swp-docs Documentation

Release 0.1

Scientific Software Platform (Champalimaud Foundation)

Jun 29, 2017

Tutorials

Here you will find documentation developed and used by the Scientific Software Platform team from the Champalimaud Foundation. Find more about us in the about-label section.

CHAPTER 1

Writing documentation with Sphinx

Getting started

Install and configure sphinx

First, we need to create a folder to store our documentation. Normally, it should be in the project root folder.

You also need to install sphinx dependencies.

```
mkdir docs
pip install sphinx sphinx-autobuild
pip install sphinx-rtd-theme # this is an optional theme
```

Then, you need to run the sphinx-quickstart script in order to generate the make files for sphinx. This script will prompt you with several questions. Below, we leave a suggestion of what you should answer

```
cd docs
sphinx-quickstart
```

Suggested answers:

- > Separate source and build directories (y/n) [n]: **y**
- > Do you want to use the epub builder (y/n) [n]: **y** (optional)
- > autodoc: automatically insert docstrings from modules (y/n) [n]: **y**
- > intersphinx: link between Sphinx documentation of different projects (y/n) [n]: **y**
- > viewcode: include links to the source code of documented Python objects (y/n) [n]: **y**
- > Create Makefile? (y/n) [y]: **y**
- > Create Windows command file? (y/n) [y]: **y**

For all the other questions, leave the default answer or provide required information (e.g., your project name).

Changing the conf.py file

Any configuration related with sphinx should be inside the *conf.py* file which you should find in the */docs/source* folder.

For example, you can change the default theme by looking for the *html_theme* tag.

```
# The theme to use for HTML and HTML Help pages.  See the documentation for
# a list of builtin themes.
html_theme = 'sphinx_rtd_theme' # use Read The Docs theme
```

Edit, build, preview cycle

From now on, you can start working on your documentation. The process should go like this:

- Make changes to your .rst files
- Build your documentation:

```
cd docs
make clean # you only need to do this once a while
make html
```

- Preview the results by opening the file *docs/build/index.html*

See also:

You can find more information on this on the the following resources:

http://www.sphinx-doc.org/en/1.5.1/tutorial.html https://docs.readthedocs.io/en/latest/getting_started.html#write-your-docs http://matplotlib.org/sampledoc/

You can find projects using sphinx here:

http://www.sphinx-doc.org/en/1.5.1/examples.html

Publish on Read The Docs

Read The Docs (RTD) *hosts documentation, making it fully searchable and easy to find.*

To get started, create an account on RTD and import your project via a link to your git repository.

For simple projects (no complex dependencies), you don't need to configure much. RTD uses sphinx by default so it should be straightforward.

However, if your project needs to install several dependencies, you will need to install them via pip or conda.

You can configure RTD in 2 ways:

- configure RTD in the web interface and specify a requirements file (that should be in your project source code)
- include a *readthedocs.yml* in your project source code root folder

Then you can specify what packages to install on the RTD environment via pip, conda, or both:

- pip: specify the requirements file on the web interface or in the *readthedocs.yml*
- conda: specify the path for the conda *environment.yml* in the *readthedocs.yml*
- conda + pip: specify the path for the conda *environment.yml* in the *readthedocs.yml* and include pip dependencies

Below you will find an example of the third option which is the most flexible one.

```
my_project
        docs # documentation folder
                build
                source
                environment.yml # conda settings
                Makefile
        myproject # source code folder
        readthedocs.yml # RTD settings
```

```
# readthedocs.yml

conda:
    file: docs/environment.yml
```

```
# environment.yml

name: my-project-env # this is just a name for the environment
dependencies:
  - sphinx_rtd_theme
  - python=3.5
  - pyqt=4
  - numpy
  - pip:
      - sphinx
      - recommonmark
      - https://bitbucket.org/fchampalimaud/logging-bootstrap/get/master.zip
```

See also:

You can find more information on this on the the following resources:

https://docs.readthedocs.io/en/latest/ https://docs.readthedocs.io/en/latest/getting_started.html#write-your-docs

http://conda.pydata.org/docs/building/meta-yaml.html#the-meta-yaml-file https://docs.readthedocs.io/en/latest/conda.html?highlight=conda

Source code documentation

Configuring sphinx

Sphinx lets you document your Python code in a very versatile way. Moreover, it can find your code docstrings and automatically generate documentation for you. Finally, it can generate class diagrams. Awesome right?! So let's see how to do this.

See also:

If you never used sphinx before, please follow the guide *Getting started*.

Changing the conf.py file

First you have to configure sphinx by adding some information on the *conf.py* file which you should find in the */docs/source* folder.

Python path configuration

Sphinx must know where to search for your code. Because it will search on the Python path, you have 2 options:

- Install your project with pip before building the documentation (you have to repeat this every time you make changes)
- Inject your project source code directly in the Python path by changing the *conf.py* file

```
# If extensions (or modules to document with autodoc) are in another␣
↪directory,
# add these directories to sys.path here. If the directory is relative to the
# documentation root, use os.path.abspath to make it absolute, like shown␣
↪here.

sys.path.append(os.path.abspath('../../myproject'))
sys.path.append(os.path.abspath('../../'))

# Also import any other stuff you need here
from pysettings import conf; conf += 'pyforms_generic_editor.settings'
```

Add extensions for auto generated documentation and diagrams

If you followed our guide, you should already have the *autodoc* extension. In that case, just add the *graphviz* and the *inheritance_diagram* extensions.

```
extensions = [
    'sphinx.ext.autodoc',
    'sphinx.ext.viewcode',
    'sphinx.ext.intersphinx',
    'sphinx.ext.graphviz',
    'sphinx.ext.inheritance_diagram',
]
```

See also:

You will need to install the Graphviz package on your system.

On Mac OSX: **`brew install graphviz`**

On Ubuntu: **`apt-get install graphviz`**

Writing documentation

Writing documentation can be done in one of the following ways:

- *Manualy adding code directives*
- *Automatically extract docstrings from your code* (**recommended**)

Manualy adding code directives

You can document your entire code manually with sphinx.

For example, the following sphinx code

```
.. py:class:: CommAdpater

        Implement communication details

        .. py:method:: send_message(self, sender, recipient, message_body,␣
↪[priority=1])

            Send a message to a recipient

            :param str sender: The person sending the message
            :param str recipient: The recipient of the message
            :param str message_body: The body of the message
            :param priority: The priority of the message, can be a number 1-5
            :type priority: integer or None
            :return: the message id
            :rtype: int
            :raises ValueError: if the message_body exceeds 160 characters
            :raises TypeError: if the message_body is not a basestring
```

would generate this:

class CommAdpater

Implement communication details

send_message(*self*, *sender*, *recipient*, *message_body*[, *priority=1*])

Send a message to a recipient

Parameters

- **sender** (*str*) – The person sending the message
- **recipient** (*str*) – The recipient of the message
- **message_body** (*str*) – The body of the message
- **priority** (*integer or None*) – The priority of the message, can be a number 1-5

Returns the message id

Return type int

Raises

- **ValueError** – if the message_body exceeds 160 characters
- **TypeError** – if the message_body is not a basestring

See also:

You can find more information on this on the sphinx documentation page.

http://www.sphinx-doc.org/en/latest/domains.html#the-python-domain

Automatically extract docstrings from your code

Documenting your entire code manually can be a pain in the neck. Moreover, if you already included docstrings during development time, you would have to repeat this task and have to maintain documentation on two places.

A much better approach is to use the sphinx extension for discovering your source code structure and automatically extract docstrings from it.

Generating an inheritance diagram

```
.. inheritance-diagram:: pycontrolgui.models.experiment.experiment_uibusy
    :parts: 1
```

Auto extracting docstrings from source code

```
.. automodule:: pycontrolgui.models.experiment.experiment_window
    :members:
    :show-inheritance:
    :private-members:
```

You can see a nice example of the usage of the *autodoc*, *graphviz* and *inheritance extensions* for the *experiment_window.py* module under the pycontrol-gui project:

- `source code`
- `sphinx code`
- final result

See also:

You can find more information on this on the the following resources.

http://www.sphinx-doc.org/en/latest/ext/autodoc.html http://thomas-cokelaer.info/tutorials/sphinx/docstring_python.html

http://www.sphinx-doc.org/en/latest/ext/graphviz.html

http://www.sphinx-doc.org/en/1.5.1/ext/inheritance.html

http://pycontrol-gui.readthedocs.io/en/latest/

CHAPTER 2

Installing Python

On Mac OS

Mac OS ships with a Python installation which is used by internal services. You should not mess up with this version.

Homebrew (or just brew) is a package manager for Mac OS that offers a lot of useful libraries and software. We will use it to install Python on Mac OS.

Installing Homebrew

You can install Python directly from Homebrew or to use Pyenv. Either way, you will need Homebrew installed. Just follow the instructions here.

Instaling Python with pyenv [recommended]

If you want several Python versions installed at the same time, pyenv is what you need!

pyenv lets you easily switch between multiple versions of Python. It's simple, unobtrusive, and follows the UNIX tradition of single-purpose tools that do one thing well.

Note:

- pyenv should be also available for other Unix systems.
- You don't need to install Python with Homebrew, if you use pyenv.

How-to

1. **Installing pyenv and pyenv-virtualenv using an Homebrew formula**

```
brew install pyenv
echo 'export PYENV_ROOT="$HOME/.pyenv"' >> ~/.bash_profile
echo 'eval "$(pyenv init -)"' >> ~/.bash_profile
brew install pyenv-virtualenv
echo 'eval "$(pyenv virtualenv-init -)"' >> ~/.bash_profile

# show all installed versions
pyenv versions

# show current default python
pyenv version

# show all installed virtualenvs
pyenv virtualenvs
```

Note: You can still install a Python version using Homebrew. Pyenv does not conflict with that.

2. **Installing a python version and creating virtualenv**

```
# choose whatever version you want here, the autocomplete helps
# for advanced usage like creating executables, compiling packages, you need the␣
↪python framework included
env PYTHON_CONFIGURE_OPTS="--enable-framework" pyenv install 3.5.3

# create new virtualenv with python version just created
pyenv virtualenv 3.5.3 my-virtualenv-py3.5.3

# activate virtualenv
pyenv activate my-virtualenv-py3.5.3

# deactivate virtualenv
pyenv deactivate

# removing virtualenv
pyenv uninstall my-virtualenv-py3.5.3
```

More info

Note:

- https://github.com/pyenv/pyenv
- https://github.com/pyenv/pyenv-virtualenv

Installing Python from brew

If you don't like pyenv or just need only one version of Python3 or Python2 you can use Homebrew.

1. **Install Python3** using an Homebrew formula (also works for Python2)

```
brew install python3
```

Optional scientific packages can be installed also with Homebrew. More information:

- [Scientific Python on Mac OS X 10.9+ with homebrew | Jörn's Blog](https://joernhees.de/blog/2014/02/25/scientific-python-on-mac-os-x-10-9-with-homebrew/)
- [Installing scientific Python on Mac OS X | Lowin Data Company](http://www.lowindata.com/2013/installing-scientific-python-on-mac-os-x/)

Warning: These links may not be up-to-date. Please read carefully.

On Windows

Using WinPython [recommended]

Installing Python on Windows can be simpler if you use WinPython. This way you do not mess with other Python installations on the system.

You can download the latest version here: https://sourceforge.net/projects/winpython/files/latest/download?source=files

CHAPTER 3

Installing PyQT

PyQt5 installation with QScintilla2 using pip

PyQt5 installation is very easy and cross-platform. Just use pip as stated below.

```
pip install pyqt5
pip install qscintilla
```

Installing PyQt5 on Mac OS with brew [DEPRECATED]

Warning: PyQt5 and QScintilla2 can now be installed via PIP which is much easier.

1. Uninstall Qt4 (optional)

```
brew uninstall pyqt
brew uninstall qt
brew uninstall sip
brew uninstall qscintilla2 # optional
brew uninstall opencv3 # optional
brew untap cartr/qt4 # if installed
```

2. Install Qt5, QScintilla2 and OpenCV3

```
# install pyqt5
brew install pyqt5 --with-python

# install qscintilla2 with qt5 support
brew install qscintilla2 --with-plugin --with-python
```

PyQt4 Installation with QScintilla [DEPRECATED]

On Mac OS

Homebrew no longer officially supports Qt4. Everything by default uses Qt5. This is an alternative method to install Qt4 with QScintilla.

> **Warning:** You should remove any version of Qt5 and PyQt5 installed via Homebrew.
>
> Upgrading QScintilla may overwrite Qt4 installation.

1. **Install Qt4**

```
brew tap cartr/qt4
brew install cartr/qt4/qt
```

2. **Install PyQt** using an old Homebrew formula

```
brew install pyqt.rb --with-python3
```

`pyqt.rb`

> **Note:** This file is an old version of Homebrew for PyQt4. You can update some details if you want in order to adequate to your system (e.g. Mac OS version, pyqt version).

3. **Install QScintilla** using an old Homebrew formula

```
brew install qscintilla2.rb --with-python3
```

`qscintilla2.rb`

> **Note:** This file is an old version of Homebrew for PyQt4. You can update some details if you want in order to adequate to your system (e.g. Mac OS version, qscintilla version).

CHAPTER 4

Installing OpenCV for Python development

On Mac OS

Installing Opencv3 with qt5, ffmpeg and opengl support

```
# install opencv3 with qt5, ffmpeg and opengl support
brew install opencv3 --with-python3 --with-qt --with-ffmpeg --with-opengl

# export file to your python installation
echo /usr/local/opt/opencv3/lib/python3.6/site-packages >> /usr/local/lib/python3.6/
↪site-packages/opencv3.pth
```

On Windows

Installing Opencv3 with qt5, ffmpeg and opengl support

If you use Python2 with WinPython, you can download the official OpenCV binaries.

1. Install opencv310 from http://opencv.org/releases.html
2. Copy opencv\build\python\x86\2.7\cv2.pyd to WinPython-32bit-2.7.10.3\python-2.7.10\Lib\site-packages
3. Copy opencv_ffmpeg310.dll to WinPython-32bit-2.7.10.3python-2.7.10

For python3, you need to install the unofficial wheel: http://www.lfd.uci.edu/~gohlke/pythonlibs/.

CHAPTER 5

The SWP Team

Scientific Software Platform (Champalimaud Foundation)

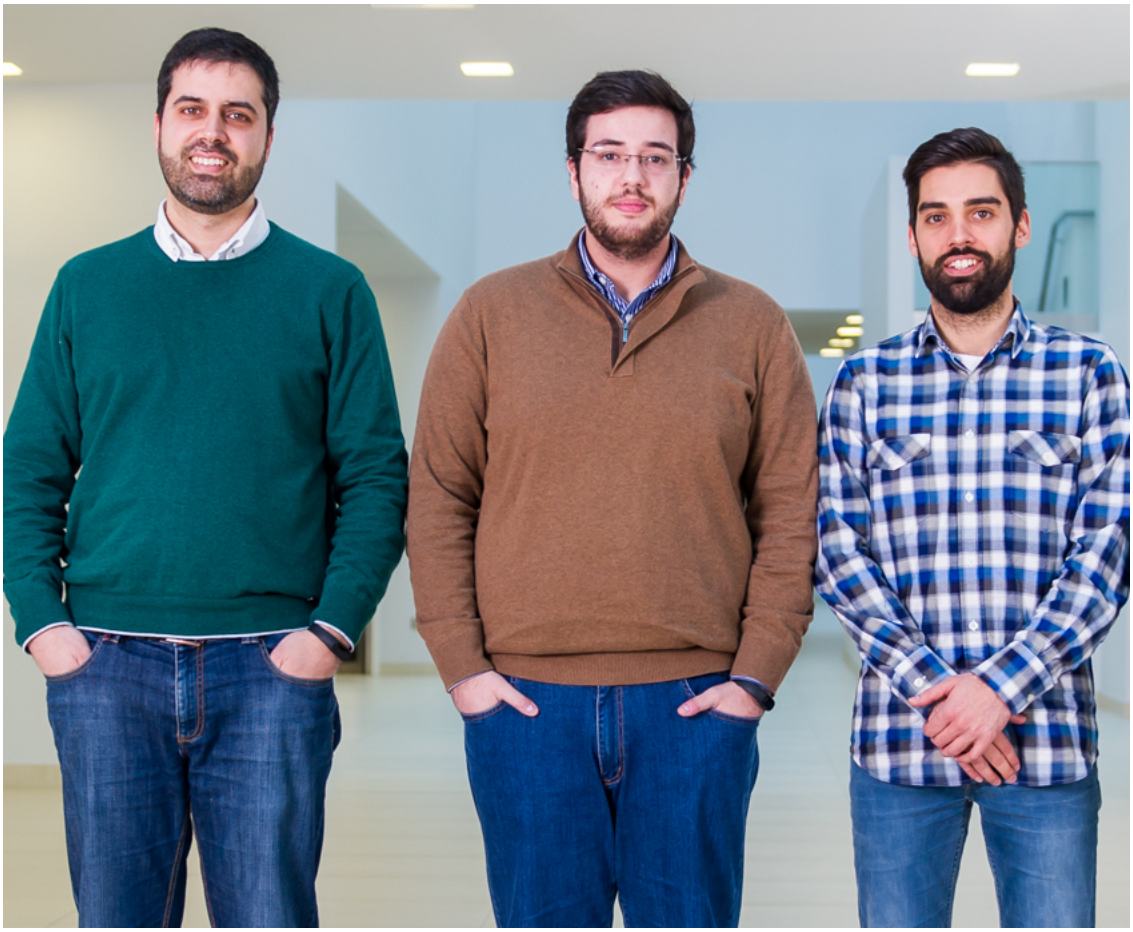
The Scientific Software Platform (SWP) from the Champalimaud Foundation provides technical know-how in software engineering and high quality software support for the Neuroscience and Cancer research community at the Champalimaud Foundation.

We typical work on computer vision / tracking, behavioral experiments, image registration and database management.

Members

The current and former members of the **pybpod-api** team.

- @UmSenhorQualquer Ricardo Ribeiro
- @JBauto João Baúto
- @cajomferro Carlos Mão de Ferro **[former member]**



License

We use the GNU General Public License version 3.

Questions?

If you have any questions or want to report a problem with this library please fill a issue here.

Index

C

S