eCultura Documentation

Release latest

mar. 23, 2017

Brukerdokumentasjon

eCultura er et digitalt rammeverk for idrett og kultur for kommuner. Dette rammeverket bygger på Wordpress Read the Docs‘_ hosts documentation for the open source community. We support Sphinx docs written with reStructuredText and CommonMark. We pull your code from your Subversion, Bazaar, Git, and Mercurial repositories. Then we build documentation and host it for you. Think of it as *Continuous Documentation

The code is open source, and available on GitHub.

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

- user-docs
- feature-docs
- about-docs

Information about development is also available:

- dev-docs
- design-docs

KAPITTEL 1

Hvordan komme i gang med eCultura

For å kunne ta i bruk eCultura må du registrere deg. Det er kun kommuner som kan registrere seg som bruker av eCultura.

eCultura er svært enkelt å sette opp. Før du begynner bør du ha klar 2-3 bilder som du vil vise på Forsiden. Du kan legge inn opp til 6 bilder på Forsiden. Disse bildene må ikke være for lyse. Se på eCultura sin side for å få en pekepinn.

Note: Når du laster opp bilder til eCultura må disse være mindre enn 1Mb. Du laster opp bilder fra Kontrollpanelet.

Import Your Docs.

Write Your Docs

You have two options for formatting your documentation:

- *In reStructuredText*
- *In Markdown*

In reStructuredText

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

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

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

Create a directory inside your project to hold your docs:

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

Run `sphinx-quickstart` in there:

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

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

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

```
$ make html
```

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

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

In Markdown

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

```
$ pip install recommonmark
```

Then in your `conf.py`:

```
from recommonmark.parser import CommonMarkParser

source_parsers = {
    '.md': CommonMarkParser,
}

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

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

Import Your Docs

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

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

- Bazaar: `lp:pasta`

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

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

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

We support multiple versions of your code. You can read more about how to use this well on our *Versions* page.

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

KAPITTEL 2

Versions

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

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

How we envision versions working

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

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

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

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

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

Redirects on root URLs

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

KAPITTEL 3

Build Process

Files: tasks.py - doc_builder/

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

- 15 minutes
- 1GB of memory

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

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

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

How we build documentation

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

Sphinx

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

Then Sphinx will build any files with an `.rst` extension. If you have a `README.rst`, it will be transformed into an `index.rst` automatically.

Mkdocs

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

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

Understanding what's going on

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

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

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

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

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

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

An example in code:

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

Builder Responsibility

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

Packages installed in the build environment

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

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

Writing your own builder

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

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

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

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

Deleting a stale or broken build environment

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

Build environment

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

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