MolGears Docs

version 1

Adrian Jasinski

December 04, 2014

Contents

The MolGears's documentation!

An overview of the MolGears

What is it?

This is a opensorce framework/database/toolkit based on a TurboGears web framework and RDKit open source toolkit for cheminformatics:

- programed in Python2.7 language
- BSD license
- source code on Github (https://github.com/admed/molgears)

Software Dependencies/Third-party software component

- RDKiT (http://www.rdkit.org)
- TurboGears (http://turbogears.org/)
- Postgresql Database (http://www.postgresql.org/)
- RDKit database cartridge for postgresql (http://www.rdkit.org/docs/Cartridge.html)
- Genshi (http://genshi.edgewall.org/)
- JSME editor (http://peter-ertl.com/jsme/)
- **Python based libraries:**
    - SQLAlchemy (http://www.sqlalchemy.org/)
    - razi (http://razi.readthedocs.org/en/latest/index.html)
    - Numpy (http://www.numpy.org/)
    - Scipy (http://www.scipy.org/)
    - Matplotlib (http://matplotlib.org/)
    - Xlwt & Xlrd (http://www.python-excel.org/)
    - xhtml2pdf (http://www.xhtml2pdf.com/)
    - pillow (https://github.com/python-pillow/Pillow)

Design goal

- Project management tool
- Efficient data storage
- Sorting, analysis, aggregation and reporting of data
- Data visualization
- Improved data access
- Automation of procedures
- Facilitate communication

Features

- Multi-projects
- Adding molecules by drawing or pasting SMILES code
- Reading molecules from file (csv, smi, sdf, mol, txt)
- Data presenting in sortable columns
- Exporting data to file (file formats: xls, pdf, csv, sdf, txt, png)
- PAINS (Pan Assay Interference Compounds) filtering (DOI: 10.1021/jm901137j)
- History of changes
- Compound filtering (by similarity, structure, identity, compound name, creator, adding date etc.)
- Stars rating
- IC50 determination based on least squares method,
- Automatic data processing
- Graphs generation
- Access managing
- Tags

License

Project Workflow Overview

Describe the workflow general schemes associated with a project.

Multiproject

- MolGears allows you divide your work into projects. The solution is designed to handle project management for small compounds.
- Molgears allows you to manage access and grant authorized users the right to use the service and access permision for each project independly.
- **In addition each project has basic workflow divided into tasks:**
    - designing process,
    - ordering synthesis,
    - tracking sythesis process,
    - storage compound in library,
    - result tool for compounds activity.

Project basic workflow

- A repeatable process that brings similar stages each time
- A clear division of responsibility between different people
- A better basis to estimate task length
- A simple method to communicate process and data to all or selected teem members/employees/colabollators/
- Each task has history records so you can see who added, approved and edited each record.

Basic Workflow Schema

For each project in menu bar you have access to 5 main tables connected to the project workflow.

- Compounds - root table. Storage for molecules structures; both: ideas and existing compounds. Unique records for each structure.
- Requests - put ideas into real things. Table for synthesis requests.
- Synthesis - table for tracking synthesis progress, priority managing, and analytical data storage.
- Library - table only for existing compounds. The library of compounds with ability for tracking location and the amount of compound.
- Results - compound activity data storage connected to library instance. It's allowed to add for one library record many results.

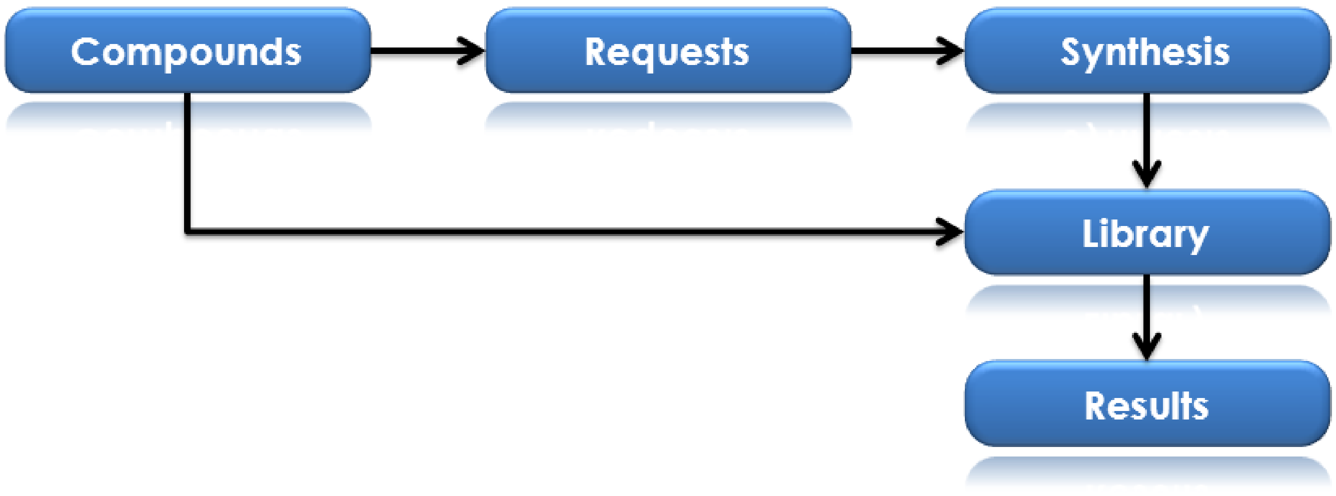
Workflow schema for the tables:

I. Add structures to Compounds table.

II. You can choose two path:

1. Internal synthesis:
    a. Create synthesis request
    b. Accept synthesis by chemist and proceed synthesis phases
    c. Add finished compound from synthesis to library
2. External synthesis (e.g. you're buying ready product):
    a. Accept compound directly to library

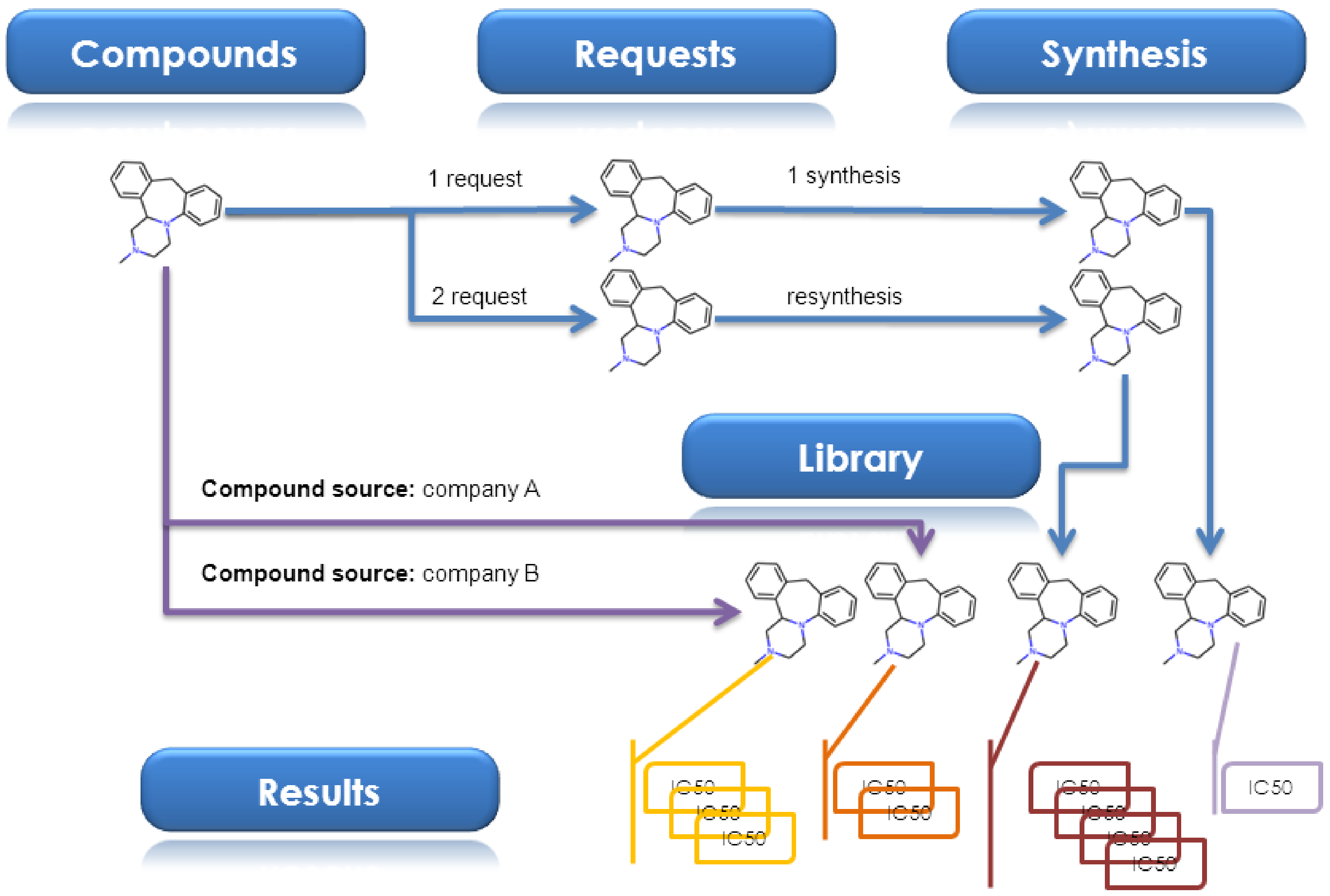
III. Add one or more activity results to library instance.



Example for the various sources of sample

Compounds table is containing unique structures of molecules. This means you can't add the same structure twice to Compounds Table. Only exception for this rule is adding compound as isomer. [references]. This rule also don't apply to other tables. You can create many request for one compound (structure) and then accept them to synthesis. One of the example of using this mechanism is performing synthesis of small amount of some compound. Later when we need more we can order the resynthesis by putting the same compound second time into request and follow the workflow once again.

Separate case is when you don't want to carry out the synthesis procedure and want to add a compound directly to the library e.g. when you're buying ready product from company A. Then follow the second path in workflow (see Project basic workflow). But you can also add this like that many times. If you add this twice and earlier when you performed 2 synthesis procedure as a result you will have 4 instances in library table referenced for the same structure. Than you can add one or more activity results independently for each of library instance of your compound. The shema illustrating this is presented below:



Database model

Database model was built based on Project Workflow (see: Project Workflow Overview).

Simple Model

Compounds and Projects tables are connected by many to many relationship with allowing to adding one compound to many projects and many compounds to one project.

Compounds Table is containing basic information about molecule struture like:

- SMILES and InChi code,
- name
- fingerprints
- molecular weight
- logP
- numer of atoms with and without Hydrogens
- number of rings
- Hbond acceptors
- Hbond donors
- Teorethical PSA

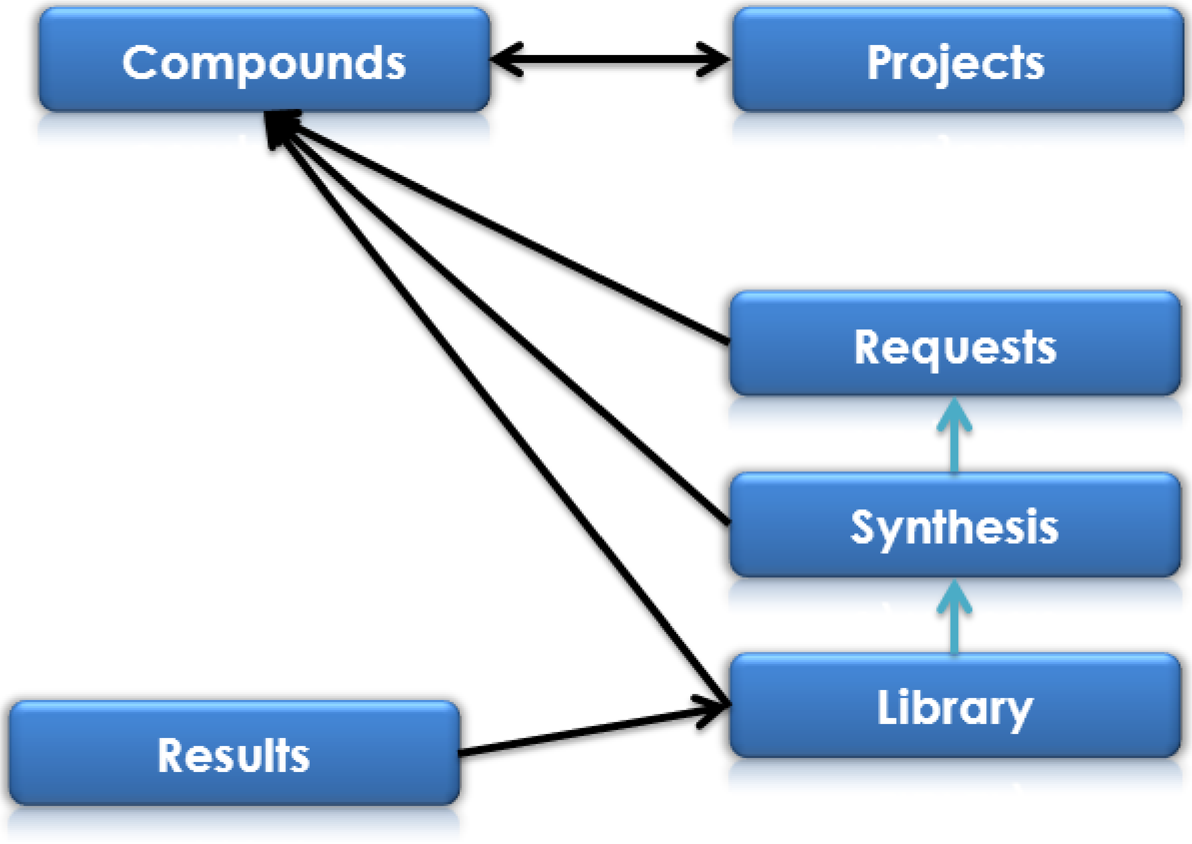
Requests, Synthesis and Library tables are refering to this data by relationship to the Compounds. This mean that editing and changing structure for Compound will make changes for all branches in all connected tables.

Synthesis table is storing ID of Request instance from which it is derived. This making the posibility for tracking the pathway of compound from one to other table and the ability to implement The Project Workflow in a consistent way.

The same connection is between Synthesis and Library tables.

Results table has many to one relationship to Library. In this way it's aviailble to add many results to one library instance.

Basic schema of database model is presented below:



Detailed Model

Entity - Relationship Diagram for MolGears Database model is presented below.

Diagram was generated using DBSchema.

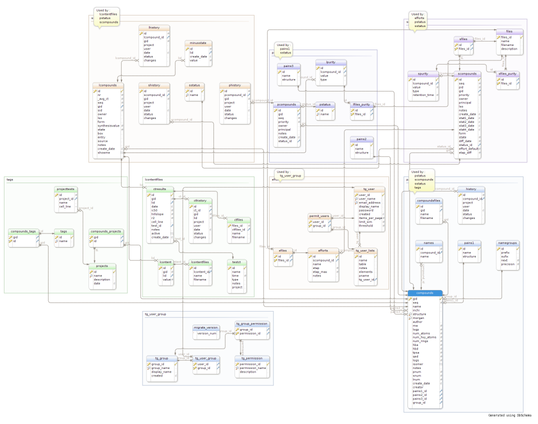
Main 6 Tables are respectively named as:

Model Name	SQL name
Projects	projects
Compounds	compounds
Requests	pcompounds
Synthesis	scompounds
Library	lcompounds
Results	ctoxicity

- Each of the main tables (except the Projects Table) has dedicated history tables for storing changes.
- Request and Synthesis tables have dedicated tables for Status.

- Compounds, Synthesis, Library and Results tables have dedicated tables for storing files.
- There are 3 PAINS tables for storing SMARTS codes.
- The tables for access managing have "tg_" prefix (tables for Users accounts, Users Lists, Groups and Permissions).
- Other tables are auxiliary and servs i.a. for auto-naming, state and purity monitoring, test description, tags storing etc.

ER Diagram (click the image to enlarge):



Installation

Installing procedure is tested only for Ubuntu 12.04+ and other debian-derived systems but general steps should work for other systems

Installing on Ubuntu

Installation steps one by one.

1. Installing TurboGears

TurboGears is meant to run inside python virtualenv and provides its own private index to avoid messing with your system packages and to provide a reliable set of packages that will correctly work together. First install python & python-virtualenv:

```
$ sudo apt-get install python
$ sudo apt-get install python-virtualenv
```

Running virtualenv with the option --no-site-packages will not include the packages that are installed globally. This can be useful for keeping the package list clean in case it needs to be accessed later.:

```
$ virtualenv --no-site-packages tg2env
```

in this tutorial we assume that you are in your home directory (~/ or /home/username) but you can choose your own directory for installation destination.

To begin using the virtual environment, it needs to be activated:

```
$ source tg2env/bin/activate
(tg2env)$ cd tg2env            #now you are ready to work with TurboGears
```

Be sure that you are in virtualenv mode during all installation procedure.

Install turbogears:

```
(tg2env)$ pip install tg.devtools
```

Start the project:

```
(tg2env)$ gearbox quickstart molgears
```

Delete unneeded files created by quickstart. We will replace it later by our project:

```
(tg2env)$ rm -rf ~/tg2env/molgears/molgears
```

Download molgears project from githubhttps://github.com/admed/molgears to ~/tg2env/molgears/

or use git:

```
(tg2env)$ cd ~/tg2env/molgears/
(tg2env)$ git clone https://github.com/admed/molgears
```

2. Build rdkit from source code with INCHI support:

Read the rdkit docs Install for more informations.

Installing prerequisites

Install the following packages using apt-get:

```
flex bison build-essential python-numpy cmake python-dev
libboost-dev libboost-python-dev libboost-regex-dev
```

Setting environment variables

RDBASE: the root directory of the RDKit distribution (e.g. ~/RDKit):

```
export RDBASE="/home/username/RDKit"
```

LD_LIBRARY_PATH: make sure it includes $RDBASE/lib and wherever the boost shared libraries were installed:

```
export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:$RDBASE/lib
```

PYTHONPATH: make sure it includes $RDBASE:

```
export PYTHONPATH=$PYTHONPATH:$RDBASE
```

Building the RDKit

Fetch the source, here as tar.gz but you could use git as well:

```
wget http://downloads.sourceforge.net/project/rdkit/rdkit/QX_20XX/RDKit_20XX_XX_X.tgz
```

for download the latest version (Q3 2014):

```
(tg2env)$ mkdir ~/RDKit
(tg2env)$ cd ~/RDKit
(tg2env)$ wget http://downloads.sourceforge.net/project/rdkit/rdkit/Q3_2014/RDKit_2014_09_1.
```

unpack the archive:

```
(tg2env)$ tar -xvf RDKit_2014_09_1.tgz
```

move files to ~/RDKit directory:

```
(tg2env)$ mv RDKit_2014_09_1/* ~/RDKit
```

Download InChi:

```
(tg2env)$ cd ~/RDKit/External/INCHI-API
(tg2env)$ ./download-inchi.sh
```

Building:

```
(tg2env)$ cd $RDBASE
(tg2env)$ mkdir build
(tg2env)$ cd build
(tg2env)$ cmake -DRDK_BUILD_INCHI_SUPPORT=ON ..
(tg2env)$ make
(tg2env)$ make install
```

Testing the build (optional, but recommended):

```
(tg2env)$ ctest
```

copy rdkit directory to:

```
(tg2env)$ cp ~/RDKit/rdkit ~/tg2env/lib/python2.X/site-packages/
```

3. Download razi

Razi provides extensions to SQLAlchemy to work with chemical databases.

Download razi from GitHub or from my fork

or by git:

```
(tg2env)$ cd ~/RDKit
(tg2env)$ git clone https://github.com/rvianello/razi
```

copy razi/razi to: ~/tg2env/lib/python2.X/site-packages/:

```
(tg2env)$ cp ~/RDKit/razi/razi ~/tg2env/lib/python2.X/site-packages/
```

4. Install postgresql

To install use the command line and type:

```
(tg2env)$ sudo apt-get install postgresql postgresql-contrib
```

Basic Server Setup

In a terminal, type:

```
(tg2env)$ sudo -u postgres psql postgres
```

Set a password for the "postgres" database role using the command:

```
postgres=# \password postgres
```

and give your password when prompted. The password text will be hidden from the console for security purposes.

Type Control+D to exit the posgreSQL prompt.

Since the only user who can connect to a fresh install is the postgres user, here is how to create yourself a database account (which is in this case also a database superuser) with the same name as your login name and then create a password for the user:

```
(tg2env)$ sudo -u postgres createuser --superuser $USER
(tg2env)$ sudo -u postgres psql
```

```
postgres=# \password $USER
```

Type Control+D to exit the posgreSQL prompt.

More installation information.

Creating a database

To create the first database, which we will call "molgears", simply type:

```
(tg2env)$ sudo -u postgres createdb molgears
```

Configuration

To improve performance while loading the database and building the index, I changed a couple of postgres configuration settings in postgresql.conf as they recommend in rdkit cartridge docs:

```
fsync = off                         # turns forced synchronization on or off
synchronous_commit = off            # immediate fsync at commit
full_page_writes = off              # recover from partial page writes
```

And to improve search performance, I allowed postgresql to use more memory than the extremely conservative default settings:

```
shared_buffers = 2048MB                 # min 128kB
work_mem = 128MB                        # min 64kB
```

Change requires restart:

```
(tg2env)$ sudo /etc/init.d/postgresql restart
```

5. Build the cartridge:

Go to the cartridge directory:

```
(tg2env)$ cd $RDBASE/Code/PgSQL/rdkit
```

run compilation, installation and testing:

```
(tg2env)$ make && make install && make installcheck
```

Add rdkit cartridge extension to database:

```
(tg2env)$ sudo -u postgres psql -c 'create extension rdkit' molgears
```

More info in rdkit docs and rdkit google code.

6. Add information about your database to development.ini file:

Edit the file: ~/tg2env/molgears/development.ini and comment the line:

```
# sqlalchemy.url = sqlite:///%(here)s/devdata.db
```

Than uncomment the line:

```
sqlalchemy.url=postgres://username:password@hostname:port/databasename
```

put your data like database USER name, password, hostname (or host IP i.e. 127.0.0.1), port (i.e. 8080) and databasename (in this example "molgears")

example line:

```
sqlalchemy.url=postgres://mike:hiddenpassword@127.0.0.1:8080/molgears
```

7. Install additional libraries by pip:

Install by using the command:

```
(tg2env)$ pip install library_name
```

where libraries are::

- tw2.forms
- tw.forms
- Genshi
- zope.sqlalchemy
- webhelpers
- repoe.who
- repoze.who.plugins.sa
- psycopg2
- tgext.admin
- pillow
- paste

- xhtml2pdf
- xlwt
- xlrd
- numpy
- scipy
- matplotlib
- sqlalchemy_migrate

8. RUN your aplication:

Go to molgears directory:

```
(tg2env)$ cd ~/tg2env/molgears
```

run setup.py:

```
(tg2env)$ python setup.py develop
(tg2env)$ gearbox setup-app
```

run gearbox:

```
(tg2env)$ gearbox serve
```

than you can open your browser and go to address:

```
http://hostname:port (i.e. http://127.0.0.1:8080).
```

put your data:

```
login: manager
passwor: managepass
```

enjoy :)

Getting started

Installing your doc directory

You may already have sphinx sphinx installed -- you can check by doing:

```
python -c 'import sphinx'
```

If that fails grab the latest version of and install it with:

```
> sudo easy_install -U Sphinx
```

Now you are ready to build a template for your docs, using sphinx-quickstart:

```
> sphinx-quickstart
```

accepting most of the defaults. I choose "sampledoc" as the name of my project. cd into your new directory and check the contents:

```
home:~/tmp/sampledoc> ls
Makefile      _static         conf.py
_build                _templates      index.rst
```

The index.rst is the master ReST for your project, but before adding anything, let's see if we can build some html:

```
make html
```

If you now point your browser to _build/html/index.html, you should see a basic sphinx site.

Fetching the data

Now we will start to customize out docs. Grab a couple of files from the web site or git. You will need getting_started.rst and _static/basic_screenshot.png. All of the files live in the "completed" version of this tutorial, but since this is a tutorial, we'll just grab them one at a time, so you can learn what needs to be changed where. Since we have more files to come, I'm going to grab the whole git directory and just copy the files I need over for now. First, I'll cd up back into the directory containing my project, check out the "finished" product from git, and then copy in just the files I need into my sampledoc directory:

```
home:~/tmp/sampledoc> pwd
/Users/jdhunter/tmp/sampledoc
home:~/tmp/sampledoc> cd ..
home:~/tmp> git clone https://github.com/matplotlib/sampledoc.git tutorial
Cloning into 'tutorial'...
remote: Counting objects: 87, done.
remote: Compressing objects: 100% (43/43), done.
remote: Total 87 (delta 45), reused 83 (delta 41)
Unpacking objects: 100% (87/87), done.
Checking connectivity... done
home:~/tmp> cp tutorial/getting_started.rst sampledoc/
home:~/tmp> cp tutorial/_static/basic_screenshot.png sampledoc/_static/
```

The last step is to modify index.rst to include the getting_started.rst file (be careful with the indentation, the "g" in "getting_started" should line up with the ':' in `:maxdepth`:

```
Contents:

.. toctree::
   :maxdepth: 2

   getting_started.rst
```

and then rebuild the docs:

```
cd sampledoc
make html
```

When you reload the page by refreshing your browser pointing to _build/html/index.html, you should see a link to the "Getting Started" docs, and in there this page with the screenshot. Voila!

Note we used the image directive to include to the screenshot above with:

```
test
```

Indices and tables

- genindex
- modindex
- search