We Have, We Need Documentation

Release 0.1b

WHWN contributors

May 27, 2013

CONTENTS

Learn about We Have, We Need

CHAPTER

ONE

LOOKING TO HELP?

Are you a coder? Want to make WHWN better? Then check out our contribution guidelines to learn how you can help improve WHWN.

1.1 Contribution Guidelines

Please be familiar with the certain programming languages of files before attempting to contribute to the code.

1.1.1 Expectations

1.1.2 Licensing

1.1.3 Governance

Code of Conduct

Try not to be a dick.

1.1.4 Getting Help

Click here to join our technical support chatroom. At least one of our many talented engineers is usually in here and would be happy to help you.

If nobody is in the chat room, tweet @wehaveweneed or @WesVetter and we'll get in there. If you'd prefer to send us an email, you can reach us at support@wehave-weneed.org

1.2 Submitting Bugs and Feature Requests

Before submitting a request, please search through the submitted requests to prevent creating any duplicates. If has been a similar request made, you can make a new request with a link back to the related request and also the differences and why the two requests are not duplicates.

1.2.1 Submitting a Bug Report

Once you have determined that you have found a bug that has not been mentioned in the Issue Tracker, go ahead to begin creating a new request. Step #: Create a good title, mentioning the feature of the bug and the error or deviation from the expected outcome. Step #: Add the label "bug" and also other relevant categories and labels available. Step #: State the feature the issue is about and list the deviation(s) about this issue. Step #: List steps on how to reproduce this error. Step #: (For developers) Provide as much information as you can about the conditions the error was encountered in. (for example: the operating system, browser, last pull request) Step #: Provide a console output at the end of the request.

1.2.2 Submitting a Feature Request

If you have a feature you think would be beneficial to add, feel free to make a request for it. Step #: Create a good title, mentioning the feature to be added and to where. Step #: Add the label "Feature Request" and also other relevant categories and labels available. Step #: State the feature idea, purpose, and reason it should be added. Step #: Give examples of this feature and how or why it would be beneficial.

1.3 How to Commit Code

Step #: create/modify/delete files in the repository Step #: use `git add [filename.extension]` to add the file to commit Step #: type `git status` to view files that are to be committed Step #: commit files with `git commit -m "[message about committed files]"` Step #: view the past commits with `git log` Setp #: push the commits with `git push localBranch repositoryBranch`, which updates the branch in the repository with the current local branch git is on.

1.3.1 Commit Guidelines

What: Commit the files that have been changed When: Commit only after making significant changes to code. How: Add a good description about the changes to code and files being committed.

1.3.2 Picking Something to Work On

Look on GitHub Issue Tracker(issues tab) for files to work on. Once issue to be worked on has been selected, confirm or search for other users who are also working on the same or similar issue and comment on the issue to inform others that the issue will be worked on.

1.3.3 Making a Pull Request

Type `git pull` to update all the files in the home directory with the files in the repository. If there have been files that are modified, the files changes need to be removed by committing or stashing. To stash, type `git stash`.

1.3.4 Code Review

Committed code will be displayed in the github feeds, for example, in the hipchat room. The code will be made available for others to view, comment, and approved.

1.4 Setting up the Development Environment

Connecting with github project

1.4.1 Clone The Repository

```
$ git clone le_url.git
```

This will clone the repository to your local machine for the virtual machine to run the code.

1.4.2 Setup the VM

The easiest way to get set up is with the virtual machine. Mostly.

install virtual box & vagrant

How to setup VM: On Windows:

On Linus/OSx/Ubuntu

installing it locally

how to install locally

1.4.3 Start Django

```
$ vagrant up
$ fab serve
```

In a browser, go to localhost:8000/ profit

TODO: seeding & fab file

1.4.4 Recombiling CSS server files

```
$ fab guard
```

1.4.5 End Server

```
$ vagrant halt
```

Type the command to end the virtual machine

1.5 How to Write Tests

1.5.1 When to Write Tests

Always son.

1.5.2 How to Write Tests

Carefully.

1.5.3 Continuous Integration

Talk to Travis.

1.6 Writing Documentation

Documentation falls into two categories: documentation for **users** and documentation for **developers**. Developer documentation can be further broken down into **in-line** and **standalone** documentation. Here we focus on standalone documentation for developers. For information on in-line documentation, visit here <www.ihavenotdonethis.wat>.

Note that while it is not necessary for developers to contribute standalone documentation, we will **NOT** accept pull requests without sufficient in-line documentation.

1.6.1 General guidelines

When writing any kind of documenation, in-line or standalone, please try to adhere to the following conventions:

- Please use proper grammar and spelling conventions.
- When writing & commenting Python code, try to adhere to PEP-8 standards.
- Try to write as simply as possible, assume that non-native English speakers may be reading.

1.6.2 Sphinx

To generate the standalone documentation, we use a library called Sphinx. You can build the documentation locally or read it online at ReadTheDocs.

Sphinx conventions

When working on standalone documentation, please use the following conventions:

- In section titles, capitalize only initial words and proper nouns.
- Whenever possible, wrap lines at 80 characters.
- In the interest of readability, seperate links and their target definitions.
- **We Have, We Need** - Always capitalized in documentation, can be lowercase in code. Can be abbreviated **WHWN**.

Directory structure

All documentation can be found in the `docs/` folder of the root application. Documentation intended for developers is contained in the `docs/developer/` folder. Each HTML page is built from an .rst file in the `docs` folder.

Building the docs

To build the HTML documentation, run `make html` from the `docs/` directory. This will generate HTML in `docs/_build`.

reStructuredText

Sphinx uses reStructuredText files to build HTML documentation. Every standalone HTML page is built from an `.rst` file. Each `.rst` file contains plain-text that is parsed and used to generate HTML.

Additional resources

Use these resources to assist in documentation:

For a quick reference, see *A beginner's guide to reStructuredText*.

A Sphinx primer

1.7 Coding Conventions

Use vim to edit all your code.

1.7.1 Our Conventions

If possible, join our weekly programming Sprints on Saturday mornings. You can connect with us on the Tech Support room in our HipChat room.

1.7.2 Django Conventions

1.7.3 Python Conventions

1.8 A beginner's guide to reStructuredText

This is a quick reference for reStructuredText (reST). It covers some of the most commonly used features. The following resources are much more comprehensive and come straight from the authoritive reST sources:

- Sphinx - reST Primer
- Docutils - A reST Primer
- Docutils - the complete reST markup specification

If you are using Vim, you may want to look at *.vimrc configurgation* to make writing documentation easier.

1.8.1 Sections

When creating a new section, make sure that the underline (and overline if necessary) have the same number of characters as the section title.

```
==============
Section Header
==============

Section
=======

Subsection
----------

Subsubsection
^^^^^^^^^^^^^

Paragraphs are generated by one or more blank lines. All lines of the
paragraph must have the same level of indentation.
```

1.8.2 Lists

```
* This is an unorderded (bulleted) list.
* It has two items, the second item spans
  multiple lines and has to be appropriately indented.

#. This is a numbered list of two items.
#. The numbers are automatically generated.
```

1.8.3 Styling

```
*emphasis (italics)*
**emphasis (boldface)**
``code``
```

1.8.4 Code blocks

Code blocks are formed the following way:

```
::

    This text will appear in a code block.
    This inner text must be indented and seperated by a newline.
```

1.8.5 Internal hyperlinks

To make an internal link, first define a label above the section/header you want to link to.

```
.. _widget-making:

How to make widgets
-------------------
```

Then, link to it using `:ref:`my-label-name`:

```
For more information on widget building, check out :ref:`widget-making`.
```

1.8.6 External hyperlinks

External links are formed like this: `` `this text is a link <http://www.example.com>`_ ``

When linking to external sources, seperate the link from the target definition. This means the the link title and the address are seperate, which makes the documentation more readable from the text editor. Put all the target definitions at the bottom of the section or file. The following is an example.

Example: If you want to search for things, you should go to Google. Google is really fast!

```
If you want to search for things, you should go to `Google`_. Google is
really fast!

.. _Google: http://www.google.com
```

Anonymous links can also be used with the following syntax. In some cases, this style is cleaner to use. The targets of each label are assigned to the anonymous links sequentially.

```
For my searching needs, I use `Google`__.
Whenever I am looking for the weather I go to `Yahoo!`__.
And if I am want the latest movie news, I check out `Rotten Tomatoes`__.

__ http://www.google.com
__ http://www.yahoo.com
__ http://www.rottentomatoes.com
```

Note: Note that there are **two** underscores following anonymous links/labels.

1.8.7 Admonitions

See Also:

this is a test of the **seealso** directive

Note: this is a test of the **note** directive

Warning: this is a test of the **warning** directive

These can be generated with the following code:

```
.. seealso:: this is a test of the **seealso** directive
.. note:: this is a test of the **note** directive
.. warning:: this is a test of the **warning** directive
```

1.8.8 .vimrc configurgation

If you use Vim as your text editor, edit your `.vimrc` to include the following macros for quickly making section headings.:

```
" reStructuredText files
au BufRead,BufNewFile *.rst set textwidth=80
let @h='yypVr=yykP'    " makes a section header (with overline)
let @o='yypVr='        " makes a section heading
let @i='yypVr-'        " makes a subsection heading
let @u='yypVr^'        " makes a subsubsection heading
let @f='gq}'           " format until next paragraph, fixes column widths
```

Not a developer? Well you still might be able to help. Designers, promoters, translators, and more, are welcome to join in on the fun. Look at the links below for ways to contribute without writing code.

1.9 Help out with non-coding tasks

1.10 Designers Wanted

1.11 Help Promote We Have, We Need

1.12 Translate We Have, We Need

CHAPTER

TWO

INDICES AND TABLES

- *genindex*
- *modindex*
- *search*