Read The Docs Documentation

Release 1.0

Eric Holscher, Charlie Leifer, Bobby Grace

December 17, 2013

Contents

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 documentation for the site is organized into two different sections below. One is for users of readthedocs.org, that is the first section. The next section is for users of the code that powers the site. All of the RTD code is open source, so you can run your own instance. Presumably in an internal install inside your company, or something.

1.1

It's really easy to start using RTD for your project's documentation. This section shows you how.

If you are already using Sphinx for your docs, skip ahead to .

1.1.1

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 if you need help). Build them to see how they look:

```
$ make html
```

Edit and rebuild until you like what you see, then commit and/or push your changes to your public repository.

1.1.2

Sign up for an account and 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 keywords, then click “Create”.

Within a few minutes your code will automatically be fetched from your public repository, and the documentation will be built.

If you want to keep your code updated as you commit, configure your code repository to hit our Post Commit Hooks. Otherwise your project will get rebuilt nightly.

1.2

1.2.1 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 modules missing, you should enable the virtualenv feature, which will install your project into a virtualenv. If you are still seeing missing dependencies, you can install them with a pip requirements file in your project settings.

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

1.2.2 ?

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

1.2.3 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 %}
```

1.2.4 C

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

class Mock(object):
    def __init__(self, *args, **kwargs):
        pass

    def __call__(self, *args, **kwargs):
        return Mock()

    @classmethod
    def __getattr__(cls, name):
        if name in ('__file__', '__path__'):
            return '/dev/null'
        elif name[0] == name[0].upper():
            mockType = type(name, (), {})
            mockType.__module__ = __name__
            return mockType
        else:
            return Mock()

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

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

1.2.5 ?

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.

1.2.6 Blue/Default Sphinx

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.

1.2.7

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.

1.2.8

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

1.2.9 ?

This is something that has been long planned. In fact, we have a language string in the URLs! However, it isn't currently modeled and supported in the code base. However, you can specify the conf.py file to use for a specific

version of the documentation. So, you can create a project for each language of documentation, and do it that way. You can then CNAME different domains on your docs to them. Requests does something like this with it's translations:

- http://ja.python-requests.org/en/latest/index.html
- http://docs.python-requests.org/en/latest/index.html

1.3

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 readthedocs@librelist.org is also available for support.

1.4

Backwards Incompatible Changes will be emailed to the mailing list, readthedocs@librelist.org. 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

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.

1.5.1 Github

We now support linking to GitHub by default in the sidebar. It links to the page on GitHub, and directly links to the edit view as well. This should help people quickly update typos and send pull requests to contribute to project documentation.

If you want to integrate this 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 }}{{ co
    Show on GitHub</a></li>
  <li><a href="https://github.com/{{ github_user }}/{{ github_repo }}/edit/{{ github_version }}{{ co
    Edit on GitHub</a></li>
{% endif %}
```

1.5.2

The page talks about the different way 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.

1.5.3

We run Varnish in front of RTD, so a lot of the docs you look at will be served out of memory. This is really great for the "Look up and link" that happens a lot on IRC channels. The person who looks up the link will cache the page, and the person they link it to will get it served really quickly.

We also bust caches on all documentation on the RTD domain (not CNAMEs, yet) when you build your docs, so you shouldn't have problems with stale caches.

1.5.4

Versions are supported at the Version Control level. We support tags and branches that map to versions in RTD parlance. Not all version control systems are equally supported. We would love to accept patches from users of other VCS systems to gain equivalent features across systems.

1.5.5

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

1.5.6 PDF

When you build your project on RTD, we automatically build a PDF of your projects 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.

1.5.7

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.

1.5.8

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

1.5.9 Intersphinx

We host intersphinx catalogs for all projects built on Read the Docs. For more info on this support, read the Sphinx docs on `Intersphinx`. Your configuration should look something like this:

```
intersphinx_mapping = {
  'python': ('http://python.readthedocs.org/en/latest/', None),
  'django': ('http://django.readthedocs.org/en/latest/', None),
  'sphinx': ('http://sphinx.readthedocs.org/en/latest/', None),
    }
```

Then usage is pretty similar. You reference something using normal sphinx syntax, but can use the namespace of the project you want to reference, like so:

```
:mod:`Intersphinx <sphinx.ext.intersphinx>`
:mod:`Intersphinx <sphinx:sphinx.ext.intersphinx>`
```

This will create a link to the official Sphinx documentation for intersphinx.

More information can be found on Reinout van Rees' blog: http://reinout.vanrees.org/weblog/2012/12/01/django-intersphinx.html

1.6

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

1.6.1

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

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

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 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.

1.6.2

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

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

1.6.3

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

1.7

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.

1.7.1 Github

Github, :

- “settings”
- “Service Hooks”
- , “ReadTheDocs”
- “Active”
- “Update Settings”

1.7.2 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, click “POST”
- Put “http://readthedocs.org/bitbucket” as the URL
- Click “Save Settings”

1.7.3

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`.

1.8

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

1.8.1

Every project has a subdomain that is available to serve it’s 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

1.8.2 CNAME

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.readthedocs.org.
```

1.8.3 RTFD.org

You can also use <slug>.rtfd.org as a short URL for the front page of your subdomain'd site. For example, http://pip.rtfd.org redirects to it's documentation page. We're looking for more fun ways to use this domain, so feel free to suggestion an idea.

1.9

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.

1.9.1

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

1.10

1.10.1

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'
```

1.10.2

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.

1.10.3

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)
- dvipng
- graphviz
- libxslt1.1
- libxml2-dev

1.10.4

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.

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. Currently we only support Sphinx, but we are looking to expand this selection.

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 rsync'd across to our application servers from the build server. Once on the application servers, they are served from nginx and then cached in Varnish for a week. This varnish cache is pro-actively purged whenever a new version of your docs are built.

An example in code:

```
update_imported_docs(project, version)
(ret, out, err) = build_docs(project=project, version=version,
                                    pdf=pdf, man=man, epub=epub,
                                    record=record, force=force)
#This follows the builder workflow layed out below.
purge_version(version, subdomain=True,
                mainsite=True, cname=True)
```

1.10.5

The documentation build system in RTD is made plugable, 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 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)
```

1.11

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

CHAPTER 2

2.1

So you're thinking of contributing some of your time and design skills to Read the Docs? That's **awesome**. This document will lead you through a few features available to ease the process of working with Read the Doc's CSS and static assets.

To start, you should follow the instructions to get a working copy of the Read the Docs repository locally.

2.1.1

Once you have RTD running locally, you can open `http://localhost:8000/style-catalog/` for a quick overview of the currently available styles.

This way you can quickly get started writing HTML – or if you're modifying existing styles you can get a quick idea of how things will change site-wide.

2.1.2 Typekit

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.

2.1.3 Readthedocs.org

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.

2.1.4 Sphinx

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.

2.1.5

Contributions should follow the *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).

3.1

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
```

Create a folder in here, and clone the repository:

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

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

```
cd readthedocs.org
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
```

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 a project with the `update_repos` command:

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

3.1.1 Solr ()

Apache Solr is used to index and search documents.

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 xvzf 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 servelet 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.

3.1.2

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 either create new documentation, or import your existing docs provided that they are in a git or mercurial repo.

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.

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'.

3.2

3.2.1

- Internationalization (i18n)
- Screencast of "Hosting your docs in 1 minute"
- Write better API docs at http://docs.readthedocs.org/en/latest/api.html (https://github.com/deceze/Sphinx-HTTP-domain)

3.2.2 Sprintable

- Fast frontend for filtering docs as you type (api-driven)
- Inline commenting ala Django Book (http://ucomment.org/contents/)
- Add a widget for RTD like travis-ci
- **Add a way to track pageviews internal to RTD.**
  - Have a slug
  - Use the API to track this.
  - Count Pageviews, which domain/CNAME it was served on, version, etc.

3.2.3

- Have a way to hide the bottom right widget (slides away)
- Make a richer experience with the widget (eg. PDF downloads)

3.2.4 /

- Internationalization (i18n)
- Make hosting RTD internally more supported/easier
- Make the readthedocs.org front-end i18n

3.3 Read the Docs

You should read the Open Comparison Contributing Docs. They are some of the best I've seen, this document will some day copy their structure with example for Read the Docs, but until that day, I highly recommend reading theirs.

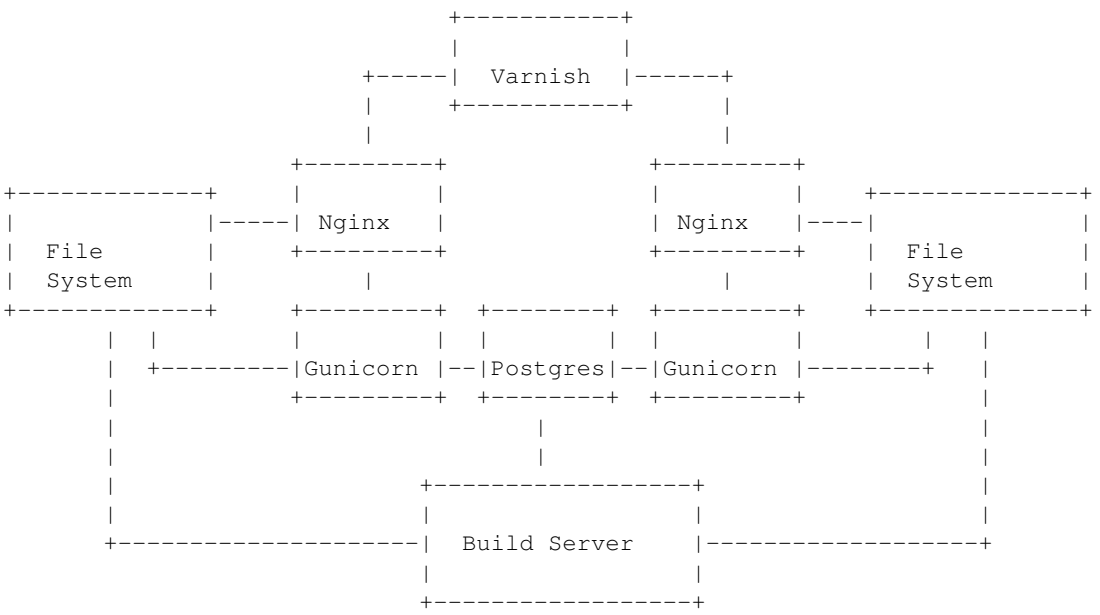
3.3.1

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

3.4

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. Varnish is the only real single point of failure currently, but we have plans to eliminate that as well.

3.4.1



3.5

3.5.1 USE_SUBDOMAIN

: `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 developemnt and other possible deployments, this might be False.

3.5.2 PRODUCTION_DOMAIN

: `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.

3.5.3 VARNISH_SERVERS

: `undefined`

This is a list of the varnish servers that you are using. It is used to perform cache invalidation. If this settings is not defined, no invalidation will be done.

3.5.4 MULTIPLE_APP_SERVERS

: `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.

3.5.5 SLUMBER_USERNAME

: `test`

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

3.5.6 SLUMBER_PASSWORD

: `test`

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

3.5.7 INDEX_ONLY_LATEST

: `False`

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

3.5.8 DOCUMENT_PYQUERY_PATH

: `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.

3.5.9 USE_PIP_INSTALL

: `False`

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

3.6

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.

3.6.1

The fine folks over at Django CMS have been nice enough to sponsor our CI setup on their hudson instance. You can check out the current build status: http://ci.django-cms.org/job/readthedocs.org/

3.7

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 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.

3.7.1

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

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 the 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.

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.

:

```
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')),
)
```

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")
```

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.

3.7.2

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 %}
```

3.7.3 Python

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

Strings in Python are more complex for two reasons:

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

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

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

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

Interpolation is done through normal Python string formatting:

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

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

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

Translator comments are normal one-line Python comments:

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

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

```
from django.utils.translation import ungettext

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

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()`.

3.8

3.8.1

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.

3.8.2 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.

3.8.3 Transifex

Transifex, `tx push -s` (for English) or `tx push -t <language>` (for non-English).

Transifex, `tx pull -a`. Transifex, (*MO*).

`tx` Transifex.

3.9 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.

3.9.1 Slumber API

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)
```

3.9.2

```
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']
```

3.9.3 API

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
```

3.9.4

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.

3.9.5

`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.

3.9.6

`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.

3.9.7

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.

3.9.8

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.

3.9.9

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.

3.9.10

`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.

3.9.11

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.

3.9.12

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.

3.9.13

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.

3.9.14

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.

3.9.15

```
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"
}
```


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"
}
```


`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..."
            },
            ...
        ]
    }
```


`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",
            ...
        ]
    }
```

3.10 API

Read The Docs API, .

3.10.1

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

bookmarks.views.**bookmark_add**(*request*, **args*, ***kwargs*)
Add a new bookmark for the current user to `url`.

bookmarks.views.**bookmark_remove**(*request*, **args*, ***kwargs*)
Remove the current user's bookmark to `url`.

bookmarks.views.**user_bookmark_list**(*request*, **args*, ***kwargs*)
Show a list of the current user's bookmarks.

3.10.2

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)

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

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

builds.views.**build_detail**(*request*, *project_slug*, *pk*)
Show the details of a particular build.

builds.views.**build_list**(*request*, *project_slug=None*, *tag=None*)
Show a list of builds.

3.10.3

doc_builder.base

class doc_builder.base.**BaseBuilder**(*version*)
The Base for all Builders. Defines the API for subclasses. All workflow steps need to return true, otherwise it is assumed something went wrong and the Builder will stop

build(*id=None*, ***kwargs*)
Do the actual building of the documentation.

changed
Says whether the documentation has changed, and requires further action.

This is mainly used to short-circuit more expensive builds of other output formats if the project docs didn't change on an update. Subclasses are recommended to override for more efficient builds.

Defaults to `True`

clean(***kwargs*)
Clean up the version so it's ready for usage.

This is used to add RTD specific stuff to Sphinx, and to implement whitelists on projects as well.

It is guaranteed to be called before your project is built.

force(**args*, ***kw*)
An optional step to force a build even when nothing has changed.

move(***kwargs*)
Move the documentation from it's generated place to its final home.

This needs to understand both a single server dev environment, as well as a multi-server environment.

doc_builder.backends

doc_builder.backends.sphinx

class doc_builder.backends.sphinx.**Builder**(*version*)
The parent for most sphinx builders.

Also handles the default sphinx output of html.

3.10.4

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=None*, **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

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.**github_build**(**args*, ***kwargs*)
A post-commit hook for github.

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 500 handler so we get media

core.views.**subdomain_handler**(*request*, *lang_slug=None*, *version_slug=None*, *filename=''*)
This provides the fall-back routing for subdomain requests.

This was made primarily to redirect old subdomain's to their version'd brothers.

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`.

3.10.5

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, version, copyright, theme, suffix, default_version, default_branch, requirements_file, documentation_type, analytics_code, path, conf_py_file, featured, skip, use_virtualenv, python_interpreter, use_system_packages, django_packages_url, crate_url, privacy_level, version_privacy_level)

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

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_doc_path(*version='latest'*)
The path to the documentation root in the project.

full_epub_path(*version='latest'*)
The path to the build latex docs in the project.

full_find(*file*, *version*)
A balla API to find files inside of a projects dir.

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

rtd_build_path(*version='latest'*)
The path to the build html docs in the project.

rtd_cname_path(*cname*)
The path to the build html docs in the project.

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.**update_docs_pull**(*record=False*, *pdf=False*, *man=False*, *force=False*)
A high-level interface that will update all of the projects.

This is mainly used from a cronjob or management command.

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.views

projects.views.public

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.**project_index**(*request*, *username=None*, *tag=None*)
The list of projects, which will optionally filter by user or tag, in which case a 'person' or 'tag' will be added to the context

projects.views.public.**search**(*request*)
our ghetto site search. see roadmap.

projects.views.public.**search_autocomplete**(*request*)
return a json list of project names

projects.views.public.**tag_index**(*request*)
List of all tags by most common

projects.views.public.**version_autocomplete**(*request*, *project_slug*)
return a json list of version names

projects.views.private

projects.views.private.**export**(*request*, **args*, ***kwargs*)
Export a projects' docs as a .zip file, including the .rst source

projects.views.private.**project_dashboard**(*request*, **args*, ***kwargs*)
A dashboard! If you aint know what that means you aint need to. Essentially we show you an overview of your content.

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.

3.10.6

watching.admin

watching.models

watching.urls

watching.views

3.10.7 vcs

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.

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

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).

4.1

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 of VPN, this is for you.

4.1.1

Read the Docs has a lot of that help customize your install. This document will outline some of the more useful ways that these can be combined.

If you put a file named `local_settings.py` in the `readthedocs/settings` directory, it will override settings available in the base install.

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 c
    '%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.

5.1

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

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.

- readthedocs.com

(nginx)

- Asgard

- /etc/nginx/sites-enabled/default

- nginx running from init

- Chimera
- Asgard

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

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

- Build

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

- celery (running from supervisord as docs user)

- DB

Solr

- DB

Redis

- DB

5.1.2

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

Bash

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

5.1.3

. & :

```
fab push
```

web:

```
fab restart
```

celery:

```
fab celery
```

Python Module Index

b

bookmarks.admin, ??
bookmarks.models, ??
bookmarks.urls, ??
bookmarks.views, ??
builds.admin, ??
builds.models, ??
builds.urls, ??
builds.views, ??

c

core.admin, ??
core.forms, ??
core.management.commands.build_files, ??
core.management.commands.update_repos, ??
core.middleware, ??
core.models, ??
core.views, ??

d

doc_builder.backends.sphinx, ??
doc_builder.base, ??

p

projects.admin, ??
projects.constants, ??
projects.forms, ??
projects.models, ??
projects.search_indexes, ??
projects.tasks, ??
projects.utils, ??
projects.views.private, ??
projects.views.public, ??

v

vcs_support.base, ??