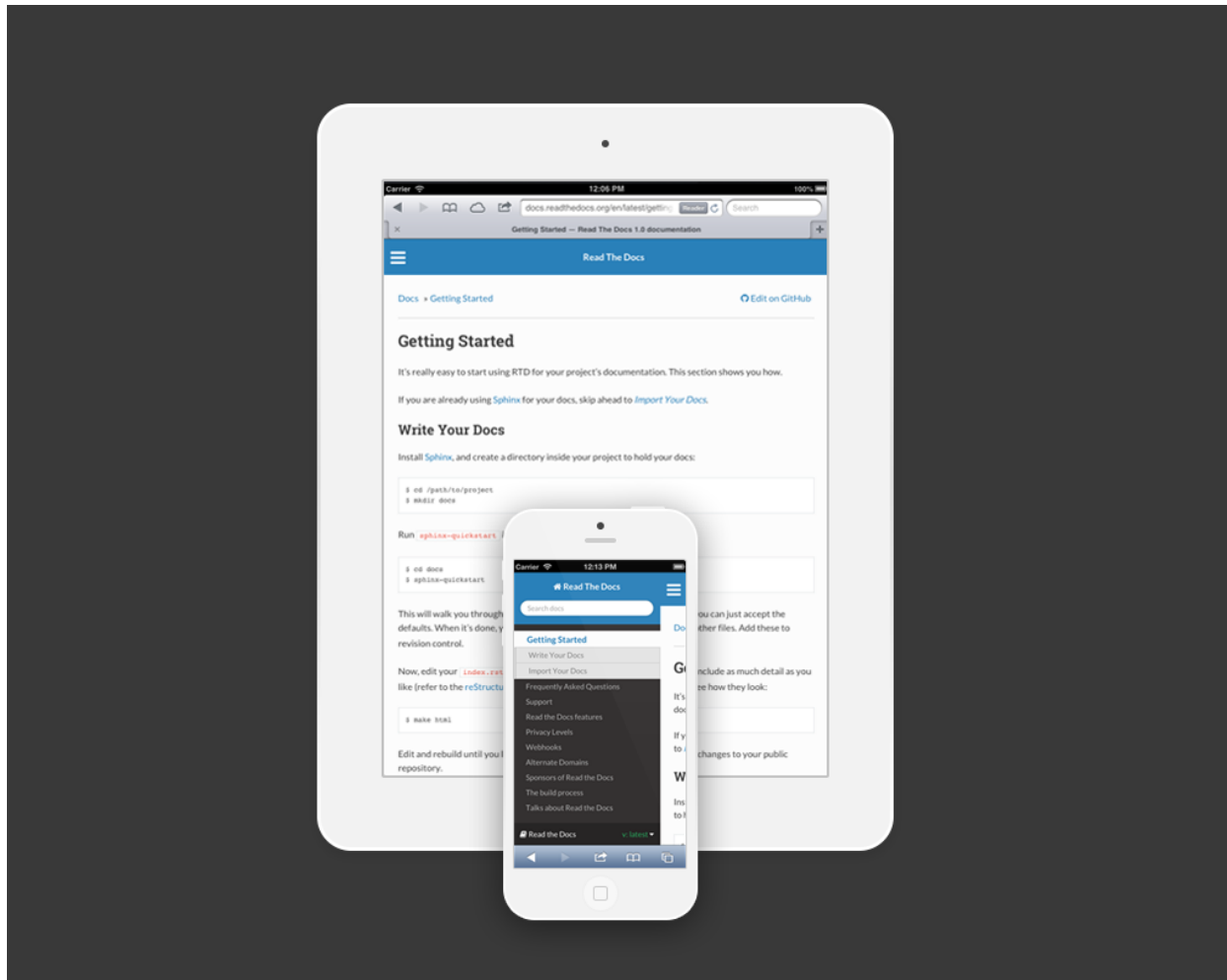
Of note, projects will retain the version of that file they were last built with – so if you’re editing that file and not seeing any changes to your local built documentation, you need to rebuild your example project.

3.1.5 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).

3.2 Read the Docs Theme



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.

3.2.1 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.

3.2.2 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.

3.2.3 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.

3.2.4 How do I use this locally, *and* on Read the Docs?

Unfortunately, at the moment Read the Docs can't handle importing `sphinx_rtd_theme`, so if you try to use that theme for building on both Read the Docs and locally, it will fail. To build it locally, and on Read the Docs:

```
# on_rtd is whether we are on readthedocs.org
import os
on_rtd = os.environ.get('READTHEDOCS', None) == 'True'

if not on_rtd: # only import and set the theme if we're building docs locally
    import sphinx_rtd_theme
    html_theme = 'sphinx_rtd_theme'
    html_theme_path = [sphinx_rtd_theme.get_html_theme_path()]

# otherwise, readthedocs.org uses their theme by default, so no need to specify it
```

About Read the Docs

4.1 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.

4.1.1 Current sponsors

- [Rackspace](#)
- You? ([Gittip](#))

All of the development of Read the Docs is funded on [Gittip](#). If you appreciate the service, please consider helping support development on Gittip.

4.1.2 Past sponsors

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

4.2 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

Operations Documentation

5.1 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.

5.1.1 Deploying Code

This uses the `fabfile.py` located in the root of project.

Pushing code to servers. This updates code & media:

```
fab push
```

Restart the webs:

```
fab restart
```

Restart the build servers celery:

```
fab celery
```

5.1.2 Deploying Nginx

This uses the fabfile located in `deploy/fab/fabfile.py` to deploy the nginx configs in `deploy/nginx/`.

To update the nginx configs:

```
fab nginx_configs
```

To reload nginx after the configs have been updated:

```
fab nginx_reload
```

5.1.3 Elastic Search Setup

```
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()
```

5.1.4 Servers

The servers are themed somewhere between Norse mythology and Final Fantasy Aeons. I tried to keep them topical, and have some sense of their historical meaning and their purpose in the infrastructure.

Domain

- readthedocs.com

Load Balancer (nginx)

- Asgard

Important Files

- /etc/nginx/sites-enabled/lb

Important Services

- nginx running from init

Restart

```
/etc/init.d/nginx restart
```

Web

- Chimera
- Asgard

Important Files

- /etc/nginx/sites-enabled/readthedocs
- /home/docs/sites/readthedocs.org/run/gunicorn.log

Important Services

- nginx running from init
- gunicorn (running from supervisord as docs user)

Restart

```
/etc/init.d/nginx restart
```

Build

- Build
- Bari

Important Files

- /home/docs/sites/readthedocs.org/run/celery.log

Important Services

- celery (running from supervisord as docs user)

Restart

```
supervisorctl restart celery
```

Database

- DB

Important Services

- Postgres running under init

Elastic Search

- DB
- Backup

Solr

- DB

Redis

- Build

5.1.5 Site Checkout

`/home/docs/sites/readthedocs.org/checkouts/readthedocs`

Bash Aliases

- `chk` - Will take you to the checkout directory
- `run` - Will take you to the run directory

b

`bookmarks.admin`, 40
`bookmarks.models`, 40
`bookmarks.urls`, 40
`builds.admin`, 40
`builds.models`, 40
`builds.urls`, 41
`builds.views`, 41

c

`core.admin`, 41
`core.forms`, 41
`core.management.commands.build_files`,
43
`core.management.commands.update_repos`,
43
`core.middleware`, 42
`core.models`, 42
`core.views`, 42

d

`doc_builder.backends.sphinx`, 41
`doc_builder.base`, 41

p

`projects.admin`, 43
`projects.constants`, 43
`projects.forms`, 43
`projects.models`, 43
`projects.search_indexes`, 45
`projects.tasks`, 45
`projects.utils`, 45
`projects.views.private`, 46
`projects.views.public`, 46

v

`vcs_support.base`, 46