How to Document in RTD

Release 0.0.1

Bibhash Mitra, Prasang Gupta

Jul 20, 2022

CONTENTS

This document is a **templatised** form for a quick documentation of any type of python module. This serves as a quick documentation solution for a python module / toolkit (*python module* in this case) hosted on Github. Apart from this, it also serves as a one stop solution for understanding the documentation process in RTD format.

Check out the *Installation* section for instructions on installation of required packages.

To add a Github-hosted documentation (in RTD format) to your module / toolkit, check out the *Usage* section for details and steps.

Feel free to reach out to the authors for any additional information:

Bibhash C Mitra, Prasang Gupta

CHAPTER

ONE

INSTALLATION

Packages required for using this repo can be installed as follows:

```
$ pip install sphinx sphinx_rtd_theme sphinx-autodoc-typehints nbsphinx
```

See also:

Sphinx
: Understanding sphinx

Sphinx Read the docs theme
: Understanding how to use the read the docs theme for documentation

CHAPTER

TWO

USAGE

In this section, we will discuss the implementation details of the documentation. This includes getting the documentation ready locally and hosting it on github.

We offer 2 solutions for this exercise:

- An *In-depth implementation* using `sphinx-quickstart`. This would include you setting a lot of parameters with provided explanations for your use case.

  (Recommended if you want to implement and learn at the same time and if you have some spare time on your hands OR if you already are a PRO)

- A *Quick-and-Dirty implementation* to get your documentation up and running. This would involve modifying an already implemented docs code swapping out sections with your case-specific details.

  (Recommended if you don't really care about what is happening in the background OR simply don't have time)

Before proceeding with any of the steps, make sure that you have installed the necessary packages from the *Installation* page.

2.1 In-depth implementation

2.1.1 Generate basic documentation

1. Ensure that your repository is structured as

```
|_ module
|   |_ __init__.py
|   |_ some_file.py
|   |_ ....
|_ setup.py (optional)
|_ README.md
|_ ....
```

Note: Make sure that `__init__.py` file is present in each directory within the main module.

Note: The auto-summary requires proper docstrings to be written. Functions without docstrings would be ignored.

2. Create a folder named `docs` in the root directory, so that the structure of repo now is

```
|_ docs
|_ module
|   |_ __init__.py
|   |_ some_file.py
|   |_ ....
|_ setup.py (optional)
|_ README.md
|_ ....
```

3. Run `sphinx-quickstart` from inside the `docs` folder followed by `make html`. This will auto-generate all necessary files required for a static documentation website.

```
$ cd docs
$ sphinx-quickstart

> Separate source and build directories (y/n) [n]: y   # This helps in␣
↪organising the generated files into separate folders
> Project name: <Your Project name goes here>
> Author name(s): <Comma separated author names go here>
> Project release []: <Version of the module goes here>

$ make html
```

Files/Folders generated and their functions are listed below

build
: This folder contains HTML files for the static website.

source
: This folder contains all the source files needed to build the documentation (rst and configuration files)

Makefile
: Makefile for Linux/MacOS and Windows are provided to build the static documentation files (**build**) from the source files (**source**).

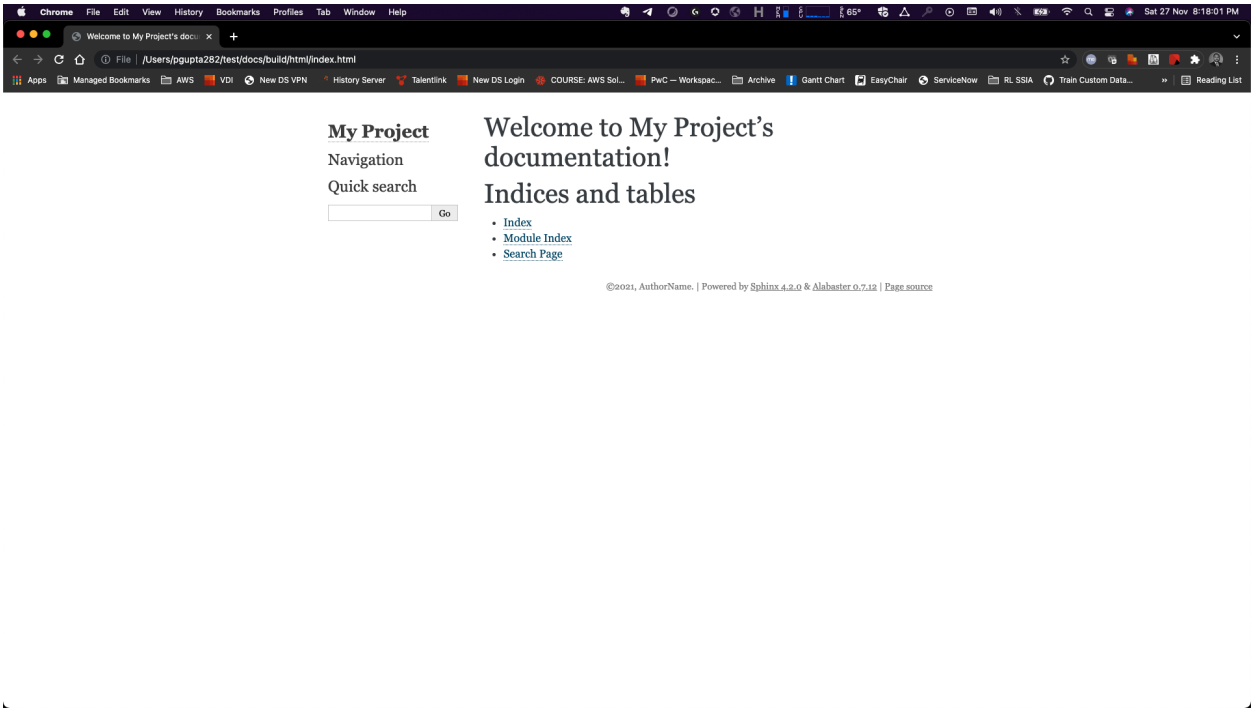
4. At this stage, your documentation would look like this: (You can check by opening `build/html/index.html`)

2.1.2 Modifying configuration file

1. If you have installed the python module i.e. if you have installed it via `python setup.py install` you **don't have to do this** but if you have not installed it then you have to add the following lines to your `conf.py` so that it can find your code references:

```
import os
import sys
sys.path.insert(0, os.path.abspath('./../../'))
```

Note: The abspath needs to be provided for the root of the repository (where the module folder is



present) which in our case is `'./../../'`. This is needed for the autodoc to auto-generate documentation for the code part.

2. Change the value of copyright variable

```
copyright = '2021, Emerging Technologies'
```

3. Add the required extensions in the `extensions` section

```
extensions = [
    'sphinx.ext.autodoc',           # for generating documentation from␣
↪docstrings
    'sphinx.ext.autosummary',       # generate summaries for autodoc
    'sphinx.ext.autosectionlabel',  # allow reference sections to use titles
    'sphinx.ext.intersphinx',       # link to other project documentations
    'sphinx.ext.viewcode',          # add links to python source code for␣
↪documentation
    'sphinx_autodoc_typehints',     # automatically document param types
    'nbsphinx',                     # integrate with jupyter notebooks
    'sphinx.ext.napoleon',          # support for numpy and google␣
↪docstrings
    'sphinx.ext.coverage',          # collect document coverage
    'sphinx_rtd_theme'              # RTD theme
]
```

Warning: We have provided here the extensions that would be enough for most purposes. However, for advanced modifications, you may need to include additional extensions.

Sphinx's extension page has more extensions and details about each if you want to get creative !

4. Add some variables

```
intersphinx_mapping = {                          # documents to be included for␣
↪intersphinx functionality
    "python": ("https://docs.python.org/3/", None),
}
autosummary_generate = True                      # turn on autosummary generation
autosummary_generate_overwrite = True            # turn on overwriting for␣
↪subsequent builds
autoclass_content = "both"                       # add __init__ doc (ie. params)␣
↪to class summaries
html_show_sourcelink = False                     # remove 'view source code' from␣
↪top of page (for html, not python)
autodoc_inherit_docstrings = True                # if no docstring, inherit from␣
↪base class
set_type_checking_flag = True                    # enable 'expensive' imports for␣
↪sphinx_autodoc_typehints
nbsphinx_allow_errors = True                     # continue through Jupyter␣
↪errors
add_module_names = False                         # remove namespaces from class/
↪method signatures
```

5. Update the theme by replacing the

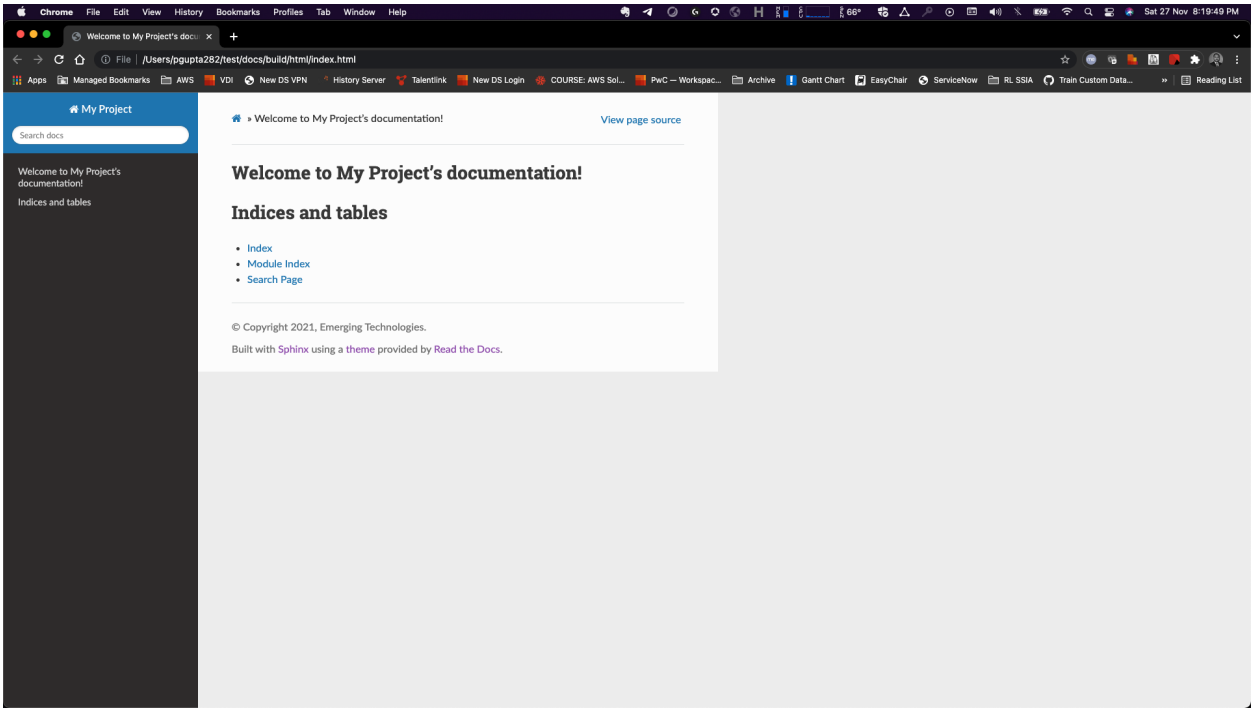
```
html_theme = 'alabaster'
```

line of code with

```
try:
    import sphinx_rtd_theme
    html_theme = "sphinx_rtd_theme"
    html_theme_path = [sphinx_rtd_theme.get_html_theme_path()]
    html_css_files = ["readthedocs-custom.css"]
except:
    html_theme = 'alabaster'
```

This would render the documentation in the RTD theme. If however, some error is encountered in loading the theme, it would fall back to the `alabaster` theme supported out of the box by Sphinx.

6. Update your documentation by running the `make html` command again from the `docs` directory. At this stage, your documentation would look like this: (You can check by opening `build/html/index.html`)



2.1.3 Modifying Makefile

There are two makefiles which generate after you run `sphinx-quickstart`

- **Makefile**: This makefile is used in Linux or MacOS
- **make.bat**: This makefile is used in Windows

So based on which Operating System you are on, you need to modify the corresponding file

Linux or MacOS

In this case we need to replace the line below `%: Makefile` section so that it becomes:

```
@$(SPHINXBUILD) -M $@ "$(SOURCEDIR)" "$(BUILDDIR)" $(SPHINXOPTS) $(O); touch .
↪nojekyll; echo '<meta http-equiv="refresh" content="0; url=./build/html/
↪index.html" />' > index.html
```

> **Warning:** You might run into an error saying: `make: *** No rule to make target `html'. Stop`. If this happens, you need to just correct the indentation and make sure that it starts with a tab rather than spaces. (This is a GNU make dependency)

Windows

In this case we need to add two lines before `goto end` so that it becomes:

```
%SPHINXBUILD% -M %1 %SOURCEDIR% %BUILDDIR% %SPHINXOPTS% %O%
type NUL > .nojekyll
echo ^<meta http-equiv="refresh" content="0; url=./build/html/index.html" /^> ␣
↪> index.html
goto end
```

Note: Explanation for adding these two lines:

- As we are not using *Jekyll* theme in our project we need to create a `.nojekyll` file in the `/docs` folder
- Now as we have added a `.nojekyll` file, *github pages* will try to find a `.html` file in the root directory (i.e. `docs` which we will select in *How to host on Github Pages ?*, but our page is present in `build/html/index.html`. So we need a helper page which can redirect to the main build page in `docs` folder. For that we create a new `index.html` file in `docs` folder.

2.1.4 Adding rst files

1. Create a file `api.rst` inside the `source` directory with the following content. This file is responsible for creating the autogenerated documentation for the python module.

```
API
======

.. autosummary::
    :toctree: _autosummary
    :template: custom-module-template.rst
    :recursive:

    module
```

Note: Replace `module` with the directory name of your python module / toolkit

---

Warning: We recommend using a template for docstring documentation as default function documentation is not very readable. You can copy the `source/_templates` folder from the parent repository of this documentation or you can use this drive link to manually download the folder and put it in the `source` directory. If you do not plan on using this template, REMOVE THIS LINE: `:template: custom-module-template.rst` from the `api.rst` file.

2. It is recommended that there are certain sections that are added to the documentation. We recommend to add the following sections and hence, create an `.rst` file for each in the same `source` directory.

   `installation.rst`
   Steps to install the dependencies and the package itself with warnings and solutions to common installation problems

changelogs.rst
Changelogs is an important part of version management to allow rollbacks and efficient debugging

references.rst
References used for the package development with URLs

Each of these files will be of the form

```
HEADING
=======

Content here in appropriate format
```

3. Replace the contents of the `index.rst` file with the following. This is done to adhere to a standard style of formatting the main file and section it in a readable format.

```
.. toctree::
    :hidden:
    :maxdepth: 3

    Home <self>
    Installation <installation>
    Module / Toolkit <_autosummary/module>
    Changelogs <changelogs>
    References <references>

NAME OF THE TOOLKIT
===================

Content to be shown on the main documentation page

...

Authors : Author 1, Author 2, ...
```

- This section here starts the Sphinx TOC tree. The `maxdepth` parameter sets the maximum depth for the tree.

```
.. toctree::
    :hidden:
    :maxdepth: 3
```

- This section lists the entries to populate the left pane of the documentation. The format followed here is `XXX <YYY>` where `XXX` is the name that will be displayed on the left pane and `YYY` is the name of the `.rst` file present in the `source` directory which will be displayed when a user goes to that section.

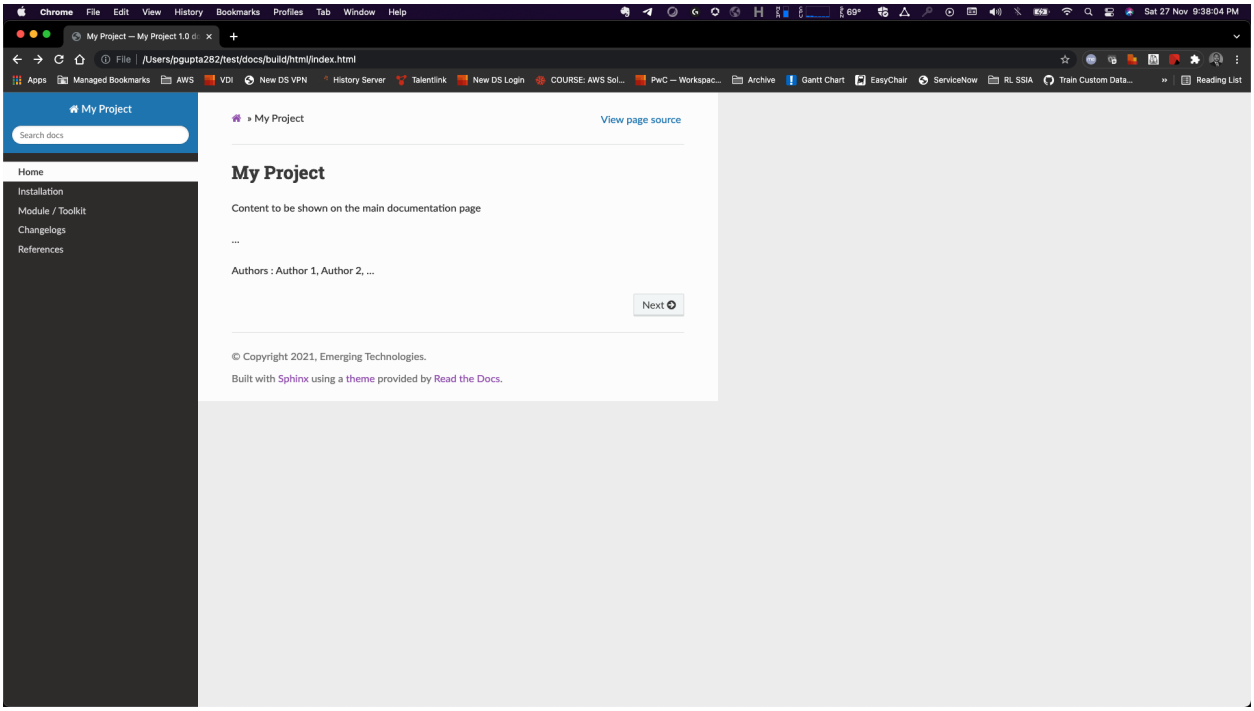
```
Home <self>
Installation <installation>
Module / Toolkit <_autosummary/module>
Changelogs <changelogs>
References <references>
```

Note: Include any and all sections that need to be added based on the format.

The `_autosummary/module` format is for the autogenerated docstrings documentation where `module` needs to be replaced with the name of the python toolkit directory.

- The rest of the section follows the same structure as the rest of the `.rst` files.

4. Update your documentation by running the `make html` command again from the `docs` directory. At this stage, your documentation would look like this: (You can check by opening `build/html/index.html`)



2.2 Quick-and-Dirty implementation

2.2.1 How to use the content of this repository directly

If you do not care about what is happening in the background and want a very rapid documentation with all the basic components, the following steps will help.

- **Step 1**
  The structure of your repository should have the following structure. For this example we will call this module **abracadabra**.

  Before

```
|_ abracadabra
|   |_ __init__.py
|   |_ some_file.py
|   |_ ....
|_ setup.py (optional)
|_ README.md
|_ ....
```

Now copy the `docs` folder of this repository in the root directory of your repository containing the python module. So the structure of the repository will become

After

```
|_ docs
|_ abracadabra
|   |_ __init__.py
|   |_ some_file.py
|   |_ ....
|_ setup.py (optional)
|_ README.md
|_ ....
```

- **Step 2**
  Delete all the `.rst` files inside `docs/source/_autosummary` as it contains the documentation of the python module of this repository.
- **Step 3**
  - Inside `docs/source/conf.py` you can find a section called **SECTION TO EDIT** where you can edit the fields according to your project
    - project
    - copyright
    - author
    - release

    Warning: If you have installed the module with `python setup.py install` then you need to remove `sys.path.insert(0, os.path.abspath('./../../'))` from `docs/source/conf.py`. This is to ensure there is no conflict between the installed module and the module in this repository.
  - Inside `docs/source/index.rst` change:
    - Content of the introductory section
    - `Python module` to `abracadabra`
    - <_autosummary/module> to <_autosummary/abracadabra>
  - Inside `docs/source/api.rst` change:
    - `module` to `abracadabra` (i.e. the name of your module or the folder from where autosummary will start generating automated documentations)
  - Inside `docs/source/installation.rst` add the steps of your installation.
  - Inside `docs/source/usage.rst` delete everything and add instructions on how to use your repository.

    Note: If you do not want any **Usage** section in your repository, you can remove this `usage.rst` file and remove the line `Usage <usage>` from `index.rst`.
  - Inside `docs/source/changelogs.rst` and `docs/source/references.rst` add your changelogs and references respectively.

- **Step 4**
  - Now we need to run the make command to build the html files:

```
$ cd docs
$ make clean  # This step is to remove all the prebaked files and␣
↪folders which are not necessary
$ make html
```

After you complete the following you can see the documentation if you open `docs/index.html` in any supported browser.

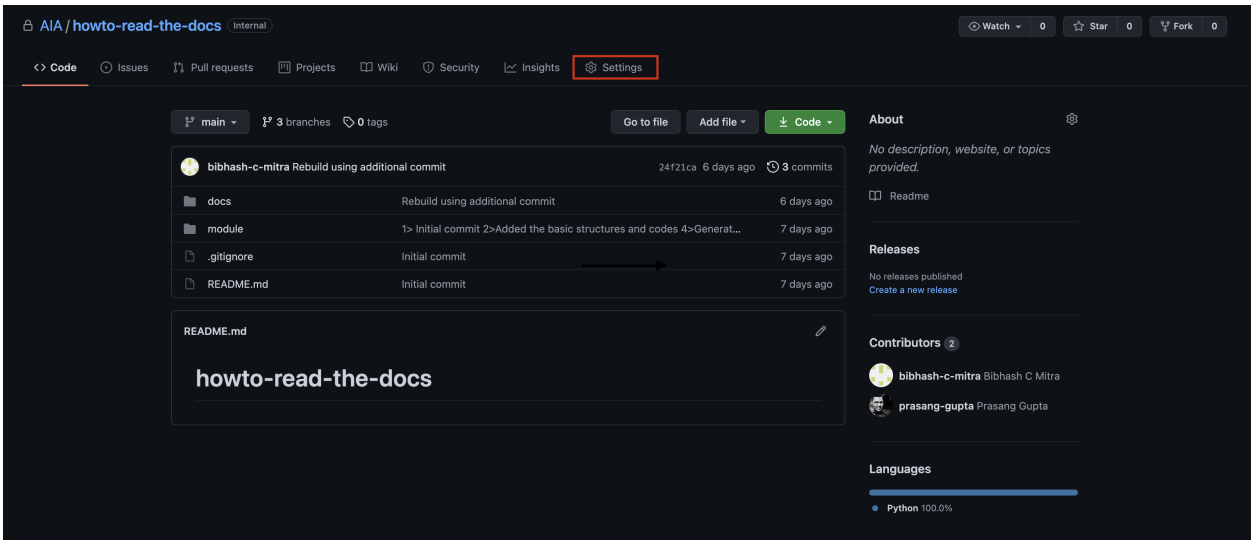
2.3 How to host on Github Pages ?

To host this page on github pages we have to follow the following steps:

- Push all the files to `github.pwc.com`

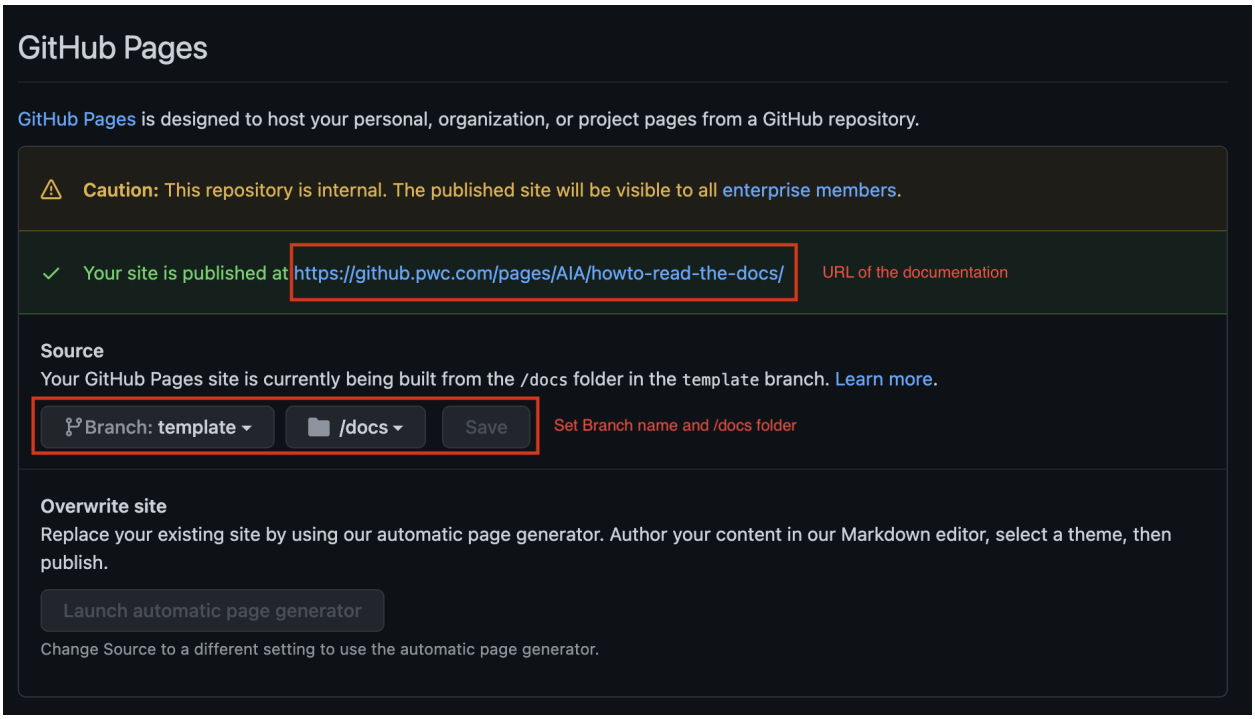
```
$ git add docs/*
$ git add docs/.nojekyll
$ git commit -m <message>
$ git push
```

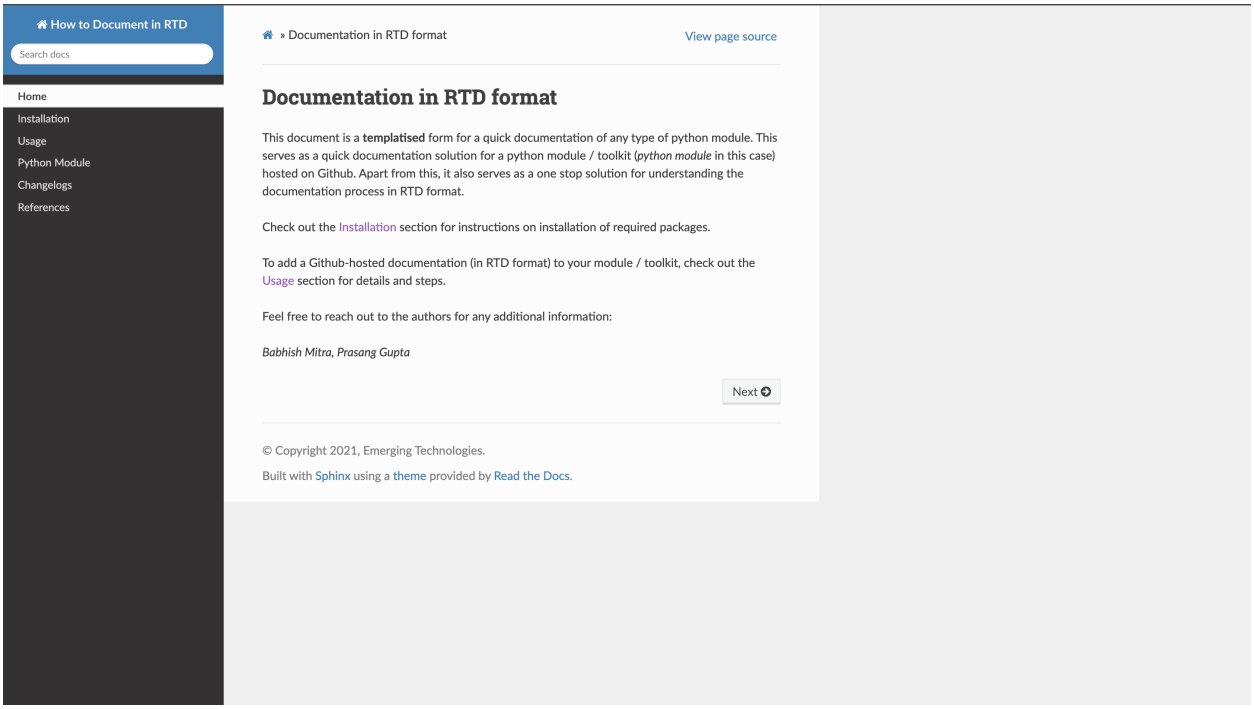
- Go to the `Settings` tab on the upper right section of your github repository



- Go down to the GitHub Pages section
- In the `Source` section select the `branch` (whichever branch you pushed the files in the previous step) and after that choose the `/docs` folder. Then click on `Save`
- Now an URL will be generated with a note `Your site is published at` <URL>, Click on the same.

- Voila ! Your documentation is ready.


GitHub Pages
GitHub Pages is designed to host your personal, organization, or project pages from a GitHub repository.
Caution: This repository is internal. The published site will be visible to all enterprise members.
Your site is published at https://github.pwc.com/pages/AIA/howto-read-the-docs/
URL of the documentation
Source
Your GitHub Pages site is currently being built from the /docs folder in the template branch. Learn more.
Branch: template
/docs
Save
Set Branch name and /docs folder
Overwrite site
Replace your existing site by using our automatic page generator. Author your content in our Markdown editor, select a theme, then publish.
Launch automatic page generator
Change Source to a different setting to use the automatic page generator.


How to Document in RTD
Search docs
Home
Installation
Usage
Python Module
Changelogs
References
» Documentation in RTD format
View page source
Documentation in RTD format
This document is a templatised form for a quick documentation of any type of python module. This serves as a quick documentation solution for a python module / toolkit (python module in this case) hosted on Github. Apart from this, it also serves as a one stop solution for understanding the documentation process in RTD format.
Check out the Installation section for instructions on installation of required packages.
To add a Github-hosted documentation (in RTD format) to your module / toolkit, check out the Usage section for details and steps.
Feel free to reach out to the authors for any additional information:
Babhish Mitra, Prasang Gupta
Next
© Copyright 2021, Emerging Technologies.
Built with Sphinx using a theme provided by Read the Docs.

2.4 How to Document?

In this section we have consolidated all the most commonly used sphinx syntaxes used in various parts of documentation. You can choose a snippet and add it in your documentation.

2.4.1 Inline formattings

- one asterisk: `*text*` for Italic. *text*
- two asterisks: `**text**` for Bold. **text**
- backquotes: ``` ``text`` ``` for code samples. `text`

2.4.2 Numbered and Bullet list

1. This is a numbered list.
2. It has two items too.
3. This is a numbered list using #

- First bullet
- Second bullet
  - A nested list with a space above
  - and some subitems
- **Third bullet**
    - Nested list without space

2.4.3 Hyperlinks

This is an hyperlink

This is an internal reference *Inline formattings*

2.4.4 Code blocks

Adding `::` at the end of paragraph and indenting them with *4 spaces* results in a code block as below

```
Code block line 1
Code block line 2
```

2.4.5 Tables

Grid Table

Header row, column 1 (header rows optional)	Header 2	Header 3	Header 4
body row 1, column 1	column 2	column 3	column 4
body row 2	…	…	

Simple Table

A	B	A and B
False	False	False
True	False	False
False	True	False
True	True	True

2.4.6 Sections

This is a heading:

```
This is a Heading
=================
```

2.5 This is a subheading

2.5.1 This is a subsubheading

This is a paragraph

This is a part

This is a chapter

Note: If a Heading is defined it will add a section in the side bar, so it is enclosed in a block in the above example

2.5.2 Roles

- *emphasis*
- **strong**
- `literal`
- subscript text
- superscript text
- *for titles of books, periodicals, and other materials*

- $a^2 + b^2 = c^2$.
- *Start* → *Programs*

For more **roles** please visit here

2.5.3 Directives

This is a figure.

Fig. 1: This is the caption
This is a legend

This is a code block

```
def my_function():
    "Loren Ipsum"
    print("Hello World")
```

This is a math block

$$_t(i) = P(O_1, O_2, \ldots O_t, q_t = S_i)$$

This is a note

Note: This is a note

This is a warning

Warning: This is a warning

- This is to add a version

  New in version 2.5: The version is 2.5

- This is to note down version changes

  Changed in version 2.6: The version changed

- This is to show deprecated functions

  Deprecated since version 3.1: Use `demo()` instead.

- This is a glossary

  Term 1
  Definition of term 1.

  Term 2
  Definition of term 2.

For more **directives** please visit here

CHAPTER

THREE

MODULE

module.dummycode	This script contains some dummy codes with their respective docstrings to use them for templatisation
module.submodule	

3.1 module.dummycode

This script contains some dummy codes with their respective docstrings to use them for templatisation

Functions

some_function	Summary or Description of the Function
square	Returned argument a is squared.
string_reverse	Returns the reversed String.

3.1.1 module.dummycode.some_function

some_function(*argument1*)

Summary or Description of the Function

Parameters
argument1 (*int*) – Description of arg1

Returns
Returning value

Return type
int

3.1.2 module.dummycode.square

square(*a*)

Returned argument a is squared.

3.1.3 module.dummycode.string_reverse

string_reverse(*str1*)

Returns the reversed String.

Parameters
str1 (*str*) – The string which is to be reversed.

Returns
The string which gets reversed.

Return type
reverse(str1)

Classes

GoogleVehicle	The Vehicle object contains a lot of vehicles
NumpyVehicle	The Vehicles object contains lots of vehicles
SphinxVehicle	The Vehicle object contains lots of vehicles

3.1.4 module.dummycode.GoogleVehicle

class GoogleVehicle(*arg*, **args*, ***kwargs*)

Bases: object

The Vehicle object contains a lot of vehicles

Parameters

- **arg** (*str*) – The arg is used for...
- ***args** – The variable arguments are used for...
- ****kwargs** – The keyword arguments are used for...

arg

This is where we store arg,

Type
str

Methods

cars	We can't travel distance in vehicles without fuels, so here is the fuels

cars(*distance*, *destination*)

We can't travel distance in vehicles without fuels, so here is the fuels

Parameters

- **distance** (*int*) – The amount of distance traveled
- **destination** (*bool*) – Should the fuels refilled to cover the distance?

Raises
RuntimeError – Out of fuel

Returns
A car mileage

Return type
cars

3.1.5 module.dummycode.NumpyVehicle

class NumpyVehicle(*arg*, **args*, ***kwargs*)

Bases: object

The Vehicles object contains lots of vehicles

Parameters

- **arg** (*str*) – The arg is used for ...
- ***args** – The variable arguments are used for ...
- ****kwargs** – The keyword arguments are used for ...

arg

This is where we store arg,

Type
str

Methods

cars	We can't travel distance in vehicles without fuels, so here is the fuels

cars(*distance*, *destination*)

We can't travel distance in vehicles without fuels, so here is the fuels

Parameters

- **distance** (*int*) – The amount of distance traveled
- **destination** (*bool*) – Should the fuels refilled to cover the distance?

Raises
RuntimeError – Out of fuel

Returns
A car mileage

Return type
cars

3.1.6 module.dummycode.SphinxVehicle

class SphinxVehicle(*arg*, **args*, ***kwargs*)

Bases: object

The Vehicle object contains lots of vehicles

Parameters

- **arg** (*str*) – The arg is used for ...
- ***args** – The variable arguments are used for ...
- ****kwargs** – The keyword arguments are used for ...

Variables
arg (*str*) – This is where we store arg

Methods

cars	We can't travel a certain distance in vehicles without fuels, so here's the fuels

cars(*distance*, *destination*)

We can't travel a certain distance in vehicles without fuels, so here's the fuels

Parameters

- **distance** – The amount of distance traveled
- **destinationReached** (*bool*) – Should the fuels be refilled to cover required distance?

Raises
RuntimeError: Out of fuel

Returns
A Car mileage

Return type
Cars

3.2 module.submodule

module.submodule.dummycode	This script contains some dummy codes with their respective docstrings to use them for templatisation

3.2.1 module.submodule.dummycode

This script contains some dummy codes with their respective docstrings to use them for templatisation

Functions

multiply	Multiply two numbers with each other
some_function	Summary or Description of the Function
string_reverse	Returns the reversed String.

module.submodule.dummycode.multiply

multiply(*a*, *b*)

Multiply two numbers with each other

Parameters

- **a** (*int*) – first number
- **b** (*int*) – second number

Returns
a x b

Return type
int

module.submodule.dummycode.some_function

some_function(*argument1*)

Summary or Description of the Function

Parameters
argument1 (*int*) – Description of arg1

Returns
Returning value

Return type
int

module.submodule.dummycode.string_reverse

string_reverse(*str1*)

Returns the reverse String.

Parameters
str1 (*str*) – The string which is to be reversed.

Returns
The string which gets reversed.

Return type
reverse(str1)

Classes

GoogleVehicle	The Vehicle object contains a lot of vehicles
NumpyVehicle	The Vehicles object contains lots of vehicles
SphinxVehicle	The Vehicle object contains lots of vehicles

module.submodule.dummycode.GoogleVehicle

class GoogleVehicle(*arg*, **args*, ***kwargs*)

Bases: object

The Vehicle object contains a lot of vehicles

Parameters

- **arg** (*str*) – The arg is used for...
- ***args** – The variable arguments are used for...
- ****kwargs** – The keyword arguments are used for...

arg

This is where we store arg,

Type
str

Methods

cars	We can't travel distance in vehicles without fuels, so here is the fuels

cars(*distance*, *destination*)

We can't travel distance in vehicles without fuels, so here is the fuels

Parameters

- **distance** (*int*) – The amount of distance traveled
- **destination** (*bool*) – Should the fuels refilled to cover the distance?

Raises
RuntimeError – Out of fuel

Returns
A car mileage

Return type
cars

module.submodule.dummycode.NumpyVehicle

class NumpyVehicle(*arg*, **args*, ***kwargs*)

Bases: object

The Vehicles object contains lots of vehicles

Parameters

- **arg** (*str*) – The arg is used for …
- ***args** – The variable arguments are used for …
- ****kwargs** – The keyword arguments are used for …

arg

This is where we store arg,

Type
str

Methods

cars	We can't travel distance in vehicles without fuels, so here is the fuels

cars(*distance*, *destination*)

We can't travel distance in vehicles without fuels, so here is the fuels

Parameters

- **distance** (*int*) – The amount of distance traveled
- **destination** (*bool*) – Should the fuels refilled to cover the distance?

Raises
RuntimeError – Out of fuel

Returns
A car mileage

Return type
cars

module.submodule.dummycode.SphinxVehicle

class SphinxVehicle(*arg*, **args*, ***kwargs*)

Bases: object

The Vehicle object contains lots of vehicles

Parameters

- **arg** (*str*) – The arg is used for ...
- ***args** – The variable arguments are used for ...
- ****kwargs** – The keyword arguments are used for ...

Variables

arg (*str*) – This is where we store arg

Methods

cars	We can't travel a certain distance in vehicles without fuels, so here's the fuels

cars(*distance*, *destination*)

We can't travel a certain distance in vehicles without fuels, so here's the fuels

Parameters

- **distance** – The amount of distance traveled
- **destinationReached** (*bool*) – Should the fuels be refilled to cover required distance?

Raises

RuntimeError: Out of fuel

Returns

A Car mileage

Return type

Cars

CHAPTER

FOUR

CHANGELOGS

v0.0.1

New Features

- Added all common sphinx templates used in documentation
- Added installation, changelog and references

CHAPTER

FIVE

REFERENCES

- **Understanding sphinx**
  Sphinx
- **Understanding how to use the read the docs theme for documentation**
  Sphinx Read the docs theme
- **Full documentation example**
  Drift Detection Documentation
- **restructuredtext cheat sheets**
  Reference 1

  Reference 2

  Reference 3

PYTHON MODULE INDEX

m