# scifig Documentation

***Release 0.1***

**F. Boulogne**

January 09, 2016

Contents

Contents:

CHAPTER 1

# What is it?

## 1.1 Introduction

Making figures for presentations, websites or publications usually take a lot of time. However, the process is usually the same (compile, convert and export). Thus, automation must be used to focus on the content and not on managing files. Based on this observation, I conceived a workflow and I wrote a software that automate this task.

The source code is available here: https://github.com/sciunto-org/scifig

## 1.2 Choices

### 1.2.1 Languages and softwares for figures

We need to choose a technology to make figures. I like to write code, I'm efficient that way, so I choose LaTeX because I already use it for my documents and my presentations. To draw figures, pstricks or Tikz can be used. I tried both and Tikz looked to me more efficient (for loops for instance). To make plots, I'm familiar with gnuplot for almost a decade now. Gnuplot has a tikz terminal and this is lucky. It will be very easy to combine drawings and plots. For rare other needs (chemical formulae), LaTeX usually have what we need.

Ktikz and gummi are really helpful to prepare drawings. They refresh the output while you edit the code. It's almost a live mode.

### 1.2.2 Output formats

- tex (standalone file)
- eps
- pdf
- svg

### 1.2.3 Workflow

Now, we need to find a strategy to make our figures.

Drawings

- tikz file: the code you write with Ktiz for instance, between \begin and \end{tikzpicture}

- tex file: we decorate the tikz file with some packages and a document class (standalone) to have a croped image
- pdf file: the tex file compiled with pdflatex
- eps file: pdf to eps
- svg file: pdf to svg

Plots with gnuplot

- plt file: this is the standard gnuplot script to make a figure, with tikz terminal
- tikz file: code generated by gnuplot
- plttikz files: optional, inject tikz code before or after the code generated by gnuplot. It allows to draw in plots!
- tex file: we decorate the plttikz file with some packages and a document class (standalone) to have a cropped image
- pdf file: the tex file compiled with pdflatex
- eps file: pdf to eps
- svg file: pdf to svg

Others

Any other chain for another tool can be implemented, such as matplotlib in python, which has a tikz terminal too.

## 1.3 Implementation

My first implementation (around 2007) was based on shell scripts and makefile. Quickly coded, but not easy to maintain and the code tend to become hard to read very rapidly.

The second implementation (around 2011) used waf, a build to in python. The idea was to reuse something existing rather than writing from scratch a workflow. However, in practice, waf is not full adapted for the workflow I need. I made a working version I used for almost 4-5 years, I found interesting features (colored output, src and build separated, export function) but the addition of new workflow would require a lot of energy.

Then, I arrived to write a third version, from scratch since I know perfectly what I need:

- a database to record what must be recompiled based on modifications
- a detector to guess the right rule to apply (based on file formats)
- a set of rules to build the figure
- a nice output

CHAPTER 2

# How to install?

## 2.1 Requirements

The code is tested with python 3.4.

Po4a is an optional requirement (see below).

## 2.2 Package manager

- [Archlinux](https://aur.archlinux.org/packages/scifig/)

## 2.3 PyPI

See Pypi

To install with pip:

```
pip install scifig
```

## 2.4 Manual installation

From sources

```
python setup.py install
```

## 2.5 Files for po4a for translations (i18n)

Po4a part (optional)

According to your distribution, move pm files located in po4a/ to

- /usr/share/perl5/vendor_perl/Locale/Po4a
- or /usr/share/perl5/Locale/Po4a (debian)

CHAPTER 3

# libscifig API

## 3.1 Tasks

A *Task* is composed of the following steps:

1. Transform the source files to a tex file
2. convert tex to pdf with `_tex_to_pdf()`
3. convert pdf to svg with `_pdf_to_svg()`
4. convert pdf to eps with `_pdf_to_eps()`
5. convert pdf to png with `_pdf_to_png()`

Step 1 can be complex and could require several sub-steps. Thus, the role of `_pre_make()` is to do all these sub-steps.

Steps 2 to 5 usually do not depend on the initial type of the task. The function `make()` do all of them.

Each format has its own export function. The function `export()` exports all of them.

**class** `task.`**`GnuplotTask`**(*filepath*, *datafiles=[]*, *tikzsnippet=False*, *tikzsnippet1=False*, *tikzsnippet2=False*, *build='build'*, *db='db.db'*)

Bases: *task.Task*

Gnuplot Task manager.

**`check`**()
Check if the task needs to be done.

**`export`**(*dst='/tmp'*)
Export built files.

**Parameters** **`dst`** – filepath of the destination directory

**`export_eps`**(*dst='/tmp'*)
Export built EPS files.

**Parameters** **`dst`** – filepath of the destination directory

**`export_pdf`**(*dst='/tmp'*)
Export built PDF files.

**Parameters** **`dst`** – filepath of the destination directory

**`export_png`**(*dst='/tmp'*)
Export built png files.

**Parameters** **dst** – filepath of the destination directory

**export_svg**(*dst='/tmp'*)
Export built SVG files.

**Parameters** **dst** – filepath of the destination directory

**export_tex**(*dst='/tmp'*)
Export TEX files.

**Parameters** **dst** – filepath of the destination directory

**get_name**()
Return the name of the task.

**make**()
Compile the figure in all formats.

**class** task.**Task**(*filepath*, *build='build'*, *db='db.db'*)
Bases: object

Parent Task manager.

**Parameters**

- **filepath** – filepath of the main file
- **build** – relative filepath of the build dir
- **db** – relative filepath of the db file

**check**()
Check if the task needs to be done.

**export**(*dst='/tmp'*)
Export built files.

**Parameters** **dst** – filepath of the destination directory

**export_eps**(*dst='/tmp'*)
Export built EPS files.

**Parameters** **dst** – filepath of the destination directory

**export_pdf**(*dst='/tmp'*)
Export built PDF files.

**Parameters** **dst** – filepath of the destination directory

**export_png**(*dst='/tmp'*)
Export built png files.

**Parameters** **dst** – filepath of the destination directory

**export_svg**(*dst='/tmp'*)
Export built SVG files.

**Parameters** **dst** – filepath of the destination directory

**export_tex**(*dst='/tmp'*)
Export TEX files.

**Parameters** **dst** – filepath of the destination directory

**get_name**()
Return the name of the task.

**make**()
Compile the figure in all formats.

**class** task.**TikzTask**(*filepath*, *datafiles=[]*, *build='build'*, *db='db.db'*)
Bases: *task.Task*

Tikz Task manager.

**check**()
Check if the task needs to be done.

**export**(*dst='/tmp'*)
Export built files.

**Parameters** **dst** – filepath of the destination directory

**export_eps**(*dst='/tmp'*)
Export built EPS files.

**Parameters** **dst** – filepath of the destination directory

**export_pdf**(*dst='/tmp'*)
Export built PDF files.

**Parameters** **dst** – filepath of the destination directory

**export_png**(*dst='/tmp'*)
Export built png files.

**Parameters** **dst** – filepath of the destination directory

**export_svg**(*dst='/tmp'*)
Export built SVG files.

**Parameters** **dst** – filepath of the destination directory

**export_tex**(*dst='/tmp'*)
Export TEX files.

**Parameters** **dst** – filepath of the destination directory

**get_name**()
Return the name of the task.

**make**()
Compile the figure in all formats.

## 3.2 database

database.**check_modification**(*name*, *dependencies*, *db_path*)
Check if at least one dependency changed.

**Parameters**

- **name** – name of the figure
- **dependencies** – list of dependencies
- **db_path** – path of the database

**Returns** boolean

database.**erase_db**(*db_path*)
Erase a database.

**Parameters** **db_path** – path of the database

database.**store_checksum**(*name*, *dependencies*, *db_path*)
Store the checksum in the db.

**Parameters**

- **name** – name of the figure
- **dependencies** – list of dependencies
- **db_path** – path of the database

## 3.3 detector

detector.**detect_datafile**(*plt*, *root*)
Detect datafiles associated with a plt file.

**Parameters**

- **plt** – plt filepath
- **root** – root filepath

**Returns** list of filepath starting at root

detector.**detect_task**(*directory*, *root_path*)
Detect the task to do depending on file extensions.

**Parameters** **directory** – directory to look at

**Returns** list of tasks

detector.**detect_tikzsnippets**(*plt*)
Detect tikzsnippets associated with a plt file.

**Parameters** **plt** – plt filepath

**Returns** tuple of 2 booleans

## 3.4 collogging

CHAPTER 4

# Indices and tables

- genindex
- modindex
- search

## c



## d



## t

## C



## D



## E



## G



## M



## S



## T