The RestructuredText Book Documentation

Release 0.1

Daniel Greenfeld, Eric Holscher

Sep 27, 2017

Contents

RestructuredText (reST) is a markup language, it's name coming from that it's considered a revision and reinterpretation of two other markup languages, **Setext** and **StructuredText**.

Source: https://en.wikipedia.org/wiki/Restructured_Text

This is a syntax guide designed to provide very clear, understandable examples and tutorials of RestructuredText usage. It borrows from several sources including:

- https://en.wikipedia.org/wiki/Restructured_Text
- http://docutils.sourceforge.net/docs/index.html
- http://sphinx-doc.org/rest.html

RestructuredText:

CHAPTER 1

RestructuredText Tutorial

CHAPTER 2

RestructuredText Guide

Basics

Improving upon the pattern established at http://markdown-guide.readthedocs.org/en/latest/basics.html, sections to add:

- Code: Block
- Code: Inline
- Emphasis: Italics
- Emphasis: Strong
- Headers
- Horizontal rules
- Images: Inline
- Line Return
- Links: Inline
- Links: Inline with title
- Links: Reference
- Lists: Simple
- Lists: Nested
- Paragraphs
- Images: Reference

Blockquotes

To enclose a segment of text in blockquotes, one must add a tab at the start of a paragraph.

RestructuredText:

```
This is normal text.

    This is blockquoted text

    This is another paragraph of blockquoted text.

        This is a nested block of text.
```

Output:

```
<p>This is normal text.</p>
<blockquote>
<div><p>This is blockquoted text</p>
<p>This is another paragraph of blockquoted text.</p>
<blockquote>
<div>This is a nested block of text.</div></blockquote>
</div></blockquote>
</div>
```

Code: Block

TODO

CHAPTER 3

RestructuredText Customizations

Sphinx:

CHAPTER 4

Sphinx Tutorial

Step 1

Getting Set Up

Philosophy

Sphinx is what is called a documentation generator. This means that it takes a bunch of source files in plain text, and generates a bunch of other awesome things, mainly HTML. For our use case you can think of it as a program that takes in plain text files in reStructuredText format, and outputs HTML.

```
reST -> Sphinx -> HTML
```

So as a user of Sphinx, your main job will be writing these text files. This means that you should be minimally familiar with reStructuredText as a language. It's similar to Markdown in a lot of ways, if you are already familiar with Markdown.

Installing Sphinx

The first step is installing Sphinx. Sphinx is a python project, so it can be installed like any other python library. Every Operating System should have Python pre-installed, so you should just have to run:

```
sudo easy_install Sphinx
```

Note: Advanced users can install this in a virtualenv if they wish. Also, `pip install Sphinx` works fine if you have Pip.

Getting Started

Now you are ready to creating documentation. Create a directory called `crawler`. Inside that directory you should create a `docs` directory, and move into it:

```
mkdir crawler
cd crawler
mkdir docs
cd docs
```

Then you can create the Sphinx project skeleton in this directory:

```
sphinx-quickstart
```

accepting all the defaults, calling the project `Crawler`, and giving it a *1.0* version. Your file system should now look similar to this:

```
crawler/
    docs/
        conf.py
        index.rst
        Makefile
```

We have a top-level `docs` directory in the main project directory. Inside of this is:

`index.rst`: This is the index file for the documentation, or what lives at `/`. It normally contains a *Table of Contents* that will link to all other pages of the documentation.

`conf.py`: which allows for customization of Sphinx. You won't need to use this too much yet, but it's good to be familiar with this file.

`Makefile`: This ships with Sphinx, and is the main interface for local development, and shouldn't be changed.

`_build`: The directory that your output files go into.

`_static`: The directory to include all your static files, like images.

`_templates`: Allows you to override Sphinx templates to customze look and feel.

Building docs

Let's build our docs into HTML to see how it works. Simply run:

```
# Inside top-level docs/ directory.
make html
```

This should run Sphinx in your shell, and output HTML. At the end, it should say something about the documents being ready in `_build/html`. You can now open them in your browser by typing:

```
open _build/html/index.html
```

This should display a rendered HTML page that says **Welcome to Crawler's documentation!** at the top.

`make html` is the main way you will build HTML documentation locally. It is simply a wrapper around a more complex call to Sphinx.

Documenting a Project

Now that we have our basic skeleton, let's document the project. As you might have guessed from the name, we'll be documenting a basic web crawler.

For this project, we'll have the following pages:

- Index Page
- Support
- Installation
- Cookbook/Examples
- Command Line Options
- Changelog

Let's start with the Support page.

Support docs

It's always important that users can ask questions when they get stuck. There are many ways to handle this, but normal approaches are to have an IRC channel and mailing list.

Go ahead and put this markup in your `support.rst`:

```
=======
Support
=======

The easiest way to get help with the project is to join the ``#crawler``
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/crawler is also available␣
↪for support.

.. _Freenode: irc://freenode.net
.. _Github: http://github.com/example/crawler/issues
```

Hyperlink Syntax

The main new markup here is the link syntax. The link text is set by putting a _ after some text. The ` is used to group text, allowing you to include multiple words in your link text. You should use the `, even when the link text is only one word. This keeps the syntax consistent.

The link target is defined at the bottom of the section with `.. _<link text>:  <target>`.

Installation docs

Installation documentation is really important. Anyone who is coming to the project will need to install it. For our example, we are installing a basic Python script, so it will be pretty easy.

Include the following in your `install.rst`:

```
============
Installation
============

At the command line::

    easy_install crawler

Or, if you have virtualenvwrapper installed:

.. code-block:: bash

    mkvirtualenv crawler
    pip install crawler
```

Code Example Syntax

This snippet introduces a couple of simple concepts. The syntax for displaying code is `::`. When it is used at the end of a sentence, Sphinx is smart and displays one `:` in the output, and knows there is a code example in the following indented block.

Sphinx, like Python, uses meaningful whitespace. Blocks of content are structured based on the indention level they are on. You can see this concept with our `code-block` directive above.

Table of Contents Tree (toctree)

Now would be a good time to introduce the `toctree`. One of the main concepts in Sphinx is that it allows multiple pages to be combined into a cohesive hierarchy. The `toctree` directive is a fundamental part of this structure. A simple `toctree` directive looks like this:

```
.. toctree::
   :maxdepth: 2

   install
   support
```

This will then output a Table of Contents in the page where this occurs. It will output the top-level headers of the pages as listed. This also tells Sphinx that the other pages are sub-pages of the current page.

You should go ahead and include the above `toctree` directive in your `index.rst` file.

Build Docs Again

Now that you have a few pages of content, go ahead and build your docs again:

```
make html
```

If you open up your `index.html`, you should see the basic structure of your docs from the included `toctree` directive.

Aside: Other formats

Make a manpage

The beauty of Sphinx is that it can output in multiple formats, not just HTML. All of those formats share the same base format though, so you only have to change things in one place. So you can generate a manpage for your docs:

```
make man
```

This will place a manpage in `_build/man`. You can then view it with:

```
man _build/man/crawler.1
```

Create a single page document

Some people prefer one large HTML document, instead of having to look through multiple pages. This is another area where Sphinx shines. You can write your documentation in multiple files to make editing and updating easier. Then if you want to distribute a single page HTML version:

```
make singlehtml
```

This will combine all of your HTML pages into a single page. Check it out by opening it in your browser:

```
open _build/singlehtml/index.html
```

You'll notice that it included the documents in the order that your TOC Tree was defined.

Step 2

Referencing Code

Let's go ahead and add a cookbook to our documentation. Users will often come to your project to solve the same problems. Including a Cookbook or Examples section will be a great resource for this content.

In your `cookbook.rst`, add the following:

```
========
Cookbook
========

Crawl a web page
----------------

The most simple way to use our program is with no arguments.
Simply run::

        crawler <url>

to crawl a webpage.

Crawl a page slowly
-------------------

To add a delay to your crawler,
use :option:`-d`::
        crawler -d 10 <url>

This will wait 10 seconds between page fetches.

Crawl only your blog
--------------------

You will want to use the :option:`-i` flag,
which while ignore URLs matching the passed regex::

        crawler -i "^blog/" <url>

This will only crawl pages that contain your blog URL.


        Only crawl certain pages
        ------------------------

        You will want to use the :option:`-i` flag,
        which while ignore URLs matching the passed regex::

                crawler -i "pdf$" <url>

        This will ignore URLs that end in PDF.
```

CHAPTER 5

Sphinx Guide

CHAPTER 6

Sphinx Customizations

Utilities:

CHAPTER 7

Testing your Documentation

If you want to confirm that your docs build successfully, you add this to your tox.ini file:

```
[tox]
envlist = sphinx, readme

[testenv:sphinx]
deps =
    Sphinx
commands =
    make --directory=docs clean html
whitelist_externals =
    make

[testenv:readme]
deps =
    docutils
    pygments
commands =
    rst2html.py --exit-status=2 README.rst /tmp/README.html
    rst2html.py --exit-status=2 CONTRIBUTING.rst /tmp/CONTRIBUTING.html
```

TODO: Explain in depth what this actually does.

TODO: Explain what tox is.

CHAPTER 8

Indices and tables

- genindex
- modindex
- search

Index