

PyROS developer guide

VERSION: 29/3/19

Official URL of this doc: <https://tinyurl.com/pyros-dev>

Authors: Etienne Pallier, Alain Klotz, Patrick Maeght

*"!!! Au nord..., c'était les CHEVRONS...,
la Terre..., c'était le python...,
le ciel..., les constellations,
les hommes..., des codeurs de fond !!!"*

INDEX TABLE

1. General architecture	5
2. Inside PyROS	6
3. Installation of needed dependencies	7
3.1. Compatible platforms	7
3.2. Installation of Python	7
3.3. Installation of MySQL server	8
3.4. Installation of Git	8
4. Downloading the PyROS software itself	9
4.1. Get sources of PyROS	9
4.1.1. Authenticate to the gitlab	9
4.1.2. Get PyROS using Linux	9
4.1.2.1. DYNAMIC VERSION (recommended) : Get a synchronized version	9
4.1.2.2. STATIC VERSION (for NON developers) : Download a static version (not synchronized and thus NOT RECOMMENDED) :	11
4.1.3. Get PyROS using Windows	11
4.2. Notes about MySql (TBC)	13

5. Structure of PyROS source code	14
6. Installing/deploying PyROS	15
6.1. The deployment of PyROS	15
6.2. Install the Comet python package (optional)	16
7. Updating PyROS (code & database)	18
7.1. Nominal situation	18
7.2. Failure on the “git pull”	18
7.3. Failure because of the database	20
8. Configuring : adapting the software to your needs	21
9. Turn on/off the virtual environment	22
10. Tests	23
10.1. Philosophy of tests	23
10.2. Unit tests	23
10.3. Testing agents	24
10.4. (OLD) Bigger tests	25
10.4.1. Demo 1 : with only the Users simulator activated	25
10.4.2. Demo2 : with all simulators activated	26
11. Running PyROS	27
11.1. Start only the webserver (for pyros website)	27
11.1.1. Now, you can access the PyROS website	27
11.1.2. You also can access the PyROS web administration interface	28
11.2. Start any agent	29
11.3. Play with the pyros shell (pyros interactive session)	30
11.3.1. Play with a pyros agent	30
11.3.2. Play with the pyros objects (data model)	37
12. Programming PyROS	39
12.1. DATABASE & TABLES management	40
12.1.1. Database schema (v0.22)	40
12.1.2. From PyROS objects to database tables	41
12.1.3. HOW TO add a new table in the database	41
12.1.4. Access the pyros database	44
12.1.5. Re-initialize the pyros database	45
12.1.6. Update the pyros database	45
13. PYROS MODULES (main functions)	47
13.1. BIG FUNCTIONS (modules)	47

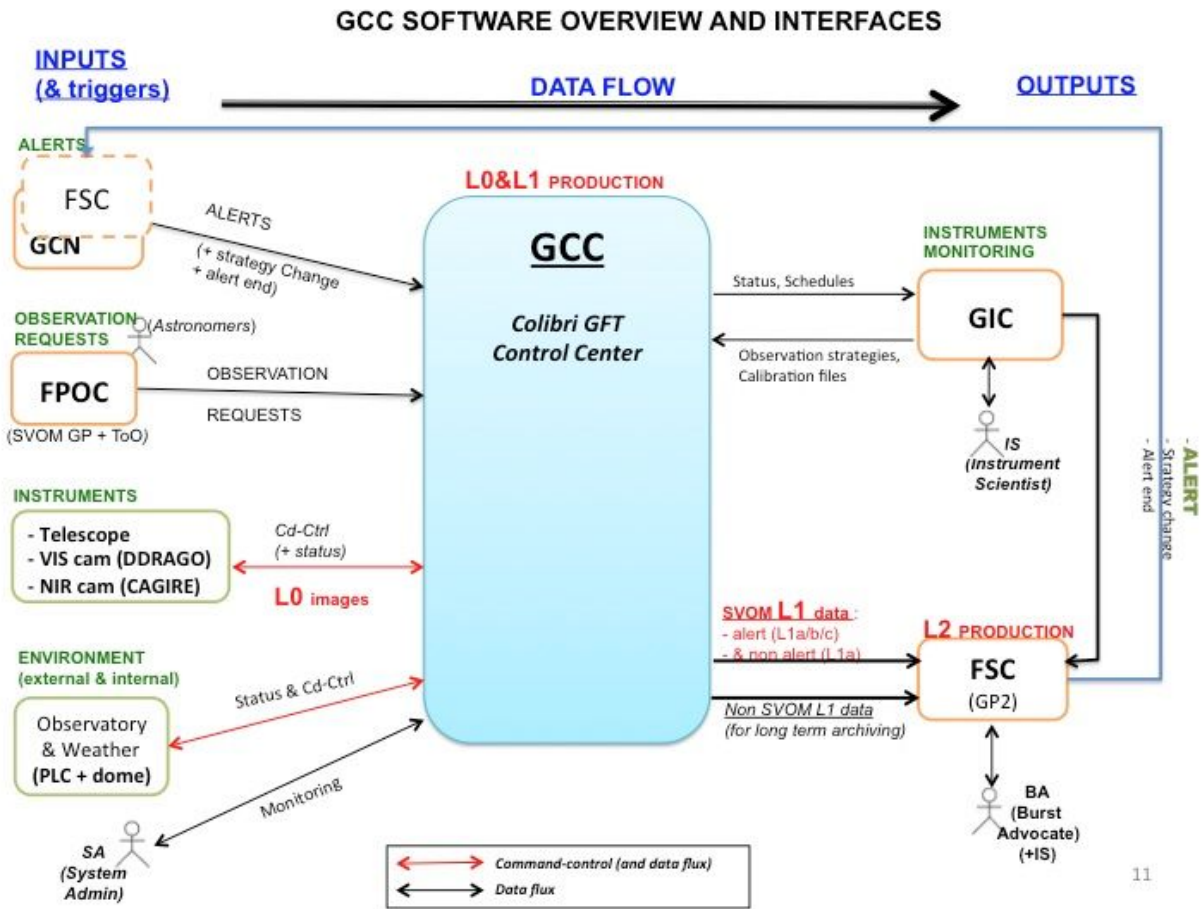
13.2. MAIN WORKFLOW	48
13.3. FUNCTION 0 - Agent	50
13.4. FUNCTION 1 - Alert Management (ALERT)	51
13.5. FUNCTION 2 - Observation Request Management (REQUEST)	52
13.6. FUNCTION 3 - Planning (PLANNER, SCHEDULER)	53
13.6.1. Specs	53
13.6.2. Deux phases principales de test	54
13.6.3. Exécution des tests existants	55
13.6.4. Jouer avec le Scheduler (via le django shell)	56
13.6.5. Procédure concrète de soumission des requetes pour les tests :	58
13.7. FUNCTION 4 - Observation EXECUTION & Instruments Monitoring (EXEC, system control)	60
13.7.1. MODULE "Majordome (Conductor, Master)"	61
13.7.1.1. Context diagram	61
13.7.1.2. States diagram	62
13.7.1.3. DEVICES STATUS LIST to be sent to GIC	63
13.7.1.4. RUN	64
13.7.1.5. TEST	64
13.7.1.6. General algorithm	66
13.7.2. MODULE "Observer (EXEC)"	68
13.7.2.1. Context diagram	68
13.7.2.2. Telescope monitoring agent	69
13.8. Javascript files -> misc/static/js	69
13.9. Navbar -> misc/templates/base.html or base_unlogged.html	69
13.10. FUNCTION 5 - ENVIRONMENT monitoring (ENV)	70
13.10.1. Contexte	71
13.10.2. PLC	72
13.10.2.1. Power management (onduleurs)	72
13.10.3. Fonctionnement du module	73
13.10.4. EXECUTION	74
13.10.4.1. Execution AVEC CELERY (version complète)	77
13.10.4.2. - (Agent) Terminal 0 - Un worker Celery dédié au Monitoring	77
13.10.4.3. - (Agent) Terminal 1 - L'Agent "Monitoring"	78
13.10.4.4. - (Agent) Terminal 2 - Le simulateur de PLC	80
13.10.5. DEVELOPMENT	89
13.11. FUNCTION 6 - DATA REDUCTION & ANALYSIS	90
13.11.1. Basic packages requirement.	90
13.12. FUNCTION 7 - DASHBOARD (Information system management)	91
13.12.1. Users management	93

13.12.1.1. Profiles	93
13.12.1.2. Scenario (workflow)	96
13.12.2. Observatory Control page	97
13.13. FUNCTION 8 - (Proposals) Scientific Programs Management	100
13.14. FUNCTION 9 - Telescope & Instruments long term Monitoring and Calibration	101
13.15. FUNCTION 10 - Data Archiving	102
14. ANNEXES	104
14.1. Annex 1 - Récupération BD après mauvaise manip (ou bien si “update” ne fonctionne pas)	104
14.2. Annex 2 - Git	105
14.2.1. Commit procedure	105
14.2.2. Pull procedure	106
14.3. Annex 3 - PyROS general python syntax rules and coding style	107
14.4. Annex 4 - PyROS objects naming rules (syntax for the models.py file)	112
14.5. Annex 5 - TESTS rules (integrated tests of the PyROS code)	116
14.5.1. Utilité des tests	116
14.5.2. Différents types de test	116
14.5.3. Organisation des tests dans le contexte de Django	117
14.5.4. Exécution des tests	117
14.5.5. Structure d’un fichier tests.py	119
14.5.6. Exemple de fichier tests.py	120
14.6. Python installation for specific operating systems	122
14.6.1. A2.1. Linux CentOS 7.1	122
14.6.2. A2.2. Linux Ubuntu, Suse	123
14.6.3. A2.3. Mac OS X	123
14.6.4. A2.4. Windows 10	124
14.7. MySQL installation for specific operating systems	125
14.7.1. A3.1. Linux CentOS	125
14.7.2. A3.2. Linux Ubuntu, Suse	125
14.7.3. A3.3. Mac OS X	126
14.7.4. A3.4. Windows 10	126
14.7.4.1. Installation of PHP to use phpmyadmin	127
14.7.4.2. Install phpmyadmin	127
14.8. Python packages installation for specific operating systems	127
14.9. Annex 5: Notes for Eclipse IDE users	128
14.10. Annex 6: Notes for PyCharm IDE users	131
14.11. Annex 7: Mount a virtual environment	132
15. ROADMAP TEMPORAIRE (TODO LIST) (à migrer dans Redmine)	133

GENERAL	133
AGENT	135
15.1. OLD TODO LIST (déjà migrée dans Redmine)	138
16. SOFTWARE HISTORY	142

1. General architecture

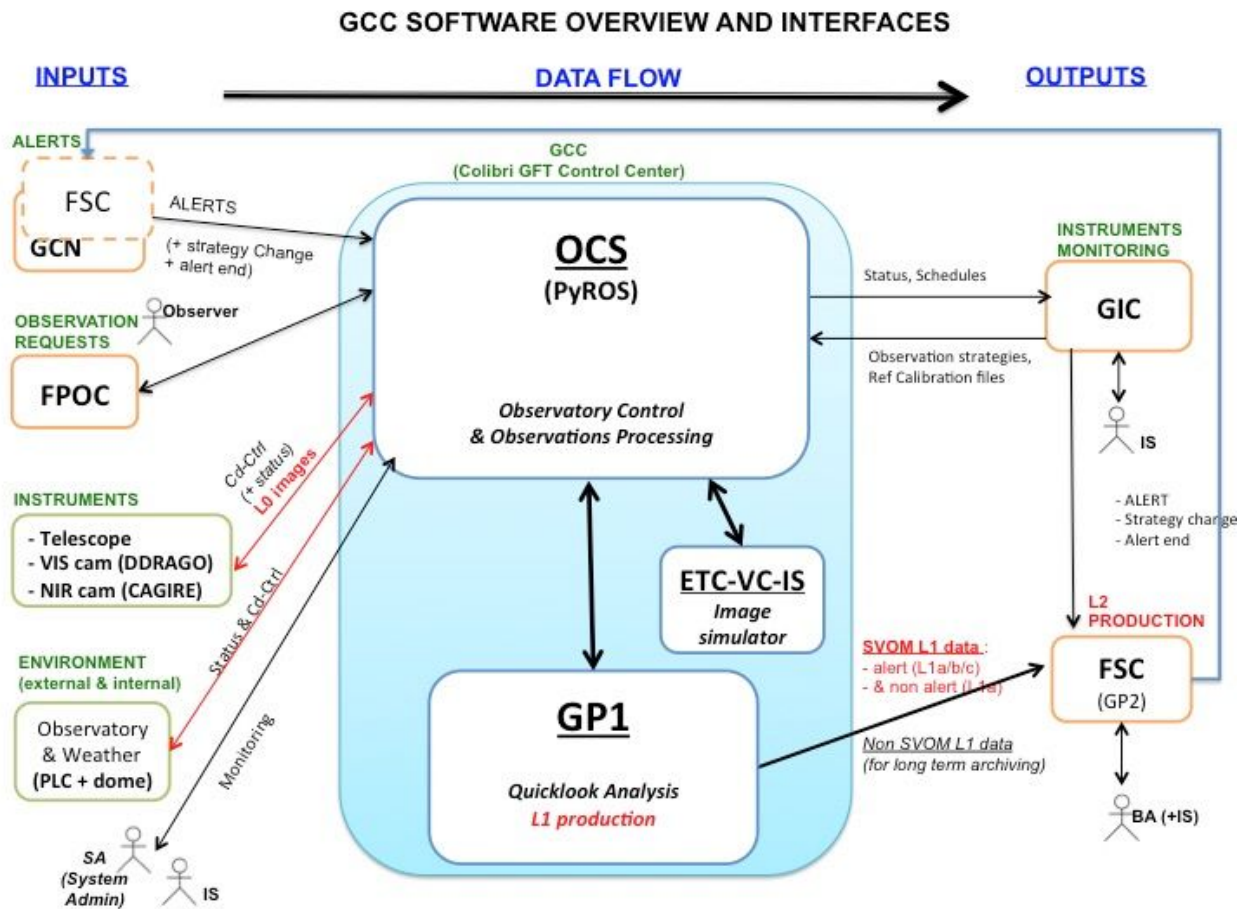
(updated 16/7/18)



Computers, Agents, Database, Inputs, Outputs, Configuration, Web

2. Inside PyROS

(updated 16/7/18)



List of agents, MySQL, inside one agent, simulators, tests

3. Installation of needed dependencies

(updated 24/1/19, EP)

Pyros needs some prerequisites :

- Python 3.6+ (3.7 recommended)
- Mysql Database server (last version recommended)
- Git client

3.1. Compatible platforms

(updated 24/1/19, EP)

This software is targeted first for Linux CentOS 7 (+ Fedora and Ubuntu), but also for Mac OS X and Windows 10.

All these systems should run Python **3.6** at least (**3.7** advised)

Pyros has been tested on these platforms:

- **Linux :**
 - (since 12/10/2018, EP) **CentOS 7.5** (with python 3.6.5, mysql 5.5.60-MariaDB) ⇒ http://planetowiki.irap.omp.eu/do/view/Computers/Hyperion2Server#PYROS_11_10_2018
 - (since 4/10/2018, EP) **Scientific Linux 6.4**, kernel 2.6.32 (with python 3.6.6, mysql Ver 14.14 Distrib 5.5.62, for Linux (x86_64) using readline 5.1) ⇒ [voir détails sur wiki](#)
 - Older installations:
 - **CentOS 7.1** (with Python 3.4)
 - Linux Mint 17.2 (== Ubuntu 14.04.3) (with python 3.5)
 - Ubuntu 16.04 (with python 3.5.2)
- **Mac OS X:**
 - (since 25/1/19, EP) **Mac OS** 10.14 (with python 3.7)
 - (older installation) Mac OS 10.13 (with python 3.6)
- **Windows :**
 - (since 24/1/19, AK) **Windows** 10 (with python 3.6)

3.2. Installation of Python

See annex to install Python on supported platforms.

3.3. Installation of MySQL server

See annex to install MySQL on supported platforms.

3.4. Installation of Git

Linux:

```
root $ apt-get install git
```

Windows:

<https://tortoisegit.org/>

4. Downloading the PyROS software itself

4.1. Get sources of PyROS

The Git server is hosted at IRAP laboratory. Hereafter the links to the Git repository of PyROS are:

- URL : <https://gitlab.irap.omp.eu/epallier/pyros>
(see Activity and Readme file)
- Browse source code (dev branch) : <https://gitlab.irap.omp.eu/epallier/pyros/tree/dev>
- Last commits : <https://gitlab.irap.omp.eu/epallier/pyros/commits/dev>
- Graphical view of commits : <https://gitlab.irap.omp.eu/epallier/pyros/network/dev>

4.1.1. Authenticate to the gitlab

In order to get this software, you must first authenticate on the IRAP gitlab <https://gitlab.irap.omp.eu/epallier/pyros>

For this, just go to <https://gitlab.irap.omp.eu/epallier/pyros> and either sign in with your LDAP account (if you are from IRAP), or register via the "Sign up" form.

4.1.2. Get PyROS using Linux

First, go to the directory where you want to install the software. It can be wherever you want, like your home directory for instance... Do not create a new directory for PyROS, it will be done automatically.

```
$ cd MY_DIR
```

4.1.2.1. DYNAMIC VERSION (recommended) : Get a synchronized version

If you do not want to contribute to this project but just want to try it, you can just download a STATIC version of it : go to next section "STATIC VERSION" below.

Windows users : you first need to get the GIT software (see below, "[For Windows users](#)")

By getting the software from git, you will get a dynamically synchronized version, which means that you will be able to update your version as soon as a new version is available (simply with the command : "git pull").

(From Eclipse : See below, section "[NOTES FOR ECLIPSE USERS](#)")

From the terminal :

```
$ git clone https://gitlab.irap.omp.eu/epallier/pyros.git PYROS
```

(the first time you get the project, it will ask you for a login and password)

((

you can also provide your login and password directly like this:

```
git clone https://username:password@gitlab.irap.omp.eu/epallier/pyros.git  
PYROS
```

))

(or also, using ssh, but not sure it works : `git clone git@gitlab1.irap.omp.eu:epallier/pyros.git PYROS`)

If you ever get this error message :

```
fatal: unable to access 'https://gitlab.irap.omp.eu/epallier/pyros.git/':  
Peer's certificate issuer has been marked as not trusted by the user.
```

Then, type this command (and then run again the git clone command):

```
$ git config --global http.sslVerify false
```

The "git clone..." above command has created a PYROS folder containing the project (with a .git/ subfolder for synchronization with the git repository)

Go into this directory :

```
$ cd PYROS
```

By default, you are on the "master" branch :

```
$ git branch  
* master
```

You should NEVER do any modification directly on this branch, so instead jump to the "dev" branch :

```
$ git checkout dev
```

```
$ git branch
```

```
* dev
```

```
master
```

4.1.2.2. STATIC VERSION (for NON developers) : Download a static version (not synchronized and thus NOT RECOMMENDED) :

Go to <https://gitlab.irap.omp.eu/epallier/pyros/tree/master>

Click on "Download zip" on the up right hand corner.

Double-click on it to unzip it.

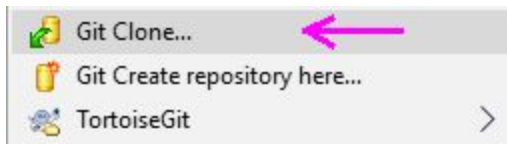
You should get a "pyros.git" folder.

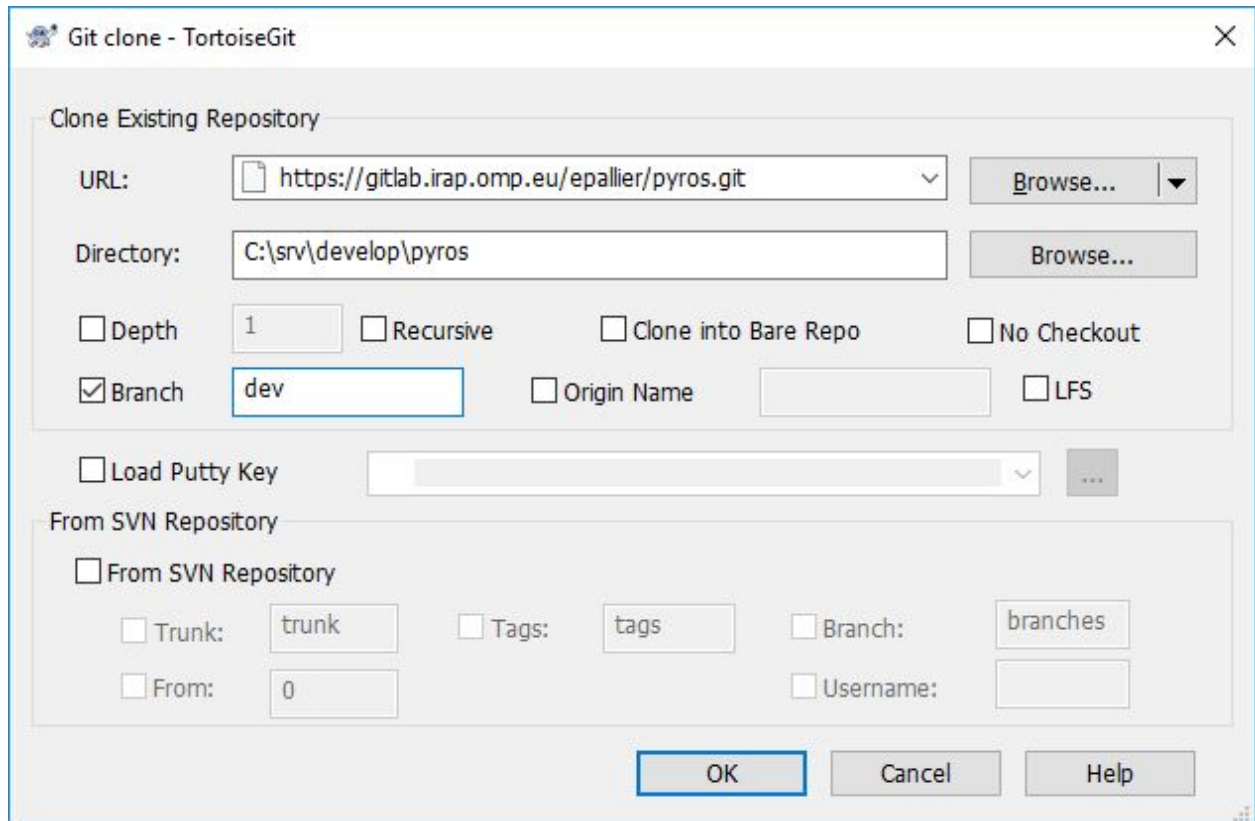
In this documentation, this software folder will be referenced as "PYROS".

(you can rename "pyros.git" as "PYROS" if you want : "mv pyros.git PYROS")

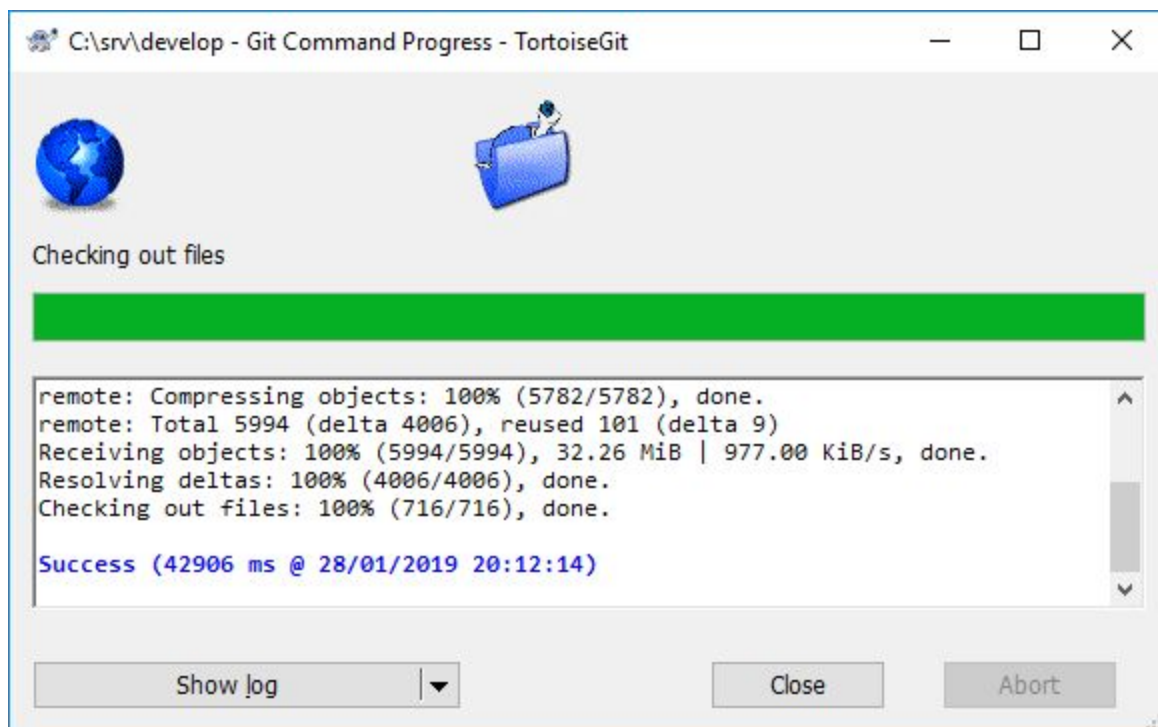
4.1.3. Get PyROS using Windows

It is recommended to use git with the graphic client TortoiseGit.





Don't forget to check the Branch writing dev in the text area.



Else, from the cmd or Powershell (not recommended):

- Download git at <https://git-scm.com/download/win>
- Run setup (keep default configurations)
- Once installed, open cmd or Powershell :

```
> git config --global http.sslVerify false
> git clone --single-branch --branch dev https://gitlab.irap.omp.eu/epallier/pyros.git pyros
```

4.2. Notes about MySQL (TBC)

Not sure this is still working... (to be tested)

By default, Pyros uses MySQL, but this implies that you have to install the MySQL database server...

Thus, to make things easier, avoid MySQL installation by using SQLite instead as the database server (which will need no installation at all) :

=> For this, just edit the file PYROS/src/pyros/settings.py and set MYSQL variable to False, and that's it. You can go to next section

Now, if you really want to use MySQL (which is the default), you will need to install it (only if not already installed), so keep reading.

(Skip this if you are using SQLite instead of MySQL)

5. Structure of PyROS source code

- src/ : conteneur du projet (le nom est sans importance)
 - manage.py : utilitaire en ligne de commande permettant différentes actions sur le projet
 - pyros/ : the actual Python package of the project
 - settings.py : project settings and configuration
 - urls.py : déclaration des URLs du projet
 - wsgi.py : point d'entrée pour déployer le projet avec WSGI
- database/ : database configuration and documentation
- doc/ : project documentation
- install/ : project installation howto
- private/ : the content of this folder is private and thus not committed to git ; it should contain your Python3 virtual environment
- simulators/ : the devices simulators
- public/ : this folder contains all public files like the web html files
 - static/

Each **APP**(lication) structure :

https://projects.irap.omp.eu/projects/pyros/wiki/Project_structure#Applications-architecture

6. Installing/deploying PyROS

(updated 12/10/18, EP)

Deploying PyROS consists to install python packages needed for PyROS, create the PyROS database and copying all the needed files into a virtual environment.

6.1. The deployment of PyROS

The needed Python packages will be automatically installed during the installation phase of PyROS.

When source code of PyROS is downloaded the Python package list is defined in the following files:

PYROS/install/REQUIREMENTS.txt

PYROS/install/REQUIREMENTS_WINDOWS.txt

The **install.py** script will install the needed packages and create the pyros database for you. Just go into the **PYROS/install/** folder and **Run the install.py without sudo privileges:**

For Windows you must run command lines in a Powershell console.

Before launching Python, remember the following fact:

Linux and Mac : it is VERY IMPORTANT that you type “python3” and not “python”

Windows : you might need to replace “python” with “py” depending the installation.

First, check that your python version is at least 3.6 :

Linux/Mac:

```
$ python3 -V
```

Windows (Powershell):

```
> python -V
```

Now run the install script:

```
cd PYROS/install/  
python3 install.py  
Windows (Powershell) : python install.py
```


If anything goes wrong with the mysql database (last step of the install process, especially with the migrations if they are too big), you can try this:

- drop your **pyros** database (and also **pyros_test** if ever it exists) :
\$ mysql -u root -p
\$ mysql> drop database pyros;
\$ mysql> drop database pyros_test;
\$ mysql> exit;
- Delete all existing migration files:
\$ cd src/common/migrations/
\$ rm 0*.py
- run again the install script

If it still does not work, try this:

- Edit src/pyros/settings.py
- Comment the django admin app like this:
INSTALLED_APPS = [
 #'django.contrib.admin',
- run again the install script
- Don't forget to comment out the django admin app

If something goes wrong with the python packages installation, you can try to install manually each package

Now that PyROS is installed, we can test it to be sure that it is really well installed.

Information for developers only :

*older version (with old Jeremy Barneron install.py script) : python3 install.py install
TODO: update "create user if exists" => does not work with mysql 5.6 (only with 5.7)*

6.2. Install the Comet python package (optional)

Comet is not needed yet, install it only if you want to work on the ALERT management part of the project. For now, do not bother with it, and go straight to next section.

Latest info on this package : <http://comet.transientskp.org/en/stable/>

Comet is needed as a broker to receive and send VOEvents

(<https://github.com/jdswinbank/Comet/tree/py3>)

You MUST have your virtualenv activated (source venv_py3_pyros/bin/activate in your 'private/' directory)

Documentation is available here : <http://comet.readthedocs.io/en/stable/installation.html>

(see also <http://voevent.readthedocs.io/en/latest/setup.html>)

Essayer d'abord la méthode automatique (avec pip) :

```
$ source private/venv_py3_pyros/bin/activate  
$ pip install comet
```

Si ça ne marche pas, essayer la méthode manuelle (download puis install) :

- Ubuntu :

```
# You can do this anywhere on your computer  
$ git clone https://github.com/jdswinbank/Comet.git  
$ cd Comet  
$ (sudo ?) python setup.py install  
$ sudo apt-get install python-lxml
```

- MacOS :

Idem Ubuntu

- Windows :

TODO:

Test Comet

```
$ twistd comet --help  
$ trial comet
```

All tests should pass

7. Updating PyROS (source code & database)

(updated 19/3/19 - EP)

Si vous avez seulement besoin de remettre en bon état votre BD après avoir supprimé une table, [voir la procédure manuelle à suivre \(Annexe 1\)](#).

7.1. Nominal situation

In order to update the pyros software, just type this command :

```
$ cd PYROS/  
$ ./pyros.py update  
(For Windows: python pyros.py update)
```

This will update 3 things :

- the python packages (pip install -r requirements)
- the source code (“git pull”)
- the database structure (“make migrations” + “migrate”)

7.2. Failure on the “git pull”

Issue : impossible de faire un git pull car j’ai modifié quelques fichiers

2 cas possibles :

- (1) soit je veux bien perdre ces modifs car elles ne sont pas importantes
- (2) soit je veux garder les modifs que j’ai faites

Cas 1 : ignorer les changements

Essayer ces solutions, dans l’ordre :

a - git checkout :

```
git checkout <file>  
ou encore “git checkout — <file>”
```

b - git reset (1) :

```
git reset HEAD <file>  
(ou “git reset HEAD” sans argument)
```

c - git reset (2) :

```
git reset --hard  
Ou encore “git reset - - hard HEAD”
```

(voire même : **git reset --hard origin/HEAD**) (if you've merged remote branches or have local commits and want to go back to the remote HEAD)

d - git update-index :

```
git update-index --assume-unchanged <file>
```

And if you want to start tracking changes again, you can undo the previous command :

```
git update-index --no-assume-unchanged <file>
```

Une fois que ces fichiers sont revenus dans leur état initial (avant modif), vérifier que tout va bien, puis faire le git pull :

```
git status
```

```
git pull
```

Cas 2 : je veux garder les modifs que j'ai faites

```
git status
```

```
git stash => met de côté mes fichiers non versionnés
```

```
git status
```

```
git pull
```

Puis, quand je voudrai récupérer mes fichiers modifiés :

```
git stash pop => récupère mes fichiers non versionnés
```

Ou encore:

```
git add <tous les fichiers que je veux versionner>
```

(ou encore :

```
git add . (ou git add --all)
```

```
git reset HEAD <tous les fichiers que je ne voulais pas versionner>
```

```
)
```

```
git commit -m "bla bla"
```

```
git pull
```

SI CONFLIT sur 1 ou des fichiers:

1) corriger les fichiers affectés

2) Ajouter les fichiers que j'ai corrigés

```
git add src/actions/fetch_new_cnes_tar_data_and_flatten_it.py src/cpipeline.py
```

```
src/chemcam_types/chemcam_file_manager.py
```

```
git status
```

3) Commiter ces corrections

```
git commit
```

```
git status
```

4) pusher ces corrections

```
git push
```

git status

7.3. Failure because of the database

In case of something goes wrong with the update, you can try these steps :

Reload the initial database configuration :

```
$ ./pyros.py initdb
```

(the initdb command will load a pyros initial fixture from the file src/misc/fixtures/initial_fixture.json)

If this is not enough, then try to recreate the database and reinitialize it :

Re-create and re-initialize the database :

```
$ ./pyros.py dbshell
  > drop database pyros;
  > create database pyros;
  > exit;
Supprimer tous les fichiers de migration actuels :
  $ rm src/common/migrations/00*.py
$ ./pyros.py initdb
$ ./pyros.py update
```

Then, you can go again to “Nominal situation” paragraph above, to be sure that everything is ok.

If it still does not work, well... then run “./pyros.py install” again (see previous “Installing PyROS” section for more details)

8. Configuring : adapting the software to your needs

(TODO:)

9. Turn on/off the virtual environment

The virtual environment is a protected space on the disk to solve the problems of dependencies of various applications running on the same OS. We use virtualenv to run PyROS.

Linux case:

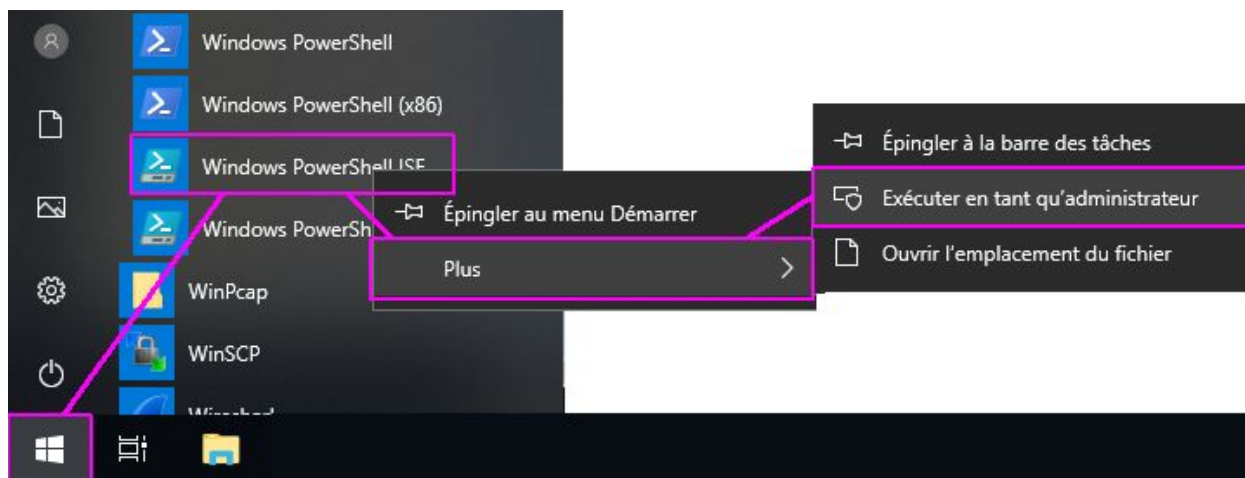
```
$ cd PYROS/  
$ source private/venv_py3_pyros/bin/activate
```

Windows console case (Win+R cmd):

```
> cd PYROS  
PYROS> private\venv_py3_pyros\Scripts\activate.bat  
(venv_py3_pyros) PYROS>
```

Windows powershell case:

The first time, you must unrestrict the execution policy. Launch Powershell in administration mode



```
PS > set-executionpolicy unrestricted
```

Then the following commands can be executed in a Powershell not in admin mode:

```
PS > cd PYROS  
PS PYROS> private\venv_py3_pyros\Scripts\activate.ps1  
(venv_py3_pyros) PS PYROS>
```

10. Tests

\test

Blabla.

10.1. Philosophy of tests

Blabla.

10.2. Unit tests

NB: For running these tests, we will use the “pyros.py” helper script, but you could do the same thing with “python manage.py test”

*When executing the tests, django creates temporarily a **Mysql database** named “**test_pyros**” (see `src/pyros/settings.py`) that it destroys in the end, so you won't ever see it with `phpmyadmin` (or very quickly and then it will vanish).*

*Some of these tests use the data fixture `/src/misc/fixtures/tests/alert_mgr_test.json`
The scheduler tests do not use any json fixture.*

Be sure that at least all unit tests pass:

```
(venv) $ ./pyros.py unittest
(Windows cmd) (venv) $ python pyros.py unittest
```

(If ever the tests don't pass because of mysql try : `$ python pyros.py updatedb`)

If unit tests pass, then try this :

```
(venv) $ ./pyros.py test_all
```

(for now, same tests than unittest)

If `test_all` passes, then **run ALL tests**:

```
(venv) $ ./pyros.py test
```

If previous step passes, then **run the Majordome test**:

```
(venv) $ cd src/majordome/
(venv) $ ./majordome_test.py
```

NB: if this test does not stop properly, try this:

```
$ ps -ef|grep agent
```



```
$ kill <pid_of_the_start_agent_majordome.py_process>
```

Attention, si le test s'est mal passé, vérifiez que `src/pyros/settings.py` contient bien `MAJORDOME_TEST = False`

(et non pas "True")

Sinon, il faut absolument le remettre à False car sinon pyros ne fonctionnera plus !!

(le test `majordome_test.py` passe cette variable à True puis la repasse à False à la fin)

Technical information if you are interested (you can skip it, it is only for dev):

Here is how to run the tests for only 1 django app, for instance the `user_manager` app:

```
(venv) $ ./manage.py test user_manager.tests.UserManagerTests
```

If you want to run only 1 specific test of this app, for instance the `test_login` test:

```
(venv) $ ./manage.py test user_manager.tests.UserManagerTests.test_login
```

(NB: You can use the "-k" option if you want to keep the test database : `./manage.py test -k ...`)

10.3. Testing agents

You can launch 2 agents (agentA and agentB) in "simulation mode".

They will send commands to each other, and stop both automatically after 1 or 2 minutes max (about 30 iterations of agentB).

```
$ [python] ./pyros.py -t start agentA,agentB
```

At end of execution, each agent will check its commands statuses (only for the commands it sent itself).

You should get something like this :

- For agentB:

```
-----***** Finished testing => result is ok (6 assertions) *****-----
```

- For agentA:

```
-----***** Finished testing => result is ok (9 assertions) *****-----
```

10.4. (OLD) Bigger tests

These demonstration tests are **dedicated to the Scheduler**. You can see the scheduler coming alive.

When executing these big tests, django creates temporarily a **Mysql database** named **"pyros_test"** (see `src/pyros/settings.py` when `CELERY_TEST=True`) that it destroys in the end. But, as this test is quite long, you have the time to see the `pyros_test` database with `phpmyadmin` (for instance you can look at the content of the `"sequence"` table and see it growing). These tests use the data fixture `/src/misc/fixtures/initial_fixture.json`

Important Note:

For now, this is still using Celery, so for this to run ok you must first :

- have RabbitMQ running
- Set `USE_CELERY = True` in `/pyros.py` and `src/pyros/settings.py`

From the same terminal, start the 3 necessary components at once :

- **web server**
- **Celery workers**
- **the simulator(s)** (placeholders for real devices)

To do this, it is as simple as launching a single script (see below)

10.4.1. Demo 1 : with only the Users simulator activated

```
(first start your venv)
(venv)$ ./pyros.py simulator_development
```

(Ctrl-c to stop)

(si pb pour stopper serveur web : \$ ps aux | grep runserver)

(si pb pour stopper celery : \$ ps aux | grep celery)

(If ever this test does not run properly, try to create manually yourself the "pyros_test" database before, with the mysql client : "create database pyros_test")

When you are asked this question (from the terminal) :

"Which simulation do you want to use ? (default="conf.json")

Then just type ENTER so that the scenario `/simulators/config/conf.json` is used for each simulator(s).

Now, access the PyROS website :

go to "http://localhost:8000" in your browser

Log in with login 'pyros' (in the Email field) and password 'DjangoPyros' to see what's happening, and click on **Schedule** to see the current scheduling process.
(You also can click on System, or on Routines and then click on a request to see its detail)

10.4.2. Demo2 : with all simulators activated

```
(first start your venv)
(venv)$ ./pyros.py simulator
```

(Ctrl-c to stop)

(si pb pour stopper serveur web : \$ ps aux | grep runserver)

(si pb pour stopper celery : \$ ps aux | grep celery)

As for demo1, Now you can access the PyROS website... (same instructions as above in DEMO 1)

Custom commands (for dev only) :

(first, activate your venv)

```
$ cd src/
```

```
$ [./manage.py] test app.tests # Run tests for the application 'app'
```

Ex:

```
$ ./manage.py test scheduler
```

```
$ ./manage.py test monitoring.tests
```

```
$ ./manage.py test routine_manager.tests
```

```
$ ./manage.py test alert_manager.tests
```

```
$ ./manage.py test common.tests
```

```
$ ./manage.py test majordome.tests
```

```
$ ./manage.py test user_manager.tests
```

```
$ [./manage.py] test app.tests.ModelTests # Run test methods declared in the class
app.tests.ModelTests
```

```
$ [./manage.py] test app.tests.ModelTests.test_method # Only run the method test_method
declared in app.tests.ModelTests
```

11. Running PyROS

\run

(updated 29/3/19 - EP)

11.1. Start only the webserver (for pyros website)

Using the pyros.py launch script :

```
$ ./pyros.py start webserver
```

Then, connect to <http://localhost:8000>

You can also connect to the admin interface: <http://localhost:8000/admin>

Or, without pyros.py :

Dans un nouveau terminal, activer l'environnement virtuel...

```
$ cd PYROS/  
$ source private/venv_py3_pyros/bin/activate
```

(Windows: \$ private\venv_py3_pyros\Scripts\activate)

... puis lancer le serveur web django :

```
(venv) $ cd PYROS/  
(venv) $ ./start_agent.py webserver  
OR :  
(venv) $ cd PYROS/src/  
(venv) $ ./manage.py runserver
```

(Windows: \$ python manage.py runserver)

General syntax is :

```
$ ./manage.py runserver IP:PORT
```

Example: \$./manage.py runserver localhost:8001

(obsolète: To check that this service is actually running, type "\$ netstat -an |grep 8000" and you should get "tcp 0 0 127.0.0.1:8000 0.0.0.0:* LISTEN")

NB: Pour désactiver l'environnement virtuel, taper "\$ deactivate" depuis n'importe où

11.1.1. Now, you can access the PyROS website

Go to "http://localhost:8000" in your browser

Login as 'pyros' with the password 'DjangoPyros'

You can click on the different sections on the left (Schedule, System, Alerts ...).

⇒ **As you can notice, those sections are all empty !!!**

Of course, because there is no activity at all : no alert coming, no observation request submitted by users, no running observation...

If you want to see something, you need to take some actions yourself on PyROS. For instance, you could create a new Routine Request and submit it so that it will be scheduled and executed...

That's why it is interesting to use **simulators**. They are a placeholder for the real hardware devices (Telescope, Cameras, PLC), and they will be used when executing an observation request. But this is not enough !

We need some **events** coming like a **GRB alert**, an **observation request submitted** by a user, a **weather alarm** (rain, clouds, wind...), a **site alarm** (human intrusion...), or even a **hardware failure** (visible camera is no more responding...).

11.1.2. You also can access the PyROS web administration interface

Go to "<http://localhost:8000/admin>" in your browser

Login as 'pyros' with the password 'DjangoPyros'

From this interface, you can see all the pyros database tables and you can add content to them or do some modifications...

11.2. Start any agent

De manière générale, chaque agent (monitoring, alert, et majordome) peut être lancé individuellement :

Using the pyros.py launch script :

```
$ cd PYROS/  
$ [python] ./pyros.py start <agent-name>
```

Ex: to start the env-monitoring agent:

```
$ ./pyros.py start monitoring
```

Ex: to start agentA:

```
$ [python] ./pyros.py start agentA
```

To start it in TEST (simulator) mode, just add option “-t” :

```
$ [python] ./pyros.py -t start agentA
```

Or, without pyros.py :

First, activate the virtual environment, then :

```
(venv) $ cd PYROS/  
(venv) $ cd src/agent/  
(venv) $ [python] ./AgentA.py
```

Now, you can also start MANY agents at once if you need:

```
$ [python] ./pyros.py start <agent1-name>,<agent2-name>,...
```

Ex: start agentA and agentB

```
$ [python] ./pyros.py start agentA,agentB
```

To start them in TEST (simulator) mode, just add option “-t” :

```
$ [python] ./pyros.py -t start agentA,agentB
```

Ex: start web server and agentB

```
$ [python] ./pyros.py start webserver,agentB
```

You could also start ALL agents at once:

```
$ [python] ./pyros.py start all
```

Now, with your browser, connect to <http://localhost:8000>

(login 'pyros' / 'DjangoPyros')

You can also connect to the admin interface: <http://localhost:8000/admin>

11.3. Play with the pyros shell (pyros interactive session)

`\play \shell`

(updated 29/3/19 - EP)

Using `pyros.py`:

```
$ cd PYROS/  
$ [python] ./pyros.py shell
```

Or, without `pyros.py` :

First activate the pyros venv (if not already done):

```
$ cd PYROS/  
$ source private/venv_py3_pyros/bin/activate
```

Then, go into the pyros project folder `src/` and launch the django shell :

```
(venv) $ cd src/  
(venv) $ ./manage.py shell  
(InteractiveConsole)
```

Below, we give 2 different examples :

- How to play with a **pyros agent**
- How to play with the **pyros objects**

11.3.1. Play with a pyros agent

(updated 29/3/19 - EP)

- 1) Dans un terminal, lancer un agentB : `[python] ./pyros.py start agentB`
- 2) Dans un autre terminal, démarrer une session pyros shell: `[python] ./pyros shell`

Depuis le pyros shell, vous allez créer un agentA avec lequel vous pourrez envoyer des commandes à agentB. Voici comment faire.

Dans le pyros shell, taper les instructions suivantes :

```
>>> from agent.AgentA import AgentA  
(ou encore : "from agent.Agent import Agent")
```

Starting with this sys.path

...

DB2 used is: pyros

Creating a new agent (agentA) :

```
>>> agent=AgentA('agent_toto')
(ou encore: "a=Agent(agent_toto)")
(moi): [LAUNCHED]
(moi): [NEW MODE IDLE]
(moi): [NEW MODE IDLE]
(moi): [NEW MODE ACTIVE]
```

```
>>> agent
I am agent agent_toto
>>> print(agent)
I am agent agent_toto
```

Loading agent config :

```
>>> agent.load_config()
----- Components of the unit -----
Configuration file is
/Users/epallier/Documents/_W_more/PROJECTS/GFT/SOFT/PYROS_SOFT/PYROS201806/c
onfig/config_unit_simulunit1.xml
Unit alias is SimulUnit1. Description is None:
- database alias is PyROSDatabase. Description is None
- mount alias is VirtualMount. Description is None
- channel alias is VirtualCamera1. Description is None
- computer alias is Computer1. Description is None
- computer alias is Computer2. Description is None
- computer alias is Computer3. Description is None
- agent alias is Superagent1. Description is None
- agent alias is AgentX. Description is None
- agent alias is AgentY. Description is None
- agent alias is TelescopeSimulator. Description is None
- agent alias is CameraSimulator. Description is None
- agent alias is Monitoring. Description is None
----- Assembly of the unit -----
Assembled mount aliases: ['VirtualMount']
Assembled channel aliases: ['VirtualCamera1']
```



```
-----  
hostname = macp1219.irap.omp.eu  
path_data = ../../config  
-----
```

Different actions on the agent :

```
>>> agent.init()  
(agent_toto): [INITIALIZING]  
  
>>> agent.show_mode_and_status()  
(agent_toto): CURRENT MODE is ACTIVE (status is INITIALIZING)  
  
>>> agent.update_survey()  
  
>>> agent.purge_old_commands_sent_to_me()  
(Looking for commands sent to me that are not executing and older than 1.0 hour(s))  
<None>
```

Get next command and process it :

```
>>> cmd = agent.get_next_valid_command()  
(agent_toto): [IN_GET_NEXT_COMMAND]  
>>> cmd  
>>> cmd is None  
True  
  
>>> if cmd: cmd = self.command_process(cmd)  
...
```

Execute routine processing :

```
>>> agent.routine_process()  
(agent_toto): [IN_ROUTINE_PROCESS]
```

Send commands to another agent (agentB) :

```
>>> cmd1 = agent.send_command('AgentB', 'eval 2+2')  
>>> cmd2 = agent.send_command('AgentB', 'eval 4+5')
```

```

>>> cmd1
<Command: Commmand 'eval 2+2' (CMD_EXECUTED) sent by agent moi to agent AgentB at
2019-03-22 11:31:39.956434+00:00>
>>> cmd2
<Command: Commmand 'eval 4+5' (CMD_EXECUTED) sent by agent moi to agent AgentB at
2019-03-22 11:31:47.997502+00:00>
>>> cmd1.result
"
>>> cmd1.refresh_from_db()
>>> cmd1.result
'4'
>>> cmd2.get_updated_result()
'9'

```

Send a command and abort it right away :

```

>>> cmd = agent.send_command('AgentB',specific1')
>>> agent.send_command('AgentB','abort')
>>> cmd.refresh_from_db()
>>> cmd.result
'in step #4/5'
>>> cmd.receiver_status_code
'CMD_KILLED'

```

Run N agent iterations (agentA) :

; Run 0 iteration :

```

>>> agent.run(0)
(agent_toto): [IN NORMAL MODE]
(agent_toto): [INITIALIZING]
Delete (false) 'running' command if exists:
<None>

```

; Run 1 iteration :

```

>>> agent.run(1)
(agent_toto): [IN NORMAL MODE]
(agent_toto): [INITIALIZING]
Delete (false) 'running' command if exists:
<None>
(agent_toto): ----- MAIN LOOP ITERATION 1 (START) -----
(agent_toto): [IN_MAIN_LOOP]

```

```
(agent_toto): CURRENT MODE is ACTIVE (status is IN_MAIN_LOOP)
(agent_toto): Waiting 2 sec (random) before starting new iteration...
(agent_toto): Looking for old commands to purge...
(Looking for commands sent to me that are not executing and older than 1.0 hour(s))
<None>
(agent_toto): [IN_GET_NEXT_COMMAND]
(agent_toto): [IN_ROUTINE_PROCESS]
(agent_toto): ----- MAIN LOOP ITERATION (END) -----
Exit because number of iterations asked (1) has been reached
>>>
```

; Run 2 iterations :

```
>>> agent.run(2)
(agent_toto): [IN NORMAL MODE]
(agent_toto): [INITIALIZING]
Delete (false) 'running' command if exists:
<None>
(agent_toto): ----- MAIN LOOP ITERATION 1 (START) -----
(agent_toto): [IN_MAIN_LOOP]
(agent_toto): CURRENT MODE is ACTIVE (status is IN_MAIN_LOOP)
(agent_toto): Waiting 4 sec (random) before starting new iteration...
(agent_toto): Looking for old commands to purge...
(Looking for commands sent to me that are not executing and older than 1.0 hour(s))
<None>
(agent_toto): [IN_GET_NEXT_COMMAND]
(agent_toto): [IN_ROUTINE_PROCESS]
(agent_toto): ----- MAIN LOOP ITERATION (END) -----

(agent_toto): ----- MAIN LOOP ITERATION 2 (START) -----
(agent_toto): [IN_MAIN_LOOP]
(agent_toto): CURRENT MODE is ACTIVE (status is IN_MAIN_LOOP)
(agent_toto): Waiting 2 sec (random) before starting new iteration...
(agent_toto): [IN_GET_NEXT_COMMAND]
(agent_toto): [IN_ROUTINE_PROCESS]
(agent_toto): ----- MAIN LOOP ITERATION (END) -----
Exit because number of iterations asked (2) has been reached
>>>
```

RESTART agentB :

```
>>> cmd = agent.send_command('AgentB','restart_init')
=> restart from iteration 1 again...
```

STOP agentB :

```
>>> cmd = agent.send_command("AgentB","exit")
```

[RELOAD howto : From the pyros shell, how to reload a python module that has just been modified ?](#)

Une fois pour toutes, importer “reload”

```
>>> from importlib import reload
```

```
>>> import agent.AgentA
>>> from agent.AgentA import AgentA
>>> agent = AgentA()
...
```

*A partir de là, je modifie le code source de **AgentA**...
Il faut donc faire un “reload()” pour mettre à jour :*

Ca marche pas du premier coup !!!

```
>>> reload(agent.AgentA)
Traceback (most recent call last):
  File "<console>", line 1, in <module>
AttributeError: 'Agent' object has no attribute 'AgentA'
```

Voici ce qu’il faut faire, en 3 instructions:

```
>>> import agent.AgentA
>>> reload(agent.AgentA)
<module 'agent.AgentA' from
'/Users/epallier/Documents/_W_more/PROJECTS/GFT/SOFT/PYROS_SOFT/PYROS201806/s
rc/agent/AgentA.py'>
>>> from agent.AgentA import AgentA
```

A partir de là, c’est bon, on a la nouvelle version de AgentA :

```
>>> agent=AgentA()
... <nouvelle version> ...
```

Dans le cas où on a modifié **Agent**, là c’est encore plus compliqué, il faut **6 instructions** !!!:

D’abord, recharger Agent... :

```
>>> import agent.Agent
>>> reload(agent.Agent)
>>> from agent.Agent import Agent
... puis recharger AgentA :
>>> import agent.AgentA
>>> reload(agent.AgentA)
>>> from agent.AgentA import AgentA
```

Et là, c’est bon :

```
>>> agent=AgentA('agent_toto')
```

... <nouvelle version> ...

11.3.2. Play with the pyros objects (data model)

Now that your pyros shell session is started, Import all the pyros objects, so that you can create instances of them :

```
>>> from common.models import *
(or also: >>> from common import models)

>>> dir(models)
['AbstractUser', 'Album', 'Alert', 'Country', 'Detector', 'Device', 'Dome', 'Filter', 'FilterWheel',
'Image', 'Log', 'NrtAnalysis', 'Plan', 'Plc', 'PlcDevice', 'PlcDeviceStatus', 'PyrosUser', 'Request',
'Schedule', 'ScheduleHasSequences', 'ScientificProgram', 'Sequence', 'SiteWatch',
'SiteWatchHistory', 'StrategyObs', 'TaskId', 'Telescope', 'UserLevel', 'Version', 'WeatherWatch',
'WeatherWatchHistory', ...]
```

1 - Play with the PyROS objects (entities) : Countries

```
>>> country = Country(name='mexico', quota=1)
>>> country.save()
(ajout si pas d'id, modif si id)

>>> country = Country(name='france')
>>> country.save()
>>> country.pk
>>> 2

>>> countries = Country.objects.all()
>>> countries
<QuerySet [<Country: France>, <Country: mexico>, <Country: france>]>
>>> countries.count
>>> <bound method QuerySet.count of <Country: mexico>, <Country: france>>
>>> countries.count()
>>> 2
>>> print(countries)
>>> <Country: mexico>, <Country: france>
>>> print(countries.query)
>>> SELECT country.id, country.name, country.desc, country.quota FROM country

>>> cs = countries.filter(name__icontains='fran')
>>> print(cs)
>>> <Country: france>
```

```
>>> cs = countries.filter(name__startswith='me')
>>> print(cs)
>>> <Country: mexico>
```

2 - Play with the PyROS objects (entities) : Requests and Sequences

```
# First, create a user
```

```
usr_lvl = UserLevel.objects.create(name="default")
user1 = PyrosUser.objects.create(username="toto", country=country,
user_level=usr_lvl, quota=1111)
sp = ScientificProgram.objects.create(name="default")
```

```
# 1) Create a new sequence (Supposing user1 and sp are already set)
```

```
req = Request.objects.create (pyros_user = user1,      scientific_program = sp
) # creating a request
```

```
seq = Sequence.objects.create ( request=req, status=Sequence.TOBEPANNED,
name="seq1",
jd1=0,
jd2=2,
priority=4,
t_prefered=1,
duration=1 )
```

```
# Get the request that this sequence belongs to :
req2 = seq.request
```

```
# 2) Update sequence attributes
```

```
seq.name = 'new name'
seq.save()
```

```
# 3) Delete sequence
```

```
seq.delete()
```

```
# 4) Fetch sequences according to some criteria
```

```
sequences = Sequence.objects.get(target='target')
```

```
# or
```

```
sequences = Sequence.objects().filter(...)
```

```
# 5) Get all plans of the sequence 1st album
```

```
album1 = seq.albums[0] # select 1st album
```

```
plans = album1.plans
# Display all plans
for plan in plans: print('plan', plan)
```

12. Programming PyROS

This section describe how to program properly PyROS.

Please see annex 1 for “**how to use GIT**”.

Please see annexes 2, 3 and 4 for “**Coding rules**” and “**Tests rules**”.

12.1.2. From PyROS objects to database tables

All the database tables are described and managed from the pyros python file **src/common/models.py**

Nearly each table in the database corresponds to a class (subclass of django.db.models.Model) in the models.py file.

The standard development workflow is to describe each database table inside the models.py file, and then update the database structure from this description, either for the creation a new table or for the modification of an already existing one. The updating of the database is simply done with the commande “pyros.py update”.

12.1.3. HOW TO add a new table in the database

Let’s see this with an example.

We want to add a new “agents_survey” table, in order to survey the current status of all the agents.

Step 1 - Add a new AgentsSurvey model class in the MODELS description file (src/common/models.py)

```
class AgentsSurvey(models.Model):
    """
    | id | created | validity_duration_sec (default=1mn) | name | mode (active/idle) | status |
    """
    created = models.DateTimeField(blank=True, null=True, auto_now_add=True)
    validity_duration_sec = models.IntegerField(blank=True, null=True)
    name = models.CharField(max_length=50, blank=True, null=True)
    mode = models.CharField(max_length=15, blank=True, null=True)
    status = models.CharField(max_length=15, blank=True, null=True)

    class Meta:
        managed = True
        db_table = 'agents_survey'

    def __str__(self):
        return (f"Agent {self.name} at {self.created} in mode {self.mode} and status
{self.status}")
```

Step 2 - Create the django “migration” python file that will be used to create the new table

First, activate the venv

```
$ source private/venv_py3_pyros/bin/activate
```

Then, create a migration file

```
(venv) $ cd PYROS/src/
```

```
(venv) $ ./manage.py makemigrations
```

A migration file has been created in the migrations folder (src/common/migrations)

```
$ ls common/migrations/
```

```
0005_agentssurvey.py
```

This file contains the python code that will create the new table in the database :

```
# Generated by Django 2.0.5 on 2019-02-21 13:35
```

```
from django.db import migrations, models
```

```
class Migration(migrations.Migration):
```

```
    dependencies = [
        ('common', '0004_telescopecommand'),
    ]

    operations = [
        migrations.CreateModel(
            name='AgentsSurvey',
            fields=[
                ('id', models.AutoField(auto_created=True, primary_key=True, serialize=False, verbose_name='ID')),
                ('created', models.DateTimeField(auto_now_add=True, null=True)),
                ('validity_duration_sec', models.IntegerField(blank=True, null=True)),
                ('name', models.CharField(blank=True, max_length=50, null=True)),
                ('mode', models.CharField(blank=True, max_length=15, null=True)),
                ('status', models.CharField(blank=True, max_length=15, null=True)),
            ],
            options={
                'db_table': 'agents_survey',
                'managed': True,
            },
        ),
    ]
```

Step 3 - Apply the migration file to add the new table in the database

```
(venv) $ ./manage.py migrate
```

Step 4 - Now, you can play with your new pyros object

First, launch a pyros shell:

```
$ cd PYROS/  
$ ./pyros shell
```

Then, import all the pyros objects :

```
>>> from common.models import *
```

Now, create a new AgentsSurvey object (instance) that we will call “asurv1” :

```
>>> asurv1 = AgentsSurvey()
```

```
>>> asurv1
```

```
<AgentsSurvey: Agent None at None in mode None and status None>
```

```
>>> asurv1 = AgentsSurvey(name='agentX', mode='idle')
```

```
>>> asurv1
```

```
<AgentsSurvey: Agent agentX at None in mode idle and status None>
```

```
>>> asurv1.name
```

```
'agentX'
```

```
>>> asurv1.created
```

```
=> nothing yet, because your object is not yet saved into the DB !!
```

```
>>> asurv1.save()
```

```
>>> asurv1.created
```

```
datetime.datetime(2019, 2, 21, 14, 5, 47, 728756)
```

```
>>> asurv1
```

```
<AgentsSurvey: Agent agentX at 2019-02-21 14:05:47.728756 in mode idle and status None>
```

```
>>>
```

Now, let's create a second instance, named asurv2, and save it directly to the DB :

```
>>> asurv2 = AgentsSurvey.objects.create(name='agentY', mode='active')
```

```
>>> asurv2
```

```
<AgentsSurvey: Agent agentY at 2019-02-21 16:31:36.351752 in mode active and status None>
```

How to get back these objects from the database ?

Get all the saved objects :

```
>>> all = AgentsSurvey.objects.all()
>>> all[0]
<AgentsSurvey: Agent agentX at 2019-02-21 14:05:47.728756 in mode idle and status None>
>>> all[1]
<AgentsSurvey: Agent agent2 at 2019-02-21 15:11:06.308349 in mode idle and status None>
```

Get a specific object :

By id :

```
>>> obj1 = AgentsSurvey.objects.get(id=1)
>>> obj1
<AgentsSurvey: Agent agentX at 2019-02-21 14:05:47.728756 in mode idle and status None>
```

By name :

```
obj2 = AgentsSurvey.objects.get(name="agentX")
```

Delete an object:

```
>>> obj = all[2]
>>> obj.delete()
```

Other queries examples :

intersection ("**et**" logique)

```
my_user = User.objects.get(is_staff=False, is_active=True)
```

disjonction ("**ou**" logique)

```
my_other_user = User.objects.get(
    Q(username__startswith='Jean') | Q(date_joined__year=2014)
)
```

select related objects

```
backlog = ProductBacklog.objects.get(pk=backlog_id).select_related('team')
```

12.1.4. Access the pyros database

You can access to the database (and manage the pyros database and its tables) by using the pyros "dbshell" command :

```
$ cd PYROS/
$ pyros.py dbshell
```

This is equivalent to activating the venv and running the command "django manage.py script" (from the src folder)

This is also equivalent to the mysql command “mysql -u pyros -p” and then typing “use database pyros;”

12.1.5. Re-initialize the pyros database

```
$ ./pyros.py initdb
```

*(the initdb command will load a pyros initial fixture from the file src/misc/fixtures/**initial_fixture**.json)*

12.1.6. Update the pyros database

```
$ ./pyros.py update
```

This command will sync your database structure from the src/common/models.py models description file. See chapter about [“Updating pyros”](#) for more details.

13. PYROS MODULES (main functions)

13.1. BIG FUNCTIONS (modules)

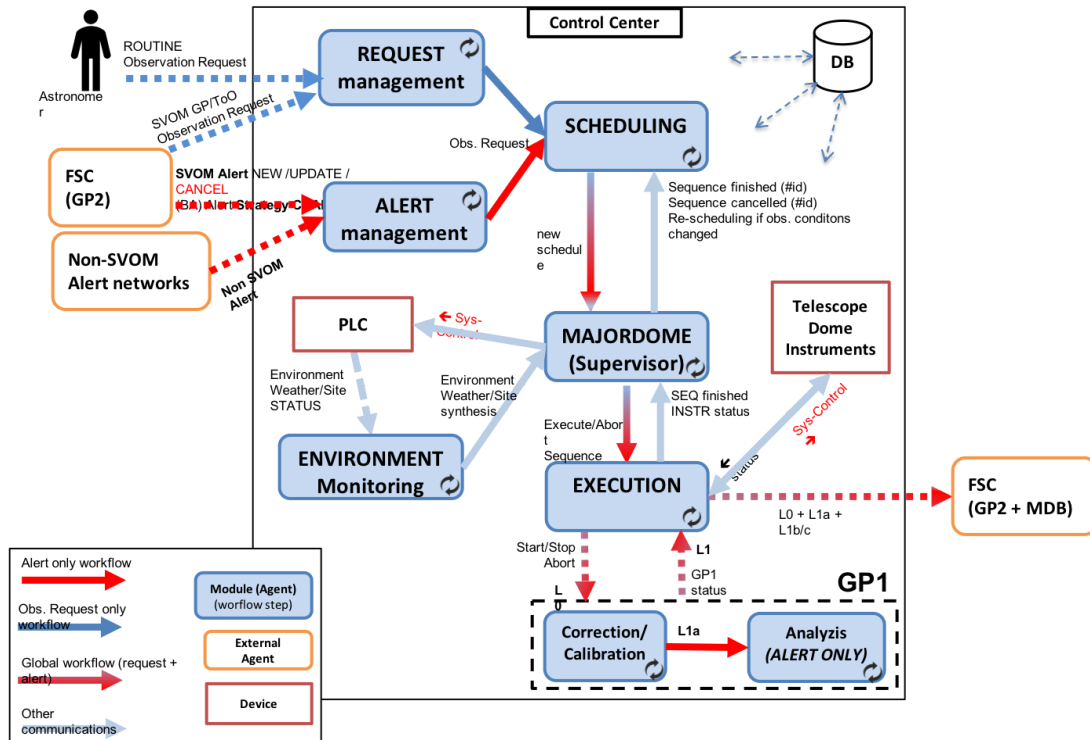
(updated 4/7/18)

GFT-REQ-147: The **COLIBRI software** shall manage the following **functions** (see Figure 1):

- 1 - Alert management
- 2 - Observation Request management (*routine management*)
- 3 - Planning
- 4 - Observation execution (cd-ctrl) & Instruments monitoring
- 5 - Environment Monitoring (inside & outside observatory) for human & instruments safety
- **6 - Data reduction & analysis**
- 7 - Information system management (**Dashboard**)
- **8 - Scientific Programs management**
- **9 - Telescope & Instruments long term Monitoring & Calibration**
- **10 - Data archiving** (short and long term)

13.2. MAIN WORKFLOW

Main workflow: Alerts and Observation Requests management



1

All modules are under the “/src/” directory

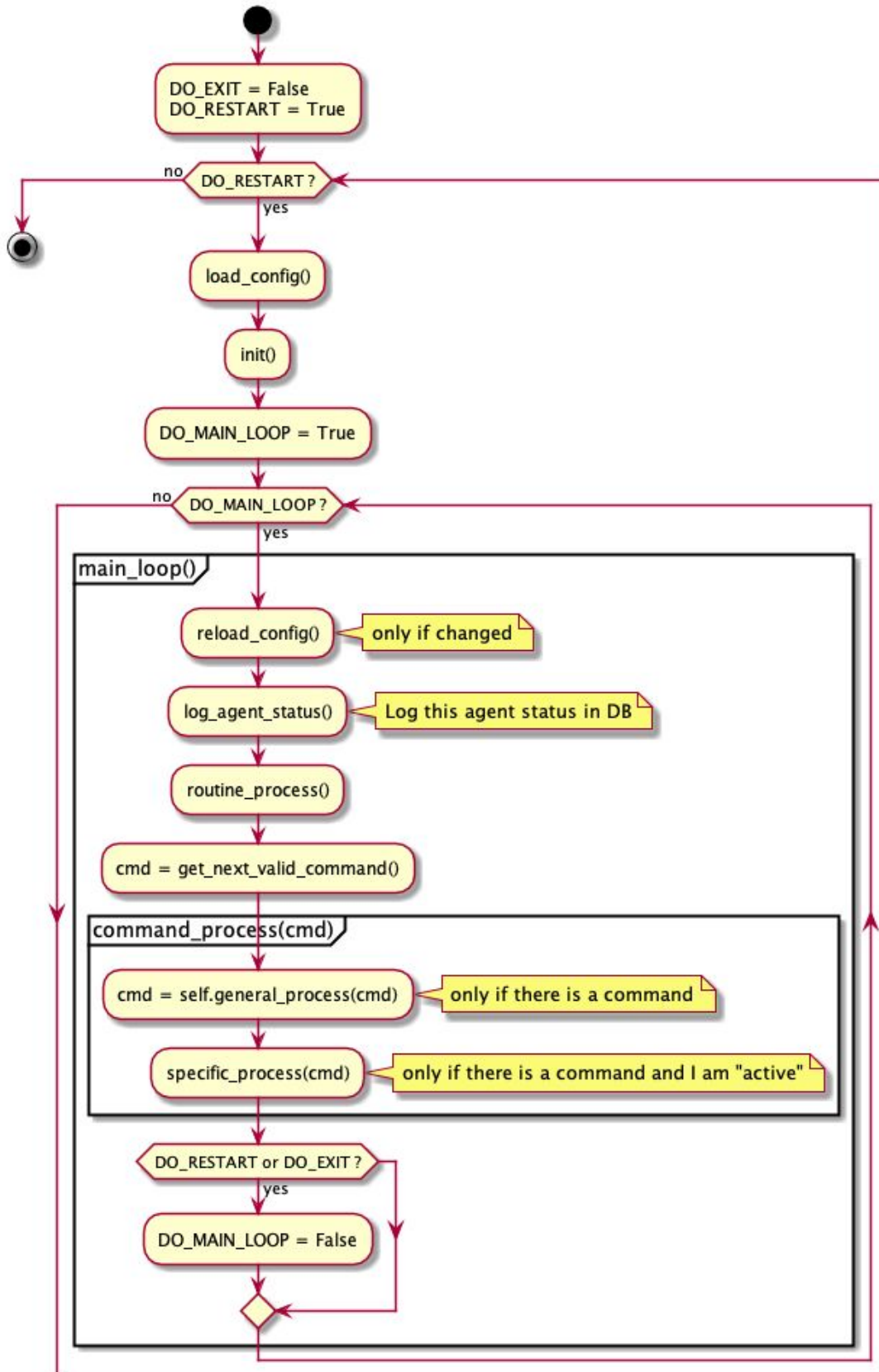
	Agent ?	Place	Start
(ENV) Environment Monitor <i>(P. Maeght)</i>	YES	src/monitoring/tasks.py Monitoring.run()	\$ pyros.py start_agent_envmonitor
(SCHED) Scheduler (Planner) <i>(A. Klotz)</i>	(NO)	src/scheduler/Scheduler.py (and simulator.py) and tasks.scheduling.run()	Appel de la méthode scheduler.tasks.scheduling.run()
(MAJ) Majordome	YES	src/majordome/tasks.py Majordome.run()	\$ pyros.py start_agent_majordome.py

(ALERT) Alert Manager	YES	src/alert_manager/tasks.py AlertListener.run()	\$ pyros.py start_agent_alert_manager.py
(REQ) Request Manager	NO		
(EYE) Observer (Executor)	(NO)		
(CAL) Calibrator (A.K & K. Noysena)	NO		
(NRTA) NRT Analyzer (A.K & K. Noysena)	NO		

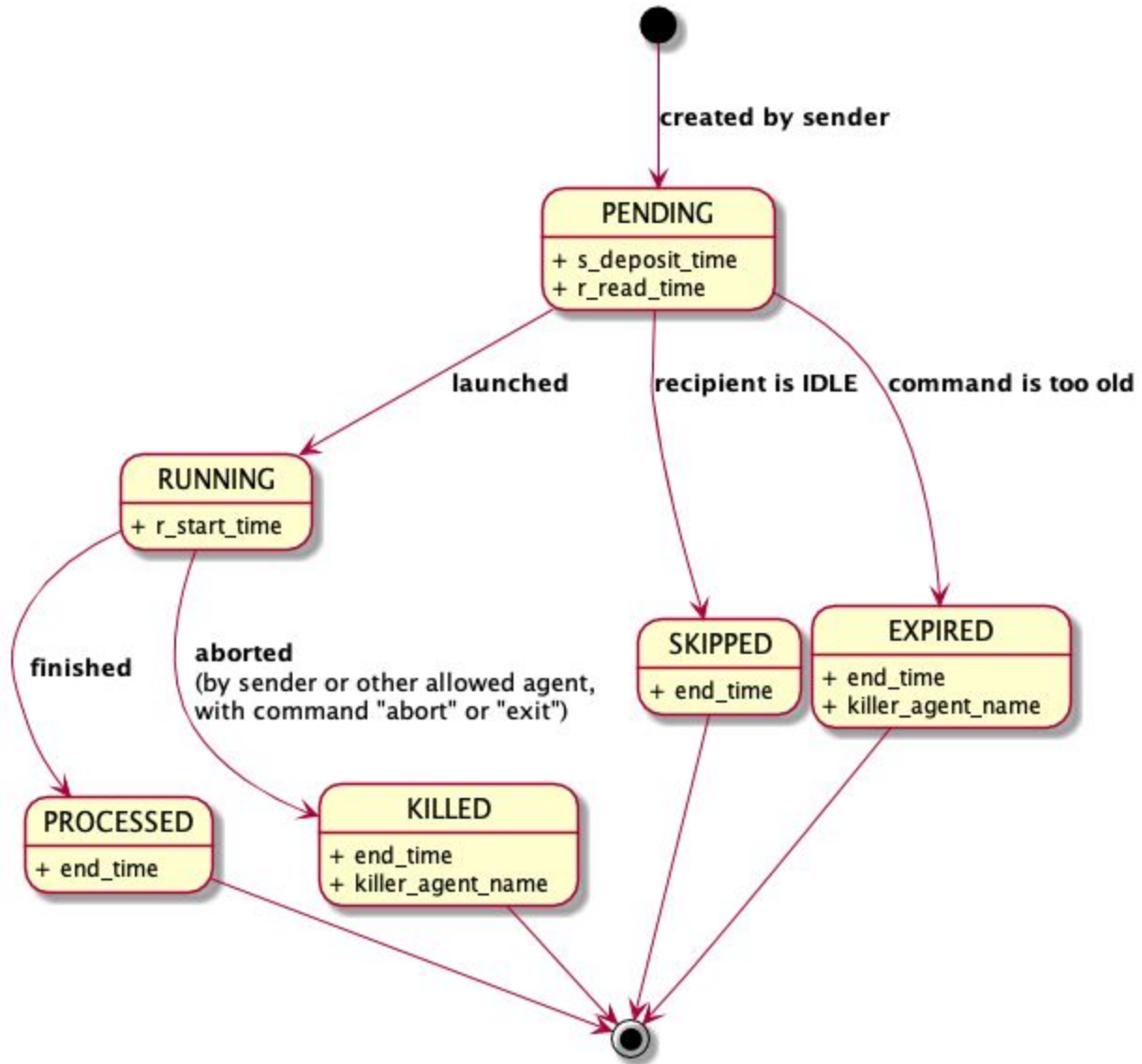
13.3. FUNCTION 0 - Agent & Command

\agent \command

Agent.run() function : Activity Diagram



models.Command class : State Diagram



Action	SENDER	RECIPIENT
PENDING	Tag "expired" if expired	Tag "expired" if expired
RUNNING	Tag "timedout" (or "expired" ?) if timed out	N/A
"Do abort"	N/A	Abort current executing command

"Do flush"	N/A	Delete all pending commands
"Do exit"	N/A	Abort current executing command Then, exit
"Set status Active"	N/A	Now status is active
"Set status Idle"	N/A	<p>Now status is idle</p> <p>Only do:</p> <ul style="list-style-type: none"> - routine_process() - Generic commands execution <p>No more do:</p> <ul style="list-style-type: none"> - Specific command execution : set them to "SKIPPED"

13.4. FUNCTION 1 - Alert Management (ALERT)

(in `src/alert_manager/tasks.py`)

⇒ `src/alert_manager/tasks.py/class AlertListener(Task)` :

⇒ `run()` : LOOP, each 1s check if new VOEvent to process and if so process them (parse, then create and save a request) :

⇒ `analyze_event(event)`

⇒ `create_related_request()`

⇒ `req = create_request_from_strategy()`

⇒ `req.validate()`

⇒ `req.save()`

⇒ `scheduler.tasks.scheduling.delay(first_schedule=True, alert=self.request.is_alert)`

Détail de la méthode `run()` :

```
def run(self):
```

```
    self.old_files = [f for f in os.listdir(VOEVENTS_PATH) if isfile(join(VOEVENTS_PATH, f))]
```

```
    Log.objects.create(agent="Alert manager", message="Start alert manager")
```

```
    while True:
```

```
        if (settings.DEBUG and DEBUG_FILE):
```

```
            log.info("Checking fresh events")
```

```
        fresh_events = self.get_fresh_events()
```

```
        for event in fresh_events:
```

```
            self.analyze_event(event)
```

```
            if (settings.DEBUG):
```

```
                log.info("Analyzed event : " + str(event))
```

```
        time.sleep(1)
```

13.5. FUNCTION 2 - Observation Request Management (REQUEST)

(TODO:)

13.6. FUNCTION 3 - Planning (PLANNER, SCHEDULER)

(updated 23/04/18)

Responsible : Alain Klotz

Fonctions de ce module :

- Get the list of sequences to be planned
- Plan sequence according to priorities, quotas, observing conditions, and sequence parameters
- Validate and save schedule
- Automatic Schedule update

13.6.1. Specs

Heure de référence = heure UTC (GMT)

(AK: pas besoin d'afficher l'heure locale)

Telescope = monture (mount)

On aura un autre canal (camera) qui permettra de mesurer la qualité du ciel (il sera sur la monture)

Unit ("façade") = 1 mount (telescope) + N canaux

Composition : On compose une unit avec : une monture + des canaux

Une "unit" décrit un telescope et son environnement.

Node = N units = Colibri + GWAC-test + GFT chinois + ...

= GFTs

Scheduler est au niveau d'une seul unit

Periode : 6 mois

1 SP = 1 quota et souvent 1 observer only

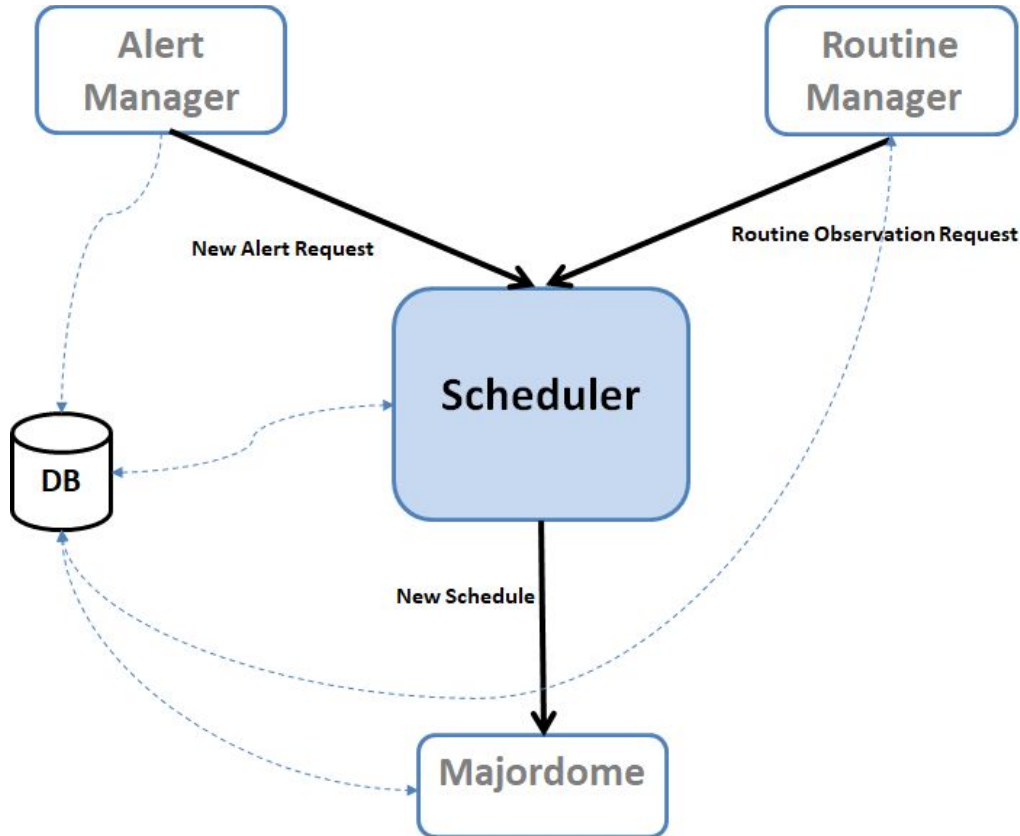
1 nuit = 24h = midi à midi

Ligne de visibilité = par pallier (toujours positif, 0 = visible) : en pointillé

Ligne (noire) de disponibilité du Tele

Observation : BESTELEV ou IMMEDIATE

The Scheduler module interfaces with other modules (inputs on top)



13.6.2. Deux phases principales de test

Phase 1 : tester le module Scheduler de manière “statique”, en “isolation”, c’est à dire seulement la fonction de planification toute seule :

- D’abord, **tester une planification “one shot”** (une seule planification et c’est fini) en vérifiant que les plannings obtenus selon différents lots de Sequences fournis en input (fixtures) sont bien ceux espérés (des séquences “simplistes” seront dans un premier temps créées en dur dans le code puis, dans un deuxième temps, on charger des séquences plus réalistes sous forme de fichiers XML ingérés dans la BD via RequestBuilder) :
 - Input = fixture1 ⇒ output = planning1,
 - Input = fixture2 ⇒ output = planning2,
 - ...,
 - Input = fixtureN ⇒ output = planningN,

- Ensuite, **tester plusieurs “re-planifications” consécutives** : d’abord on planifie quelques séquences, puis on en ajoute 1 ou 2 autres, on re-planifie, et ainsi de suite... et vérifier que les plannings obtenus à chaque étape sont conformes à ce qui est attendu

Phase 2 : tester le module Scheduler **de manière “dynamique”, dans le contexte PyROS**, c’est à dire avec des Sequences soumises “au fil de l’eau” par l’utilisateur (User simulator) ou/et par l’agent AlertManager (replanif à chaque fois qu’une nouvelle séquence arrive), qui sont exécutées au fur et à mesure (et donc leur statut change dans le planning, et replanif), avec des “conditions d’observation” qui évoluent (et donc replanif), et enfin des alarmes “météo” et “site” qui viennent perturber le tout (envoyées par la collaboration PLC et Monitoring et lues par le Majordome)..., bref tout un (très gros) programme !

Processus de développement proposé :

- Version 1 : tester les plannings obtenus avec des Sequences soumises “au fil de l’eau” par l’utilisateur (User simulator)
- Version 2 : Version 1 + Sequences alertes soumises par l’agent AlertManager
- Version 3 : Version 2 + changement des conditions d’observation
- Version 4 : Version 3 + alarmes météo ou/et site
- Version 5 : Version 4 + exécution des séquences

13.6.3. Exécution des tests existants

Il existe déjà 14 tests unitaires dédiés au Scheduler actuel.
Ces tests sont dans `src/scheduler/tests.py`

Pour les exécuter, activer l’environnement virtuel, puis :

```
(venv) $ cd src/
(venv) $ python manage.py test scheduler.tests

Creating test database for alias 'default'...

===== TEST_3_SEQ_MOVE_BOTH =====

.
===== TEST_3_SEQ_MOVE_LEFT =====

.
===== TEST_3_SEQ_MOVE_RIGHT =====
```

```

.
===== TEST_3_SEQ_PRIORITY =====

.
===== TEST_3_SEQ_PRIORITY_OVERLAP =====

.
...

Duration : 0.0984950065612793
.
-----
Ran 14 tests in 0.567s

OK
Destroying test database for alias 'default'...

```

13.6.4. Jouer avec le Scheduler (via le django shell)

Activer l'environnement virtuel, puis :

```
# Lancer le django shell
```

```
(venv) $ cd src/
```

```
(venv) $ python manage.py shell
```

```
# Créer une instance du Scheduler
```

```
>>> from scheduler.Scheduler import Scheduler
```

```
>>> scheduler = Scheduler()
```

```
>>> scheduler.max_overhead = 1
```

```
# Créer un utilisateur usr1 (dans la BD)
```

```
>>> from common.models import *
```

```
>>> c = Country.objects.create()
```

```
>>> sp = ScientificProgram.objects.create()
```

```
>>> ul = UserLevel.objects.create()
```

```
>>> usr1 = PyrosUser.objects.create(username="toto", country=c, user_level=ul, quota=100)
```

```
>>> usr1
```

```
<PyrosUser: toto>
```

```
# Créer une requete req1 (dans la BD)
>>> req1 = Request.objects.create(name="my request 1", pyros_user=usr1,
scientific_program=sp)
>>> req1
<Request: my request 1>
```

```
# Créer une requete req2 (dans la BD)
>>> req2 = Request.objects.create(name="my request 2", pyros_user=usr1,
scientific_program=sp)
>>> req2
<Request: my request 2>
```

Créer des séquences pour ces requetes

```
# Attention, s'il y a déjà des séquences dans la BD, il vaut mieux les supprimer avant :
sequences = Sequence.objects.all()
for s in sequences: s.delete()
```

```
# Création de 3 nouvelles séquences (dans la BD)
```

```
>>> seq11 = Sequence.objects.create(request=req1, status=Sequence.TOBEPLANNED,
name="seq1.1", jd1=0, jd2=2, priority=1, t_prefered=-1, duration=1)
>>> seq11
<Sequence: seq1.1>
```

```
>>> seq12 = Sequence.objects.create(request=req1, status=Sequence.TOBEPLANNED,
name="seq1.2", jd1=4, jd2=6, priority=1, t_prefered=-1, duration=1)
>>> seq12
<Sequence: seq1.2>
```

```
>>> seq13 = Sequence.objects.create(request=req1, status=Sequence.TOBEPLANNED,
name="seq1.3", jd1=7, jd2=9, priority=1, t_prefered=-1, duration=1)
```

```
# On aurait aussi pu créer cette nouvelle séquence par copie d'une autre :
```

```
>>> import copy
>>> seq13 = copy.copy(seq12)
>>> seq13
<Sequence: seq1.2>
>>> seq13.name = "seq1.3"
>>> seq13.jd1 = 7
>>> seq13.jd2 = 9
>>> seq13
```

```
<Sequence: seq1.3>
```

```
# Voyons quelles sequences sont contenues dans la requete req1:
```

```
>>> req1.sequences.all()
```

```
<QuerySet [<Sequence: seq1.1>, <Sequence: seq1.2>, <Sequence: seq1.1>]>
```

```
# On fait une planif
```

```
>>> scheduler.makeSchedule()
```

```
<Schedule: 2018-03-01 15:26:06.139674+00:00>
```

```
# On récupère le dernier planning (c'est à dire celui qu'on vient de créer) :
```

```
>>> schedule = Schedule.objects.order_by('-created').first()
```

```
>>> schedule
```

```
<Schedule: 2018-03-01 15:26:06.139674+00:00>
```

```
# Quelles sont les séquences associées à ce planning (combien) ? :
```

```
>>> shs_list = sched.shs.all()
```

```
>>> nbPlanned = len([shs for shs in shs_list])
```

```
>>> nbPlanned
```

```
3
```

13.6.5. Procédure concrète de soumission des requetes pour les tests :

1 - Créer des fichiers requetes XML dans `simulators/resources/`, tels que par exemple `routine_request_01.xml` :

```
<?xml version="1.0" ?>
```

```
<request submitted="1" relative="1" name="RequestSimulator" scientific_program="GRB" target_type="test">
```

```
  <sequence duration="10" jd1="15" jd2="60000" name="Sequence1 (200secs)" target_coords="10">
```

```
    <album detector="Visible camera" name="alb">
```

```
      <plan duration="10" filter="First infrared filter" name="simulation" nb_images="5"/>
```

```
    </album>
```

```
  </sequence>
```

```
</request>
```

2 - Appeler `RequestBuilder` pour créer un objet `Request` à partir de ce fichier XML, et le mettre dans la BD

3 - Soumettre ces requetes au `Scheduler` qui doit les planifier...

13.7. FUNCTION 4 - Observation EXECUTION & Instruments Monitoring (EXEC, system control)

(updated 23/04/18)

Responsible : Quentin Durand

Includes : MAJORDOME and OBSERVER submodules (+ controleurs & simulateurs Tele et instrum)

Mode manuel des TAROT de AK (pour ref) :

http://cador.obs-hp.fr/ros/manual/cador_actions.html

TODO:

Ajouter :

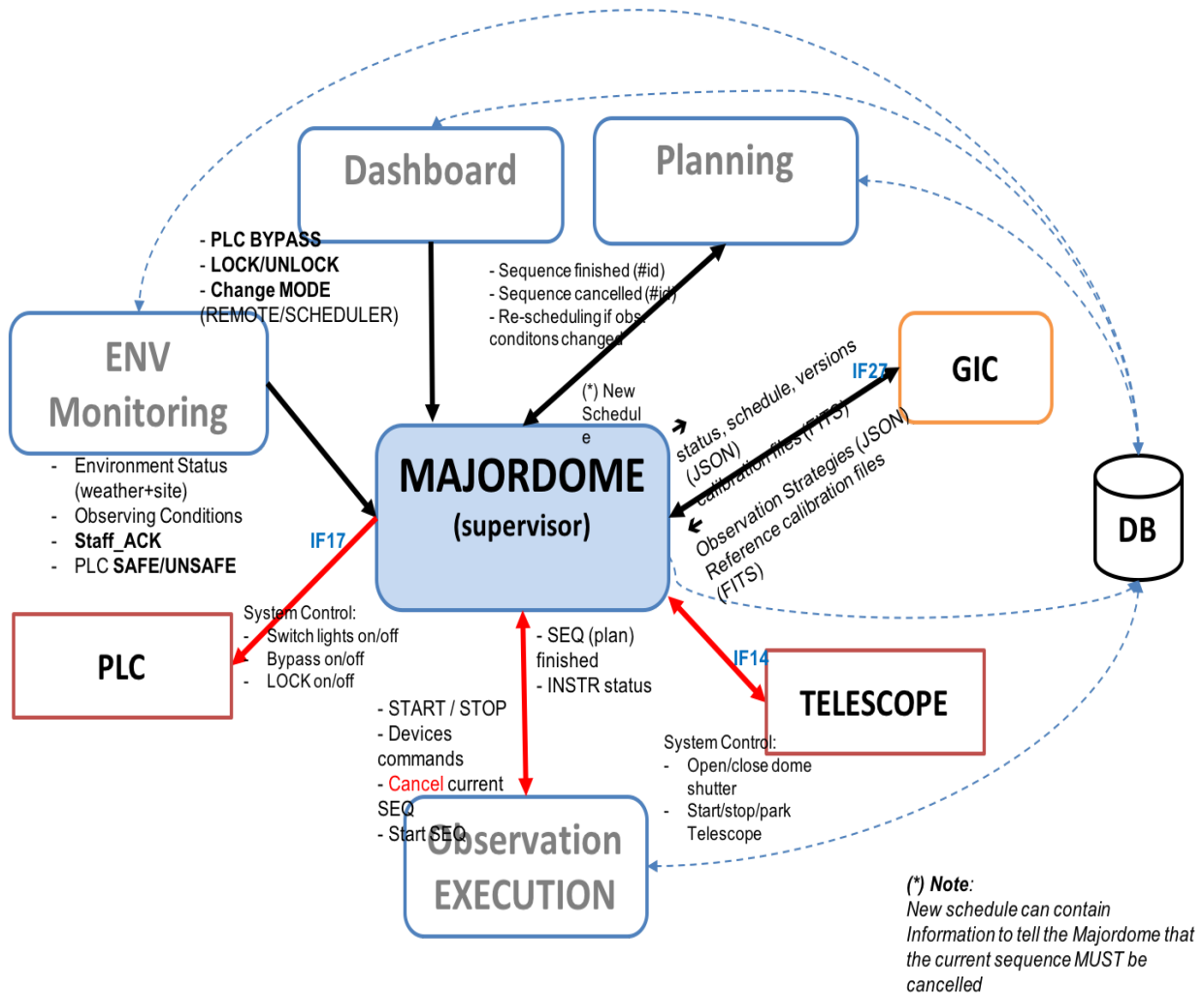
- status meteo
- status jour/nuit
- status infra : toit ouvert...

Fonctions de ce module :

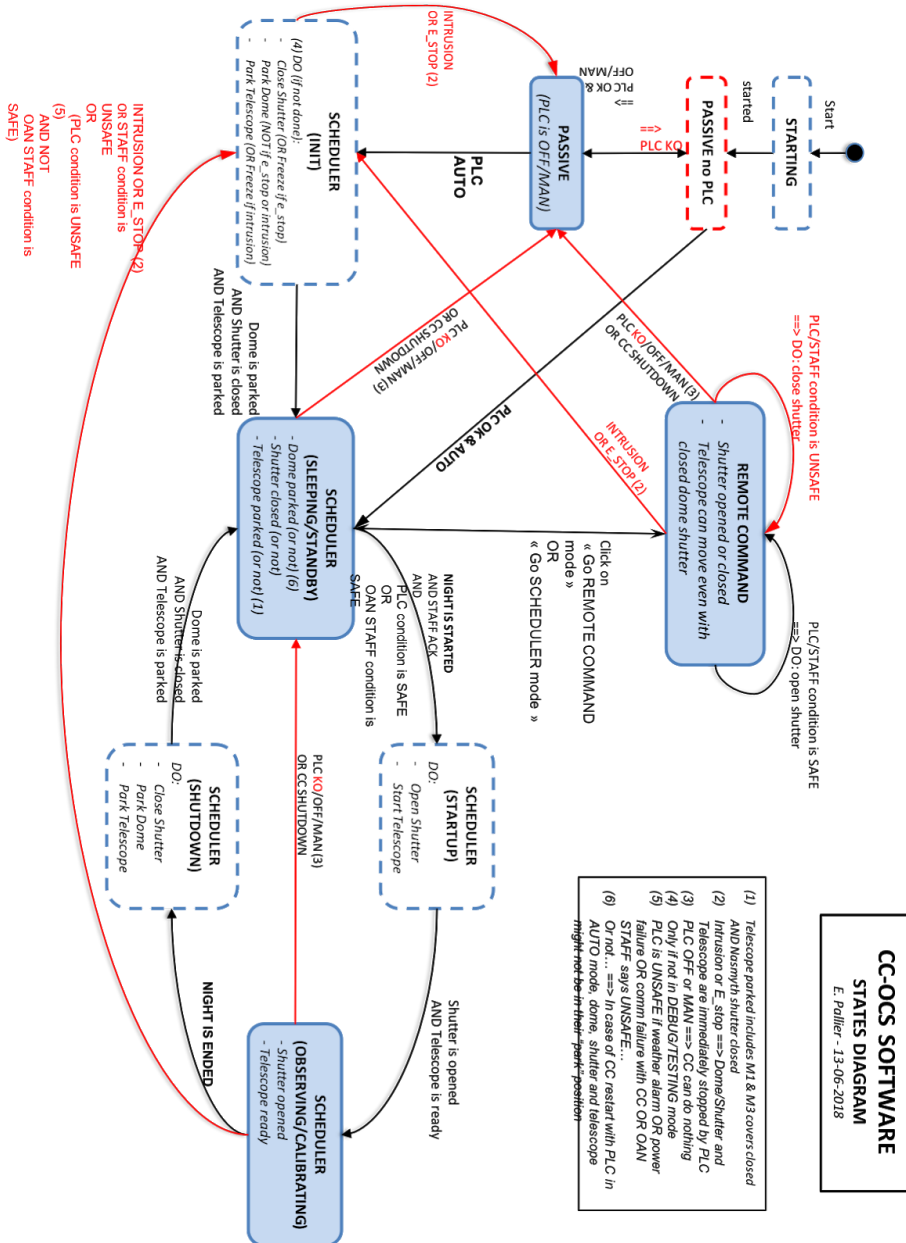
- Check observation conditions
- System control of the telescope & instruments (manuel and auto modes)
- Monitor the telescope & instruments status
- Execute planned observation sequences
- Stop current sequence and run priority sequence instead
- Get raw images from instruments
- Add useful header to images

13.7.1. MODULE “Majordome (Conductor, Master)” (updated 25/7/18, EP)

13.7.1.1. Context diagram



13.7.1.2. States diagram



13.7.1.3. DEVICES STATUS LIST to be sent to GIC

- **Telescope:** environ tous les 6 sec
 - Slewing 0/1
 - Homing 0/1
 - Park 0/1
 - Connected 0/1
 - Stop sensor 0/1 (est-il en fin de course ?)
 - Last connexion establishment (date de dernier début de connexion) : date UTC
 - Position du flat 0 1 (est-il sur la position de l'écran de flat ?)
 - 3 types de coordonnées de position du tele :
 - ra/dec
 - ha/dec
 - alt/az (suivre l'enchaînement des positions dans la nuit)
 - Date de last maintenance

- **DDRAGO & CAGIRE :**
 - Roue à filtre : homing + slewing
 - Position (sur quel filtre)
 - CCD temperature
 - Initialisation de la camera : cam init 0/1
 - Cam pending 0/1 : elle attend ou bien elle fait une pose (contraire du idle)
 - Idle 0/1
 - Date de last maintenance
 - Failure 0/1

- **Dome & shutter**
 - Homing
 - slewing
 - Position (degrés)
 - Shutter : opened/closed/intermediate/unknown

- **PLC** (weather status, observatory status) :
 - valeur de chaque capteur (sensor) : rain.sensor1 rain.sensor2 wind.sensor3
 - Rain
 - Good weather
 - Wind : Clear, Cloudy, very cloudy

13.7.1.4. RUN

1 - Agent Majordome seul

```
(venv) $ cd src/majordome/  
(venv) $ ./start_agent_majordome.py
```

Le Majordome devrait afficher les messages suivants:
CURRENT OCS (MAJORDOME) STATE: STARTING
CURRENT OCS (MAJORDOME) STATE: PASSIVE_NO_PLC
Waiting for PLC connection...
Le majordome doit rester en attente de la connexion du PLC...

2 - Agent Majordome et les autres éléments nécessaires

Aller à la racine du projet
(venv) \$./pyros.py start_agents_and_simulators_for_majordome

Ce script lance l'agent Majordome, l'agent Env-Monitoring et le simulateur de PLC

Le Majordome devrait afficher les messages suivants:
CURRENT OCS (MAJORDOME) STATE: STARTING
CURRENT OCS (MAJORDOME) STATE: PASSIVE_NO_PLC
Waiting for PLC connection...
Puis, il devrait passer à l'état PASSIVE (car le simulateur de PLC est lancé)...

NB: Vous pouvez aussi lancer le serveur web (./pyros.py server) pour voir ce qui se passe, surtout via la page web System

13.7.1.5. TEST

A - Test automatique:

```
(venv) $ cd src/majordome/  
(venv) $ ./majordome_test.py
```

Ce test modifie certaines variables dans la BD pyros_test (notamment dans les tables config et plcdevicestatus) pour forcer le majordome à passer dans différents états

B - Test manuel (de visu):

1 - Dans un terminal, démarrer le serveur web pour voir le site pyros

Aller à la racine du projet et lancer le serveur web:

```
(venv) $ ./pyros.py server
```

Cliquer sur le menu "System" à gauche, pour aller sur la page de monitoring des agents.

Cette page vous permettra de suivre l'évolution des agents "majordome" et "environment-monitoring"

(<http://localhost:8000/dashboard/system>)

2 - Dans un autre terminal, démarrer tous les agents et simulateurs nécessaires pour que le majordome fonctionne

a - Avec le script ad hoc, c'est plus facile (il lance tout pour vous)

```
(venv) $ ./pyros.py start_agents_and_simulators_for_majordome
```

(ce script démarre les agents majordome et environment-monitoring, ainsi que le simulateur de plc)

Revenez sur la page web pour suivre l'évolution des agents "majordome" et "environment-monitoring"

Vous pouvez maintenant passer à l'étape 3 ci-dessous.

b - A la mano (vous démarrez vous-même chaque agent et simulateur, un par un)

Si vous n'aimez pas le script ad hoc ci-dessus et que vous préférez avoir un contrôle total de chaque élément, alors lancez vous même chaque agent et simulateur un par un:

Dans un nouveau terminal, démarrer l'agent majordome:

```
(venv) $ cd src/majordome/
```

```
(venv) $ ./start_agent_majordome.py
```

Il devrait afficher les messages suivants:

```
CURRENT OCS (MAJORDOME) STATE: STARTING
```

```
CURRENT OCS (MAJORDOME) STATE: PASSIVE_NO_PLC
```

```
Waiting for PLC connection...
```

Le majordome attend la connexion du PLC

Démarrer le simulateur de PLC et l'agent env-monitoring qui remplit la BD à partir des données du PLC, ce qui permet au majordome de savoir que le PLC est vivant...

- Démarrage simulateur PLC (dans un nouveau terminal) :
 - (venv) \$ cd simulators/plc/
 - (venv) \$./plcSimulator.py
- Démarrage agent env monitoring (dans un nouveau terminal) :
 - (venv) \$ cd src/monitoring/
 - (venv) \$./start_agent_monitoring.py

Le majordome voit la connexion avec le PLC et passe donc de l'état PASSIVE_NO_PLC à "PASSIVE" puis "Standby" (en passant par "closing")

Si on arrête (CTRL-C) le simulateur de PLC (et le env monitoring), le Majordome passe alors de nouveau à l'état PASSIVE puis PASSIVE_NO_PLC

3 - Suivez l'évolution du Majordome (ses différents "états")

Le majordome voit la connexion avec le PLC et passe donc de l'état PASSIVE_NO_PLC à "PASSIVE" puis "Standby" (en passant par "closing")

Maintenant, pour modifier l'état du Majordome, aller sur le site web, dans Préférences, puis Simulator

TODO: Activer certains boutons du Simulator pour influencer sur l'état du Majordome

Notamment, il faudrait un bouton qui permette de dire que la communication avec le PLC est OK ou KO pour que le Majordome passe à l'état "PASSIVE" (resp. "PASSIVE no PLC")

13.7.1.6. General algorithm

in src/majordome/tasks.py

C'est lui qui lance les agents **Monitoring** et **Alert Manager** (cf `majordome/tasks.py/Majordome(class)/handleTasks()`) car il voit qu'ils n'existent pas encore (ensuite, il les check régulièrement pour les relancer si arrêtés) *

```
⇒ src/majordome/tasks.py/class Majordome(Task) :
    ⇒ run() :
        createTask() ⇒ TaskId.objects.create(task_id=self.request.id,
            task="majordome")
        updateSoftware()
        setContext() :
            tel = TelescopeController()
            vis_camera = VISCameraController()
            nir_camera = NIRCcameraController()
            plc = PLCController()
            dom = DomeController()
        setTime() : # set timers and handlers (one handler per timer)
        setTasks():
            monitoring_task = TaskId.objects.get(task="monitoring")
            alert_task = TaskId.objects.get(task="alert_manager")
        loop() ⇒ LOOP (agent) :
            - check devices status
            - check if sequence is finished
            - check environment (from DB) (and take action, re-schedule)
            - check if new schedule available :
                - executeSchedule()
                    - executeSequence() :
                        - observation_manager.tasks.execute_plan_nir()
```

- observation_manager.tasks.execute_plan_vis()
- check if start of night ⇒ if so, launch a scheduling()
- check if end of night
- * check Monitoring and AlertManager (handleTasks()) :
 - monitoring.tasks.Monitoring.apply_async()
 - alert_manager.tasks.AlertListener.apply_async()

Détail de la méthode run() :

def run(self):

```

self.createTask()
self.updateSoftware()
self.setContext()
self.setTime()
self.setTasks()
self.loop()

```

def loop(self):

```

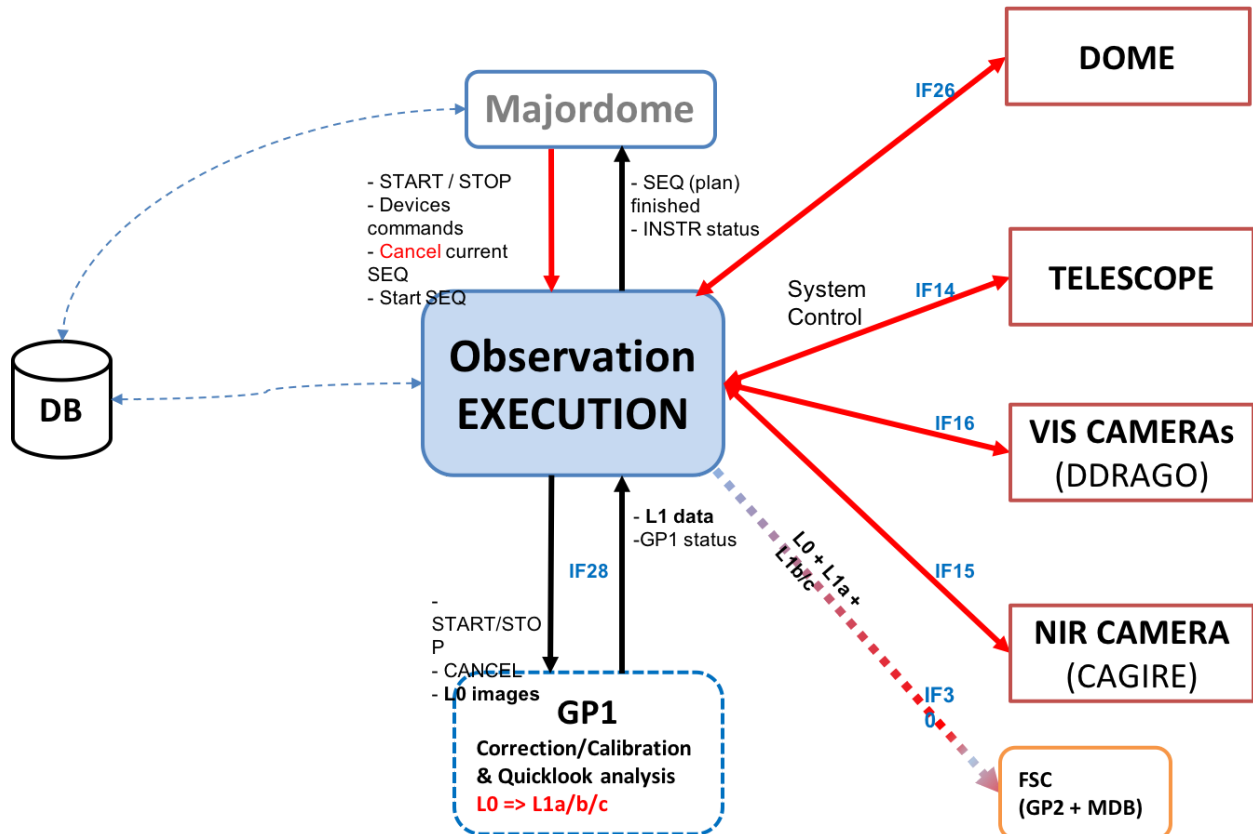
while (self.current_status != "SHUTDOWN"):
    minimal_timer = min(self.timers, key=self.timers.get)
    if (self.timers[minimal_timer] > 0):
        time.sleep(self.timers[minimal_timer])
        self.timers = {key: value - self.timers[minimal_timer] for key, value in self.timers.items()}
    for timer_name, timer_value in self.timers.items():
        if (timer_value <= 0):
            if timer_name in self.functions:
                self.logDB("Executing timer " + str(timer_name))
                self.functions[timer_name]()
            else:
                if (settings.DEBUG and DEBUG_FILE):
                    log.info("Timer : " + str(timer_name) + "is not known by the Majordome")
                    self.logDB("Timer " + str(timer_name) + " unknown")
        if (settings.DEBUG and DEBUG_FILE):
            log.info("Timer : " + str(timer_name) + " executed")

```

13.7.2. MODULE “Observer (EXEC)”

(updated 4/7/18)

13.7.2.1. Context diagram



13.7.2.2. Telescope monitoring agent

The telescope monitoring agent currently implemented is only a prototype.

Its current functionalities are the following:

When launched, the agent watch the DB looking for requests TelescopeCommand created by the server when a command is submitted on the web (remote mode)

If requests are found, it executes them using the TelescopeController.send_command() Method of the TelescopeController instantiated in the agent.

For now **IT DOES NOT USE** the RemoteControl classes which translate the generic commands into specific ones

When the requests are executed, the agent fill the answer field of the request in the DB

The requests are created in the views submit_command_to_telescope* (views.py)

And for the expert_mode, the view sleep for some milliseconds before sending an answer to the web to let the time to the agent to execute the request and fill up the db with the answer, answer which is sent by the view to the client as a response

13.8. Javascript files -> misc/static/js

13.9. Navbar -> misc/templates/base.html or base_unlogged.html

13.10. FUNCTION 5 - ENVIRONMENT monitoring (ENV)

(updated 22/6/18 - EP)

Responsible : Patrick Maeght

GFT-REQ-290: Environment monitoring shall take in charge the **following actions:**

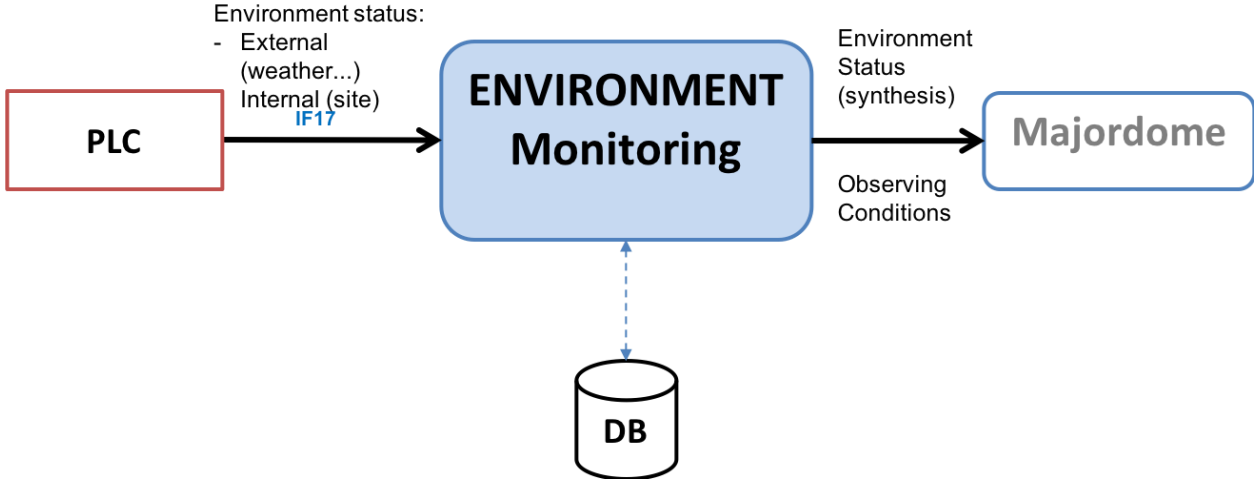
- Read outside environmental data (weather..., from the PLC) for instruments security
- Read inside environmental data (doors, lights..., from the PLC) for human safety
- Get PLC mode changes (off/manu/auto) and alarms (intrusion, e_stop)
- Correct raw data
- Compute and provide higher level (useful) parameters and synthesis from multiple detectors
- Save monitored data
- Keep a history of monitoring data
- Manage Observing conditions
- Show Weather & Observatory monitored data (in a convenient way)

Chaque capteur du PLC devra donner son time stamp

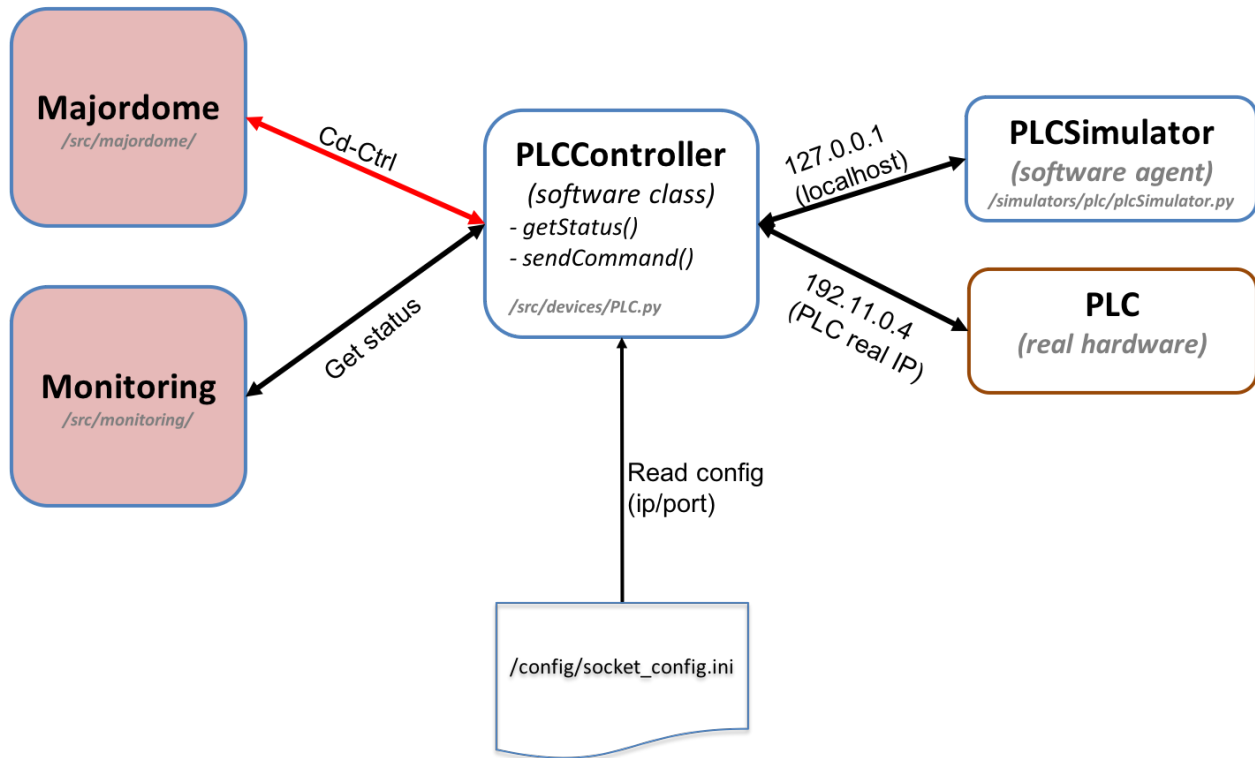
Redonder et prioriser les capteurs d'un même type de données (ex: pour l'humidité, on peut avoir 3 capteurs différents, dont un qui est le principal)

13.10.1. Contexte

The Environment Monitor module interfaces with other modules (inputs on the left)



Communication with devices (real or simulated)



13.10.2. PLC

13.10.2.1. Power management (onduleurs)

(FD):

Le PLC gère plusieurs niveaux de "warnings" pour les onduleurs :

- pas de pb
- passage sur batterie
- batterie faible, mise en protection.

Ensuite, en fonction de comment il communiquera avec l'onduleur, il sera possible d'avoir d'autres diagnostics

A priori, on ne gère pas le cas "tension plus faible qu'un seuil" (à définir, par ex: 107V au lieu de 110V nominal), cas qui n'est pas critique car l'onduleur est toujours en charge...

13.10.3. Fonctionnement du module

Principe général de fonctionnement :

⇒ A chaque itération, l'Agent Monitoring envoie une commande de status au PLC

⇒ Le PLC lui retourne son état actuel (status)

⇒ Le Monitoring en fait une synthèse qu'il met dans la BD

Fonctionnement détaillé :

(in src/monitoring/tasks.py)

*C'est lui qui lance les agents Majordome et Alert Manager (cf monitoring/tasks.py/Monitoring (class)/handleTasks()) car il voit qu'ils n'existent pas encore (ensuite, il les check régulièrement pour les relancer si arrêtés) **

⇒ **src/monitoring/tasks.py/class Monitoring(Task) :**

⇒ **run() :**

**createTask() ⇒ TaskId.objects.create(task_id=self.request.id,
task="monitoring")**

setContext() ⇒ plc = PLCController()

setTime()

setTasks() :

majordome_task = TaskId.objects.get(task="majordome")

alert_task = TaskId.objects.get(task="alert_manager")

loop() ⇒ LOOP (agent) :

Get PLC status :

handleTimerStatus() : status_plc = self.plc.getStatus() + SAVE in DB

**# * Check if the majordome and alert_manager are running (otherwise,
relaunch) :**

handleTasks() :

- **majordome.tasks.Majordome.apply_async()**

- **alert_manager.tasks.AlertListener.apply_async()**

Détail de la méthode run() :

def run(self):

self.createTask()

self.setContext()

self.setTime()

self.setTasks()

```
self.loop()
```

Détail de la BOUCLE :

```
def loop(self):
    while (self.state != "SHUTDOWN"):
        minimal_timer = min(self.timers, key=self.timers.get)
        time.sleep(self.timers[minimal_timer])
        self.timers = {key: value - self.timers[minimal_timer] for key, value in self.timers.items()}
        for timer_name, timer_value in self.timers.items():
            if (timer_value <= 0):
                if (timer_name in self.functions):
                    self.functions[timer_name]()
                else:
                    if (settings.DEBUG and DEBUG_FILE):
                        log.info("Timer : " + str(timer_name) + "is not known by the monitoring")
                        self.logDB("Timer " + str(timer_name) + " unknown")
                    if (settings.DEBUG and DEBUG_FILE):
                        log.info("Timer : " + str(timer_name) + " executed by monitoring")
```

13.10.4. EXECUTION

(updated 22/6/18 - EP)

Dans la phase de dev du module Monitoring (agent), **inutile de s'encombrer de Celery** pour tester ce module en isolation, donc **pas besoin non plus de démarrer RabbitMQ**.

On peut donc désactiver Celery ; dans 2 fichiers (pyros.py et src/pyros/settings.py), il faut s'assurer d'avoir ceci:

```
USE_CELERY = False
```

⇒ *Pour la version avec Celery, voir la section "EXECUTION AVEC CELERY" ci-dessous. Elle donne beaucoup plus de détail sur le déroulement de l'exécution.*

L'exécution se fait à partir de 2 ou 3 terminaux :

(1) - (Agent) Terminal 1 - Le serveur web (OPTIONNEL)

Optionnellement, on peut lancer le serveur web dans un 1er terminal, afin de mieux voir ce qui se passe au niveau de la météo et de l'observatoire :

```
(venv) $ ./pyros.py start_web
```

Ou encore :

```
(venv) $ cd src/
```

```
(venv) $ ./manage.py runserver
```

*Starting development server at http://127.0.0.1:8000/
(keep it running...)*

Puis se connecter sur <http://localhost:8000>

For your information, general syntax is :

```
$ ./manage.py runserver IP:PORT
```

```
Example: $ ./manage.py runserver localhost:8001
```

(obsolète: To check that this service is actually running, type "\$ netstat -an |grep 8000" and you should get "tcp 0 0 127.0.0.1:8000 0.0.0.0: LISTEN")*

On peut maintenant cliquer sur l'icone WEATHER pour voir les infos météo données par le PLC.

On peut aussi cliquer sur l'icone OBSERVATORY pour voir les infos observatoire données par le PLC.

Bien sûr, pour l'instant ces infos ne changent pas puisqu'on n'a pas lancé l'agent Env monitoring, ni le PLC.

(2) - (Agent) Terminal 2 - L'Agent "Environment Monitoring" (ENV)

Voici comment faire pour démarrer cet agent (depuis l'environnement virtuel), dans un 2ème terminal :

```
(venv) $ ./pyros.py start_agent_monitoring  
(Windows: python pyros.py start_agent_monitoring)
```

Ou encore :

```
(venv) $ cd src/monitoring/
```

```
(venv) $ ./start_agent_monitoring
```

```
(Windows: python start_agent_monitoring)
```

Ou bien encore, depuis le django shell :

```
(venv) $ cd src/
```

```
(venv) $ python manage.py shell
```

```
>>> from monitoring.tasks import Monitoring
```

```
>>> Monitoring().run()
```

```
>>> ...
```

```
Ou encore :  
>>> m = Monitoring()  
>>> m.run()  
>>> ...
```

Voilà, l'agent Env monitoring est lancé, il est en attente d'informations du PLC, mais il n'y a toujours pas de PLC, donc pas d'infos...

(3) - (Agent) Terminal 3 - Le simulateur de PLC

Dans un 3ème terminal (T3), activer l'environnement virtuel, puis :

```
(venv) $ cd simulators/plc/  
(venv) $ ./plcSimulator.py scenario_plc.json  
(Windows: python plcSimulator.py scenario_plc.json)
```

(pour info, scenario_plc.json est lu dans simulators/config/)

On peut aussi lancer le simulateur sans lui passer de scenario :

```
(venv) $ ./plcSimulator.py
```

Sans scénario, le simulateur du PLC donnera toujours la même réponse à celui qui l'interroge (l'agent Environment Monitoring). Avec un scénario, la réponse pourra varier.

Ca y est enfin, tout est prêt.

Si on a lancé le serveur web (étape 1 optionnelle), on peut maintenant cliquer sur l'icone WEATHER du dashboard pour voir évoluer les infos météo données par le PLC.

On peut aussi cliquer sur l'icone OBSERVATORY pour voir évoluer les infos observatoire données par le PLC.

13.10.4.1. Execution AVEC CELERY (version complète)

13.10.4.2. - (Agent) Terminal 0 - Un worker Celery dédié au Monitoring

Il attend des tâches à exécuter pour le Monitoring

A lancer dans un premier terminal (qu'on appellera **T0**)

⇒ **En fait, ce worker recevra seulement une tâche à exécuter : le run() de src/monitoring/tasks.py**

⇒ Voici comment faire :

Activer l'environnement virtuel, puis :

```
(venv) $ cd src/monitoring/  
(venv) $ ./start_celery_worker.py
```

NB1: cela est équivalent à exécuter cette commande :

```
$ celery worker -A pyros -Q monitoring_q -n pyros@monitoring -c 1
```

NB2: pour les curieux, voici le chemin parcouru par cet appel à celery :

```
site-packages/celery/worker/__init__.py  
site-packages/celery/bootsteps.py  
site-packages/celery/worker/consumer.py  
site-packages/celery/worker/loops.py  
site-packages/celery/apps/worker.py  
site-packages/celery/utils/dispatch/signal.py  
src/pyros/__init__.py
```

⇒ **Cela doit afficher le message suivant qui dit que le worker est en attente de tâche :**

```
[2018-02-19 11:07:04,023: WARNING/MainProcess] pyros@monitoring ready
```

⇒ *Sinon, si le message affiché est une erreur (en rouge), cela signifie que RabbitMQ n'est pas démarré :*

```
[2018-02-19 11:26:11,956: ERROR/MainProcess] consumer: Cannot connect to  
amqp://guest:**@127.0.0.1:5672//: [Errno 61] Connection refused.  
Trying again in 2.00 seconds...
```

Pour l'instant, **rien ne se passe, le worker est seulement en attente** de tâche à exécuter.

Cette instruction a seulement créé une **file d'attente** nommée **monitoring_q** et un **worker** qui lit cette "queue" en attendant quelque chose à faire, mais pour le moment il se tourne les pouces...

NB:

- Pour stopper ce worker, taper CTRL-C

- Si après avoir stoppé puis relancé ce worker, vous recevez toujours des messages de l'agent Monitoring (qui n'a donc pas été "tué" proprement), voici comment arrêter définitivement cette tâche :

```
(venv) $ ./stop_celery_worker.py
```

13.10.4.3. - (Agent) Terminal 1 - L'Agent "Monitoring"

(dans **src/monitoring/**, le fichier **tasks.py**, et plus précisément, sa méthode **run()**)

OK. On a un worker dédié au monitoring qui ne fait rien pour l'instant, à part attendre qu'on lui donne du boulot. Et bien, on va lui en donner du boulot ! On va lui donner 1 tâche à exécuter, qui sera notre agent Monitoring (ou ENV).

A lancer dans un deuxième terminal (qu'on appellera **T1**).

Voici comment faire pour démarrer cet agent dans le worker celery :

Dans un 2ème terminal donc (autre que celui qui exécute le worker ci-dessus), activer l'environnement virtuel, puis lancer le Django shell :

```
(venv) $ cd src/  
(venv) $ python manage.py shell  
>>> import monitoring.tasks  
>>> task_id = monitoring.tasks.Monitoring.dispatch()  
>>> task_id  
<AsyncResult: f225350c-e6c4-49b7-af26-d6b99fe9c596>
```

La tâche est lancée et on peut voir sur le 1er terminal (T0) les messages affichés par la tâche monitoring en cours (en fait, l'agent Monitoring, en boucle infinie) :

```
AGENT Monitoring: startup...  
FAILED TO CONNECT TO DEVICE PLC  
AGENT Monitoring: config PLC is (ip=127.0.0.1, port=5003)  
AGENT Monitoring: my timers (check env status every 2s, check other agents every 5s)  
AGENT Monitoring: Other Agents id read from DB (majordome=None, alert=None)  
  
AGENT Monitoring (ENV): iteration 0, (my state is RUNNING) :  
FAILED TO CONNECT TO DEVICE PLC  
Invalid PLC status returned (while reading) : NOT_SET  
Timer : timer_status executed by monitoring  
  
AGENT Monitoring (ENV): iteration 1, (my state is RUNNING) :
```

```

FAILED TO CONNECT TO DEVICE PLC
Invalid PLC status returned (while reading) : NOT_SET
Timer : timer_status executed by monitoring

AGENT Monitoring (ENV): iteration 2, (my state is RUNNING) :
TaskId matching query does not exist.
TaskId matching query does not exist.
Timer : timer_tasks executed by monitoring

AGENT Monitoring (ENV): iteration 3, (my state is RUNNING) :
...

```

L'id de la tâche Monitoring (task_id) est sauvegardé dans la table "task_id" (par la fonction createTask() de monitoring.tasks.run()).

Si vous avez un **phpmyadmin** installé*, vous pourrez voir une ligne de la table (base de données pyros) comme celle-ci :

id	task	created	task_id
11	monitoring	2018-02-19 14:52:15.640867	f225350c-e6c4-49b7-af26-d6b99fe9c596

**NB: Si vous n'avez pas de phpmyadmin, vous pouvez aussi utiliser la page administration de pyros : voir pour cela la section "[Accéder à la page d'administration de PyROS](#)"*

On peut constater dans les messages de celery (ci-dessus), la ligne suivante :

FAILED TO CONNECT TO DEVICE PLC

⇒ **C'est normal, car il n'y a pas de PLC pour l'instant !!!**

⇒ **Notre agent Monitoring passe son temps à interroger (à chaque itération) un device (le PLC) qui n'existe pas !!!**

⇒ **Ca n'est pas bloquant, le Monitoring continue de faire sa boucle infinie**

⇒ **Il va donc falloir lancer un simulateur de PLC à un moment ou un autre... (voir étape 3 ci-dessous).**

L'agent Monitoring contient un pointeur vers le **contrôleur du PLC** (qui s'appelle plcController)

La config du PLC (adresse IP et port) a été lue dans le fichier **/config/socket_config.ini**

Voici le contenu de ce fichier :

[Telescope]
ip=127.0.0.1
port=5000

[CameraVIS]
ip=127.0.0.1
port=5001

[CameraNIR]
ip=127.0.0.1
port=5002

[PLC]
ip=127.0.0.1
port=5003

[Dome]
ip=127.0.0.1
port=5004

On y voit que le PLC (en l'occurrence, le simulateur de PLC, voir étape suivante) écoute sur l'adresse localhost (127.0.0.1) et sur le port 5003

Le **contrôleur du PLC** est un **intermédiaire entre l'agent monitoring et le PLC**, qui permet de **dialoguer avec le PLC** (envoi de commande, et réception de la réponse).

Cette classe **PLCController** est définie dans **src/devices/PLC.py** (TODO: le dossier devices devrait plutôt s'appeler device_controllers, on fera le changement un jour...)

⇒ elle hérite de la classe **DeviceController** (définie dans src/devices/Device.py ; TODO: ce fichier devrait plutôt s'appeler DeviceController.py, on fera ça aussi un jour...)

⇒ elle est utilisée directement par l'agent Monitoring pour envoyer des commandes au PLC (**ce n'est pas un agent, c'est juste une classe**) et récupérer le résultat (listes de status)

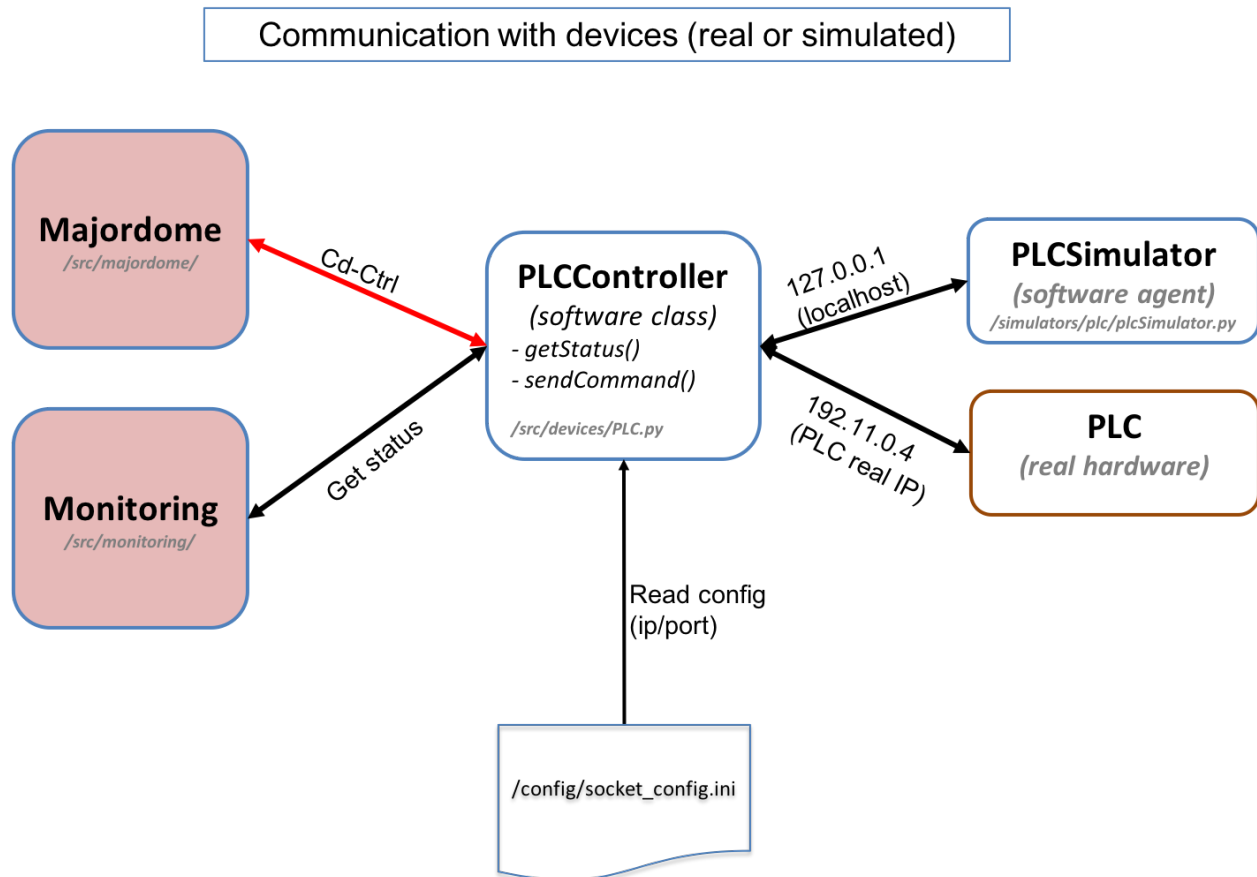
NB: On peut aussi regarder le contenu du **fichier LOG** (de Monitoring) qui contient quelques infos utiles sur ce qui se passe... : **/logs/Monitoring.log**

13.10.4.4. - (Agent) Terminal 2 - Le simulateur de PLC

Bon, récapitulons : on a un agent Monitoring qui tourne en boucle (lancé dans T1), et un worker celery dédié qui exécute cet agent (lancé dans T0)...

Mais il nous manque un PLC avec qui taper la causette ! En attendant le vrai PLC, on va utiliser un "simulateur" (très simplifié) dont la fonction sera seulement de répondre à nos questions.

Ce simulateur (PLCSimulator) **remplace donc le futur vrai PLC hardware, et sera donc capable de répondre à une requête envoyée par Monitoring (via un dictionnaire json).**



Il est défini dans `simulators/plc/plcSimulator.py`

Attention, le dossier "simulators/" est à la racine du projet et donc en dehors de Django (qui est dans src). C'est du python pur. Rien à voir avec django ou celery.

Le fichier `plcSimulator.py` définit la classe **PLCSimulator**.

Elle hérite des 2 classes `simulators/utils/device.py/DeviceSim` et `simulators/utils/StatusManager.py/StatusManager` :

- la **super-classe DeviceSim** sert à définir le comportement général d'un simulateur (comportement surchargé/adapté par les méthodes de `PLCSimulator`).

- la **super-classe StatusManager** sert à définir le comportement général d'un **générateur d'événements** à partir de la lecture d'un **fichier scénario** (json) qui lui dit quels sont les événements à générer et quand. Par exemple, le simulateur de PLC (PLCSimulator) pourra lire un fichier de scénario scenario_plc.json qui lui dit de "faire croire" qu'il pleut à partir de sa 3ème itération de boucle, et qu'il ne pleut plus à partir de sa 7ème itération, ou bien encore de "faire croire" qu'il est en panne (TODO:) pendant N itérations...

Il devra écouter sur l'adresse localhost (127.0.0.1) et sur le port 5003. Cette configuration du PLC (adresse IP et port) a été lue dans le fichier **/config/socket_config.ini** (voir étape 2) qui contient entre autres ces lignes :

```
[PLC]  
ip=127.0.0.1  
port=5003
```

Pour lancer TOUS les simulateurs, voir la fonction sims_launch() de pyros.py (\$ python **pyros.py sims_launch**). Mais ce n'est pas ce qu'on veut pour tester le Monitoring tout seul...

En fait lancer UNIQUEMENT le simulateur de PLC (et pas les autres). Pour cela il devrait suffire de faire (dans un processus ou un thread, lancé avec "subprocess.Popen()") quelque chose du genre :

```
sim = PLCSimulator(scenario_plc.json)  
sim.run()
```

Rappel : ce run() n'est pas exécuté dans Celery, ça n'a rien à voir, c'est juste du python, rien d'autre

(le fichier scenario_plc.json doit contenir les événements que le simulateur doit générer)

NB : ce simulateur PLC ne sera pas utilisé quand on aura le vrai PLC, mais il continuera toujours d'être utilisé dans les tests.

Voici comment faire :

Dans un 3ème terminal (T2), activer l'environnement virtuel, puis :

```
(venv) $ python plcSimulator.py scenario_plc.json
```

NB: plcSimulator.py peut être appelé avec ou sans argument. Sans argument (pas de fichier scenario), il le générera aucun événement, c'est à dire qu'il répondra toujours de la même façon à une même question posée par le Monitoring (alors qu'avec un scenario, la réponse pourra varier, ainsi que son comportement).

Si on regarde maintenant les messages reçus par le worker (terminal T0), on voit qu'il **n'affiche plus** le message :

```
FAILED TO CONNECT TO DEVICE PLC
```

Mais uniquement la ligne :

Timer : timer_status executed by monitoring
Cela montre que la connexion au PLC (c'est à dire au simulateur) se passe bien.

Voici en fait le résultat qu'on est censé obtenir avec le scénario scenario_plc.json :
Ce scénario contient les événements suivants (pour l'instant "time" signifie plutôt "numéro d'itération de la boucle du PLC" :

```
[
  10,
  {
    "time" : 3,
    "plcSimulator" : {"device_name":"WXT520", "value_name":"RainRate",
"value":12.0}
  },
  {
    "time" : 3,
    "plcSimulator" : {"device_name":"VantagePro",
"value_name":"RainRate", "value":12.0}
  },
  {
    "time" : 7,
    "plcSimulator" : {"device_name":"WXT520", "value_name":"RainRate",
"value":0.0}
  },
  {
    "time" : 7,
    "plcSimulator" : {"device_name":"VantagePro",
"value_name":"RainRate", "value":0.0}
  }
]
```

Comme on peut le deviner, à l'itération 3, les capteurs de pluie WXT520 et VantagePro devront indiquer un niveau de pluie de 12.0, puis un niveau 0.0 à l'itération 7.

Voyons si notre agent Monitoring (ENV) constate bien ces mêmes faits (voir les données en rouge ci-dessous). Sur T0, on obtient normalement les messages ci-dessous (au moment du lancement du simulateur sur T2, en admettant qu'il a été lancé un peu avant l'itération 19 du Monitoring) :

AGENT Monitoring (ENV): iteration 19, (my state is RUNNING) :

```
[2018-02-20 17:54:16,012: WARNING/Worker-1] Status received from PLC (read and parsed ok):
[2018-02-20 17:54:16,012: WARNING/Worker-1] [{"name": "STATUS", "from": "Beckhoff", "version_firmware":
"20170809", "site": "OSM-Mexico", "date": "2017-03-03T13:45:00", "device": [{"name": "WXT520", "type": "meteo",
"serial_number": "14656423", "valid": "yes", "values": [{"name": "OutsideTemp", "value": 12.12, "unit": "Celcius",
"comment": ""}, {"name": "OutsideHumidity", "value": 64.1, "unit": "percent", "comment": ""}, {"name": "Pressure",
"value": 769.2, "unit": "mbar", "comment": "At site elevation"}, {"name": "RainRate", "value": 0.0, "unit": "mm/h",
```

```
"comment": "", {"name": "WindSpeed", "value": 3.1, "unit": "m/s", "comment": ""}, {"name": "WindDir", "value": 202.3, "unit": "deg", "comment": "(N=0, E=90)"}, {"name": "DewPoint", "value": 8.3, "unit": "deg", "comment": ""}], {"name": "DRD11", "type": "meteo", "serial_number": "RET6789", "valid": "yes", "values": [{"name": "analog", "value": 2.345, "unit": "V", "comment": "3V=Dry <2.5V=Rain"}, {"name": "digital", "value": 1, "unit": "bool", "comment": "1=Dry 0=Rain"}], {"name": "VantagePro", "type": "meteo", "serial_number": "ERTRY2344324", "valid": "no", "values": [{"name": "OutsideTemp", "value": 12.45, "unit": "Celcius", "comment": ""}, {"name": "InsideTemp", "value": 20.28, "unit": "Celcius", "comment": ""}, {"name": "OutsideHumidity", "value": 65.3, "unit": "percent", "comment": ""}, {"name": "InsideHumidity", "value": 45.6, "unit": "percent", "comment": ""}, {"name": "Pressure", "value": 768.7, "unit": "mbar", "comment": "At site elevation"}, {"name": "RainRate", "value": 0.0, "unit": "mm/h", "comment": ""}, {"name": "WindSpeed", "value": 2.8, "unit": "m/s", "comment": ""}, {"name": "WindDir", "value": 207.0, "unit": "deg", "comment": "(N=0, E=90)"}, {"name": "WindDirCardinal", "value": "SW", "unit": "string", "comment": ""}, {"name": "DewPoint", "value": 8.5, "unit": "deg", "comment": ""}], {"name": "MLX90614-1", "type": "meteo", "serial_number": "Unknown", "valid": "yes", "values": [{"name": "SensorTemperature", "value": 23.56, "unit": "Celcius", "comment": ""}, {"name": "SkyTemperature", "value": -15.67, "unit": "Celcius", "comment": "Clear<-10 VeryCloudy>4"}], {"name": "LAMP_FLAT_CAGIRE", "type": "calib", "serial_number": "REF_3434", "valid": "yes", "values": [{"name": "status", "value": "on", "unit": "string", "comment": "on or off"}, {"name": "current", "value": 0.234, "unit": "Ampere", "comment": ""}], {"name": "LAMP FLAT CAGIRE", "type": "calib", "serial_number": "REF_3434", "valid": "yes", "values": [{"name": "status", "value": "on", "unit": "string", "comment": "on or off"}, {"name": "current", "value": 0.234, "unit": "Ampere", "comment": ""}]}}
```

Timer : **timer_status** executed by monitoring

AGENT Monitoring (ENV): iteration 20, (my state is RUNNING) :

TaskId matching query does not exist.

TaskId matching query does not exist.

Timer : **timer_tasks** executed by monitoring

AGENT Monitoring (ENV): iteration 21, (my state is RUNNING) :

[2018-02-20 17:54:18,049: WARNING/Worker-1] Status received from PLC (read and parsed ok):

```
[2018-02-20 17:54:18,049: WARNING/Worker-1] [{"name": "STATUS", "from": "Beckhoff", "version_firmware": "20170809", "site": "OSM-Mexico", "date": "2017-03-03T13:45:00", "device": [{"name": "WXT520", "type": "meteo", "serial_number": "14656423", "valid": "yes", "values": [{"name": "OutsideTemp", "value": 12.12, "unit": "Celcius", "comment": ""}, {"name": "OutsideHumidity", "value": 64.1, "unit": "percent", "comment": ""}, {"name": "Pressure", "value": 769.2, "unit": "mbar", "comment": "At site elevation"}, {"name": "RainRate", "value": 12.0, "unit": "mm/h", "comment": ""}, {"name": "WindSpeed", "value": 3.1, "unit": "m/s", "comment": ""}, {"name": "WindDir", "value": 202.3, "unit": "deg", "comment": "(N=0, E=90)"}, {"name": "DewPoint", "value": 8.3, "unit": "deg", "comment": ""}], {"name": "DRD11", "type": "meteo", "serial_number": "RET6789", "valid": "yes", "values": [{"name": "analog", "value": 2.345, "unit": "V", "comment": "3V=Dry <2.5V=Rain"}, {"name": "digital", "value": 1, "unit": "bool", "comment": "1=Dry 0=Rain"}], {"name": "VantagePro", "type": "meteo", "serial_number": "ERTRY2344324", "valid": "no", "values": [{"name": "OutsideTemp", "value": 12.45, "unit": "Celcius", "comment": ""}, {"name": "InsideTemp", "value": 20.28, "unit": "Celcius", "comment": ""}, {"name": "OutsideHumidity", "value": 65.3, "unit": "percent", "comment": ""}, {"name": "InsideHumidity", "value": 45.6, "unit": "percent", "comment": ""}, {"name": "Pressure", "value": 768.7, "unit": "mbar", "comment": "At site elevation"}, {"name": "RainRate", "value": 12.0, "unit": "mm/h", "comment": ""}, {"name": "WindSpeed", "value": 2.8, "unit": "m/s", "comment": ""}, {"name": "WindDir", "value": 207.0, "unit": "deg", "comment": "(N=0, E=90)"}, {"name": "WindDirCardinal", "value": "SW", "unit": "string", "comment": ""}, {"name": "DewPoint", "value": 8.5, "unit": "deg", "comment": ""}], {"name": "MLX90614-1", "type": "meteo", "serial_number": "Unknown", "valid": "yes", "values": [{"name": "SensorTemperature", "value": 23.56, "unit": "Celcius", "comment": ""}, {"name": "SkyTemperature", "value": -15.67, "unit": "Celcius", "comment": "Clear<-10 VeryCloudy>4"}], {"name": "LAMP_FLAT_CAGIRE", "type": "calib", "serial_number": "REF_3434", "valid": "yes", "values": [{"name": "status", "value": "on", "unit": "string", "comment": "on or off"}, {"name": "current", "value": 0.234, "unit": "Ampere", "comment": ""}], {"name": "LAMP FLAT CAGIRE", "type": "calib", "serial_number": "REF_3434", "valid": "yes", "values": [{"name": "status", "value": "on", "unit": "string", "comment": "on or off"}, {"name": "current", "value": 0.234, "unit": "Ampere", "comment": ""}]}}
```


"status", "value": "on", "unit": "string", "comment": "on or off"}, {"name": "current", "value": 0.234, "unit": "Ampere", "comment": ""}]}}]

Timer : **timer_status** executed by monitoring

AGENT Monitoring (ENV): iteration 22, (my state is RUNNING) :

[2018-02-20 17:54:18,049: WARNING/Worker-1] Status received from PLC (read and parsed ok):

[2018-02-20 17:54:18,049: WARNING/Worker-1] [{"name": "STATUS", "from": "Beckhoff", "version_firmware": "20170809", "site": "OSM-Mexico", "date": "2017-03-03T13:45:00", "device": [{"name": "WXT520", "type": "meteo", "serial_number": "14656423", "valid": "yes", "values": [{"name": "OutsideTemp", "value": 12.12, "unit": "Celcius", "comment": ""}, {"name": "OutsideHumidity", "value": 64.1, "unit": "percent", "comment": ""}, {"name": "Pressure", "value": 769.2, "unit": "mbar", "comment": "At site elevation"}, {"name": "RainRate", "value": 12.0, "unit": "mm/h", "comment": ""}, {"name": "WindSpeed", "value": 3.1, "unit": "m/s", "comment": ""}, {"name": "WindDir", "value": 202.3, "unit": "deg", "comment": "(N=0, E=90)"}, {"name": "DewPoint", "value": 8.3, "unit": "deg", "comment": ""}], {"name": "DRD11", "type": "meteo", "serial_number": "RET6789", "valid": "yes", "values": [{"name": "analog", "value": 2.345, "unit": "V", "comment": "3V=Dry <2.5V=Rain"}, {"name": "digital", "value": 1, "unit": "bool", "comment": "1=Dry 0=Rain"}]}, {"name": "VantagePro", "type": "meteo", "serial_number": "ERTRY2344324", "valid": "no", "values": [{"name": "OutsideTemp", "value": 12.45, "unit": "Celcius", "comment": ""}, {"name": "InsideTemp", "value": 20.28, "unit": "Celcius", "comment": ""}, {"name": "OutsideHumidity", "value": 65.3, "unit": "percent", "comment": ""}, {"name": "InsideHumidity", "value": 45.6, "unit": "percent", "comment": ""}, {"name": "Pressure", "value": 768.7, "unit": "mbar", "comment": "At site elevation"}, {"name": "RainRate", "value": 12.0, "unit": "mm/h", "comment": ""}, {"name": "WindSpeed", "value": 2.8, "unit": "m/s", "comment": ""}, {"name": "WindDir", "value": 207.0, "unit": "deg", "comment": "(N=0, E=90)"}, {"name": "WindDirCardinal", "value": "SW", "unit": "string", "comment": ""}, {"name": "DewPoint", "value": 8.5, "unit": "deg", "comment": ""}], {"name": "MLX90614-1", "type": "meteo", "serial_number": "Unknown", "valid": "yes", "values": [{"name": "SensorTemperature", "value": 23.56, "unit": "Celcius", "comment": ""}, {"name": "SkyTemperature", "value": -15.67, "unit": "Celcius", "comment": "Clear<-10 VeryCloudy>4"}]}, {"name": "LAMP_FLAT_CAGIRE", "type": "calib", "serial_number": "REF_3434", "valid": "yes", "values": [{"name": "status", "value": "on", "unit": "string", "comment": "on or off"}, {"name": "current", "value": 0.234, "unit": "Ampere", "comment": ""}]}, {"name": "LAMP FLAT CAGIRE", "type": "calib", "serial_number": "REF_3434", "valid": "yes", "values": [{"name": "status", "value": "on", "unit": "string", "comment": "on or off"}, {"name": "current", "value": 0.234, "unit": "Ampere", "comment": ""}]}}]

Timer : **timer_status** executed by monitoring

AGENT Monitoring (ENV): iteration 23, (my state is RUNNING) :

[2018-02-20 17:54:22,103: WARNING/Worker-1] Status received from PLC (read and parsed ok):

[2018-02-20 17:54:22,103: WARNING/Worker-1] [{"name": "STATUS", "from": "Beckhoff", "version_firmware": "20170809", "site": "OSM-Mexico", "date": "2017-03-03T13:45:00", "device": [{"name": "WXT520", "type": "meteo", "serial_number": "14656423", "valid": "yes", "values": [{"name": "OutsideTemp", "value": 12.12, "unit": "Celcius", "comment": ""}, {"name": "OutsideHumidity", "value": 64.1, "unit": "percent", "comment": ""}, {"name": "Pressure", "value": 769.2, "unit": "mbar", "comment": "At site elevation"}, {"name": "RainRate", "value": 0.0, "unit": "mm/h", "comment": ""}, {"name": "WindSpeed", "value": 3.1, "unit": "m/s", "comment": ""}, {"name": "WindDir", "value": 202.3, "unit": "deg", "comment": "(N=0, E=90)"}, {"name": "DewPoint", "value": 8.3, "unit": "deg", "comment": ""}], {"name": "DRD11", "type": "meteo", "serial_number": "RET6789", "valid": "yes", "values": [{"name": "analog", "value": 2.345, "unit": "V", "comment": "3V=Dry <2.5V=Rain"}, {"name": "digital", "value": 1, "unit": "bool", "comment": "1=Dry 0=Rain"}]}, {"name": "VantagePro", "type": "meteo", "serial_number": "ERTRY2344324", "valid": "no", "values": [{"name": "OutsideTemp", "value": 12.45, "unit": "Celcius", "comment": ""}, {"name": "InsideTemp", "value": 20.28, "unit": "Celcius", "comment": ""}, {"name": "OutsideHumidity", "value": 65.3, "unit": "percent", "comment": ""}, {"name": "InsideHumidity", "value": 45.6, "unit": "percent", "comment": ""}, {"name": "Pressure", "value": 768.7, "unit": "mbar", "comment": "At site elevation"}, {"name": "RainRate", "value": 0.0, "unit": "mm/h", "comment": ""}, {"name": "WindSpeed", "value": 2.8, "unit": "m/s", "comment": ""}, {"name": "WindDir", "value": 207.0, "unit": "deg", "comment": "(N=0, E=90)"}, {"name": "WindDirCardinal", "value": "SW", "unit": "string", "comment": ""}, {"name": "DewPoint",

```
"value": 8.5, "unit": "deg", "comment": ""}], {"name": "MLX90614-1", "type": "meteo", "serial_number": "Unknown",
"valid": "yes", "values": [{"name": "SensorTemperature", "value": 23.56, "unit": "Celcius", "comment": ""}, {"name":
"SkyTemperature", "value": -15.67, "unit": "Celcius", "comment": "Clear<-10 VeryCloudy>4"}]}, {"name":
"LAMP_FLAT_CAGIRE", "type": "calib", "serial_number": "REF_3434", "valid": "yes", "values": [{"name": "status",
"value": "on", "unit": "string", "comment": "on or off"}, {"name": "current", "value": 0.234, "unit": "Ampere", "comment":
""}], {"name": "LAMP FLAT CAGIRE", "type": "calib", "serial_number": "REF_3434", "valid": "yes", "values": [{"name":
"status", "value": "on", "unit": "string", "comment": "on or off"}, {"name": "current", "value": 0.234, "unit": "Ampere",
"comment": ""}]}}}]
```

Timer : **timer_status** executed by monitoring

TaskId matching query does not exist.

TaskId matching query does not exist.

Timer : **timer_tasks** executed by monitoring

AGENT Monitoring (ENV): iteration 24, (my state is RUNNING) :

[2018-02-20 17:54:22,103: WARNING/Worker-1] Status received from PLC (read and parsed ok):

```
[2018-02-20 17:54:22,103: WARNING/Worker-1] [{"name": "STATUS", "from": "Beckhoff", "version_firmware":
"20170809", "site": "OSM-Mexico", "date": "2017-03-03T13:45:00", "device": [{"name": "WXT520", "type": "meteo",
"serial_number": "14656423", "valid": "yes", "values": [{"name": "OutsideTemp", "value": 12.12, "unit": "Celcius",
"comment": ""}, {"name": "OutsideHumidity", "value": 64.1, "unit": "percent", "comment": ""}, {"name": "Pressure",
"value": 769.2, "unit": "mbar", "comment": "At site elevation"}, {"name": "RainRate", "value": 0.0, "unit": "mm/h",
"comment": ""}, {"name": "WindSpeed", "value": 3.1, "unit": "m/s", "comment": ""}, {"name": "WindDir", "value": 202.3,
"unit": "deg", "comment": "(N=0, E=90)"}, {"name": "DewPoint", "value": 8.3, "unit": "deg", "comment": ""}], {"name":
"DRD11", "type": "meteo", "serial_number": "RET6789", "valid": "yes", "values": [{"name": "analog", "value": 2.345,
"unit": "V", "comment": "3V=Dry <2.5V=Rain"}, {"name": "digital", "value": 1, "unit": "bool", "comment": "1=Dry
0=Rain"}]}, {"name": "VantagePro", "type": "meteo", "serial_number": "ERTRY2344324", "valid": "no", "values":
[{"name": "OutsideTemp", "value": 12.45, "unit": "Celcius", "comment": ""}, {"name": "InsideTemp", "value": 20.28,
"unit": "Celcius", "comment": ""}, {"name": "OutsideHumidity", "value": 65.3, "unit": "percent", "comment": ""}, {"name":
"InsideHumidity", "value": 45.6, "unit": "percent", "comment": ""}, {"name": "Pressure", "value": 768.7, "unit": "mbar",
"comment": "At site elevation"}, {"name": "RainRate", "value": 0.0, "unit": "mm/h", "comment": ""}, {"name":
"WindSpeed", "value": 2.8, "unit": "m/s", "comment": ""}, {"name": "WindDir", "value": 207.0, "unit": "deg", "comment":
"(N=0, E=90)"}, {"name": "WindDirCardinal", "value": "SW", "unit": "string", "comment": ""}, {"name": "DewPoint",
"value": 8.5, "unit": "deg", "comment": ""}], {"name": "MLX90614-1", "type": "meteo", "serial_number": "Unknown",
"valid": "yes", "values": [{"name": "SensorTemperature", "value": 23.56, "unit": "Celcius", "comment": ""}, {"name":
"SkyTemperature", "value": -15.67, "unit": "Celcius", "comment": "Clear<-10 VeryCloudy>4"}]}, {"name":
"LAMP_FLAT_CAGIRE", "type": "calib", "serial_number": "REF_3434", "valid": "yes", "values": [{"name": "status",
"value": "on", "unit": "string", "comment": "on or off"}, {"name": "current", "value": 0.234, "unit": "Ampere", "comment":
""}], {"name": "LAMP FLAT CAGIRE", "type": "calib", "serial_number": "REF_3434", "valid": "yes", "values": [{"name":
"status", "value": "on", "unit": "string", "comment": "on or off"}, {"name": "current", "value": 0.234, "unit": "Ampere",
"comment": ""}]}}}]
```

Timer : **timer_status** executed by monitoring

Monitoring (ENV): iteration 25, (my state is RUNNING) :

(ENV) Could not send message (on socket) to PLC : {"command": [{"name": "STATUS"}]} -> [Errno 32] Broken pipe

Invalid PLC status returned (while reading) : NOT_SET1

Timer : **timer_status** executed by monitoring

AGENT Monitoring (ENV): iteration 26, (my state is RUNNING) :

TaskId matching query does not exist.

TaskId matching query does not exist.

Timer : **timer_tasks** executed by monitoring

...

Bien sûr, le temps des itérations du Monitoring n'est pas le même que celui des itérations du PLC, donc elles ne se correspondent pas... Mais on constate bien le passage du niveau de pluie de 0 à 12, puis de nouveau à 0, sur les 2 capteurs WXT520 et VantagePro. IT WORKS !

Et maintenant, regardons ce qui s'est passé côté base de données. L'agent Monitoring met à jour 2 tables pour l'environnement :

- **weatherwatch**, pour la météo (environnement **externe** de l'observatoire)
- **sitewatch**, pour le site (environnement **interne** de l'observatoire)

Pour voir le contenu de ces tables, utilisez PhpMyAdmin (ou bien la page administration de pyros : voir pour cela la section "[Accéder à la page d'administration de PyROS](#)")

Voici le contenu de ces tables, après exécution du simulateur :

Table **weatherwatch** :

id	global_status	updated	humidity	wind	wind_dir	temperature	pressure	rain	cloud
249	OK	2018-02-21 15:32:17.118269	65.3	2.8	207.0	NULL	768.7	0	NULL
250	RAINING	2018-02-21 15:32:19.156377	65.3	2.8	207.0	NULL	768.7	12	NULL
251	RAINING	2018-02-21 15:32:21.183933	65.3	2.8	207.0	NULL	768.7	12	NULL
252	OK	2018-02-21 15:32:23.218375	65.3	2.8	207.0	NULL	768.7	0	NULL
253	OK	2018-02-21 15:32:25.245074	65.3	2.8	207.0	NULL	768.7	0	NULL

Table **sitewatch** :

id	global_status	updated	lights	dome	doors	temperature	shutter	pressure	humidity
249	OK	2018-02-21 15:32:17.119959	NULL	NULL		NULL	NULL	NULL	45.6
250	OK	2018-02-21 15:32:19.157529	NULL	NULL		NULL	NULL	NULL	45.6
251	OK	2018-02-21 15:32:21.185069	NULL	NULL		NULL	NULL	NULL	45.6

252	OK	2018-02-21 15:32:23.219474	NULL	NULL		NULL	NULL	NULL	45.6
253	OK	2018-02-21 15:32:25.246233	NULL	NULL		NULL	NULL	NULL	45.6

Autre méthode imaginable (à tester, ne marche pas pour l'instant...)

(venv) \$ python

```
>>> from simulators.plc.plcSimulator import PLCSimulator
```

```
>>> sim = PLCSimulator('config/conf.json')
```

```
>>> sim.run()
```

(mais pour l'instant, ça plante...)

13.10.5. DEVELOPMENT

Informations techniques nécessaires pour le dev :

Toute la BD est décrite dans le fichier `src/common/models.py`

Quand on clique sur les icones Weather et Observatory, ça déclenche les actions `weather` et `site` qui sont dans `src/dashboard/views.py`

Ces actions utilisent respectivement les vues `reload_weather.html` et `reload_site.html` qui sont dans `src/dashboard/templates/dashboard/`

Ces vues déclenchent respectivement les actions `views.py.weather_current` et `views.py.site_current` qui utilisent respectivement les vues `current_weather.html` et `current_site.html` (toujours dans `src/dashboard/templates/dashboard`)

13.11. FUNCTION 6 - DATA REDUCTION & ANALYSIS

13.11.1. Basic packages requirement.

Scientific package requirements for this module:

1. Astropy 3.0 : "[Installation](#)"
with other packages: numpy, h5py, BeautifulSoup, PyYAML, scipy, xmlilint, matplotlib, pytz, scikit-image, pandas, objgraph, setuptools, bleach, bintrees.
2. Scikit-Learn : "[Installation](#)".
3. Healpy : "[Installation](#)".
4. SExtractor "[Installation](#)" (if possible), need configuration files.
5. Astrometry.net "[Installation](#)" (if possible), need database.

6.3.2 Image Calibration

6.3.3 Image editor

6.3.4 Image analysis

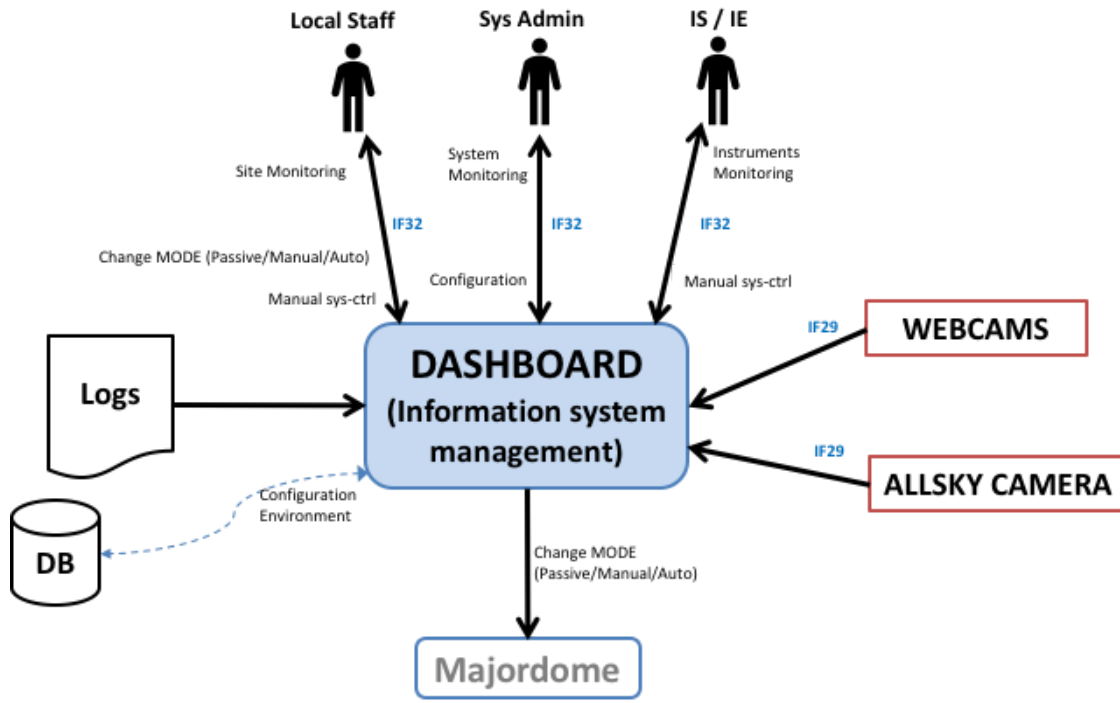
6.3.5 Source Extraction

6.3.5 Distortion correction

13.12. FUNCTION 7 - DASHBOARD (Information system management)

(updated 18/05/18)

Responsible : Théo Puhl



PYROS DASHBOARD

Monitoring

Weather



Observatory



Tele & Inst



Cameras



Science

Proposal



Request



Alerts



Schedule



Images



Fonctions du module :

GFT-REQ-299: the “Information System management” must take in charge the following actions:

- Manage users, profiles, users priority and quota
- Supply a short-term follow-up and a middle term of the observations
- Prepare and schedule correction & calibration sequences (flatfield, dark, bias, ...), and update calibration parameters
- Manage logs of the application
- Show the status of the different subsystems
- Allow to change the current MODE (manual/auto)
- Manual system control of the telescope (& instruments)
- Software & Hardware configuration
- Infrastructure management
- Backup and Restoration
- Webcams display

13.12.1. Users management

13.12.1.1. Profiles

N°	Profile	Rights
0	None (Visitor)	<ul style="list-style-type: none">+ Weather+ Observatory+ Webcams+ Alert+ Schedule
1	TAC	<ul style="list-style-type: none">+ Proposal
2	Observer (astronomer)	<ul style="list-style-type: none">+ Proposal (pour la période suivante)+ Request (associée à un SP)+ Images+ Profil perso (edit only "other email", "password")+ Tel & Inst (voir status only)
3	IE/IS	<ul style="list-style-type: none">+ Observatory control page (Change operating mode (REMOTE/SCHEDULER))+ Tele & Inst (ctrl cde)
4	Operator (LOCAL)	<ul style="list-style-type: none">+ Give starting night ACK, PLC bypass-authorize/forbid in Observatory control page- Request- Proposal- Images
5	Superoperator (AK)	<ul style="list-style-type: none">+ Config
6	PI	<ul style="list-style-type: none">+ User rights management+ Proposal (+ prio + quota) <p><i>(Prévoir protections quand même !!!)</i></p>
7	SysAdmin	Super user rights (sans file)

Detailed rights :

PROFILE / RIGHT	0	1	2	3	4	5	6	7
Weather	Green	Green	Green	Green	Green	Green	Green	Green
- Log	Green	Green	Green	Green	Green	Green	Green	Green
Observatory (status)	Green	Green	Green	Green	Green	Green	Green	Green
- Log	Green	Green	Green	Green	Green	Green	Green	Green
Observatory (Control)	Black	Black	Black	Green	Green	Green	Green	Green
- Give night ACK	Black	Black	Black	Black	Blue	Black	Black	Black
- Bypass PLC (SAFE/UNSAFE)	Black	Black	Black	Black	Blue	Black	Black	Black
- Change Operating Mode (PASSIVE/REMOTE/SCHEDULER)	Black	Black	Black	Blue	Blue	Blue	Blue	Blue
Tel & Inst	Black	Black	Green	Green	Green	Green	Green	Green
- Status	Black	Black	Green	Green	Green	Green	Green	Green
- Control Command	Black	Black	Black	Blue	Blue	Blue	Blue	Blue
Webcams	Green	Green	Green	Green	Green	Green	Green	Green
- Log	Green	Green	Green	Green	Green	Green	Green	Green
Proposals	Black	Green	Green	Green	Black	Green	Green	Green
- See all proposals	Black	Green	Green	Green	Black	Green	Green	Green
- Create	Black	Black	Blue	Blue	Black	Blue	Blue	Blue
- Update	Black	Black	Blue	Blue	Black	Blue	Blue	Blue
- View Vote	Black	Green	Green	Green	Black	Green	Green	Green
- Vote	Black	Blue	Black	Black	Black	Black	Black	Black
- Set Quota & Prio	Black	Black	Black	Black	Black	Black	Blue	Blue
- Delete (personal before SP)	Black	Black	Blue	Blue	Black	Blue	Black	Black
- Delete (all before SP)	Black	Black	Black	Black	Black	Black	Blue	Blue
Request	Black	Black	Green	Green	Black	Green	Green	Green

- Create									
- Personal SP									
- All request									
- Update									
- Delete (personal one, only if not submitted)									
- Delete ALL (warning, attention)									
Alerts									
- List of alerts									
Schedule									
- Planning									
Images									
- Personal images									
- All images									
User profile									
- Personal info									
- Edit (only "other email", "password")									
Users Management									
- All users info									
- Edit (all users, all fields)									
- Deactivate									
Config									
- Edit Config									

Black block - Hidden data for the user
 Green block - Informative data for the user
 Blue block - Interaction with the data for the user

13.12.1.2. Scenario (workflow)

Version 1

1) un nouvel utilisateur s'enregistre via un registration form ("sign in"), dans lequel il décrit ses intentions

2) il reçoit un mail l'informant que son compte a bien été créé mais qu'il est en attente de validation par le PI Colibri

3) the Colibri PI reçoit un mail contenant un lien qui permet d'activer ce nouveau compte (ou pas)

NB: pour l'instant, le mail contient seulement un lien vers la page web qui permet d'activer le compte du nouvel utilisateur (en cliquant sur un bouton "Activate") et il faudra que le PI soit connecté pour y accéder... on fera mieux plus tard...

4) le nouvel utilisateur reçoit alors aussi un mail l'informant que son compte a été activé

5) il peut alors se connecter au site et faire les actions suivantes :

- soit déposer un new proposal pour la prochaine période de 6 mois

- soit déposer une requete d'observation pour un SP (Scientific Program) de la période de 6 mois en cours, SP auquel il est déjà associé (car il l'avait soumis à la période précédente)

A tout moment, le PI peut décider de désactiver un utilisateur (il pourra toujours le réactiver si besoin). L'utilisateur concerné recevra alors un email pour l'en informer.

Un utilisateur pourra soumettre PLUSIEURS proposals et donc il pourra plus tard soumettre des requetes d'observation sur plusieurs SP. Au moment de soumettre une requete d'observation, il doit sélectionner le SP concerné dans une liste déroulante ; si cette liste est vide, il ne pourra tout simplement pas soumettre de requete.

Dans cette version, il y a un **compte UNIQUE pour un SP** (prog scientifique). Les différentes personnes contribuant à ce SP utiliseront donc le MEME compte et s'arrangeront entre elles pour gérer leur quota...

Version 2

- la personne qui dépose un proposal (futur SP) en devient automatiquement le PI (on dira le **SP-PI**)

- les autres utilisateurs peuvent demander à être associé à un ou plusieurs SP(s) par demande directe au SP-PI (email).

- le SP-PI doit donc avoir accès à une liste de TOUS les utilisateurs pour sélectionner ceux qui sont demandeurs pour son SP ou dé-sélectionner ceux qui ne le sont plus...

- les utilisateurs sélectionnés (ou dé-sélectionnés) reçoivent alors un email pour les en informer

- Un utilisateur pourra soumettre (**ou être associé à**) PLUSIEURS proposals.

13.12.2. Observatory Control page

(updated 4/7/18)

Le ACK est donné une fois pour toutes en début de nuit (il n'est pas enlevé ensuite, sauf automatiquement après la fin de la nuit)

Attention, ce ACK n'est pas donné via la page web, mais via un bouton physique dans la control room ; on reçoit donc l'info via le PLC

Voici les BOUTONS d'action qui seront disponibles sur la page web "OBSERVATORY CONTROL PAGE" :

(tous ces boutons seront bien sûr soumis à une confirmation, of course !)

Bouton 1 - "PLC_BYPASS (set to SAFE)" ou "STOP PLC_BYPASS"

=> affiché ssi UNSAFE

- "PLC_BYPASS" fait passer à SAFE.

- "STOP PLC_BYPASS" permettra de mettre fin au bypass (le CC tiendra donc à nouveau compte du PLC status, qu'il soit SAFE ou UNSAFE)

Remarque : attention, ça ne marche que si UNSAFE (pas si SAFE), comme demandé par François ; dans le cas SAFE, on utilise plutôt le bouton 2

Bouton 2 - "LOCK OBSERVATORY" ou "UNLOCK OBSERVATORY"

==> affiché ssi SAFE

- "LOCK OBSERVATORY" fait passer à STANDBY (après close dome...) et y reste bloqué (LOCKED = true)

- "UNLOCK OBSERVATORY" permettra de repasser la variable LOCKED à False ce qui permettra de sortir du mode STANDBY dans lequel on était bloqué

Remarques:

- attention, ces boutons ne changent pas l'état SAFE)

- pour moi, ces boutons doivent pouvoir aussi être activables depuis le mode REMOTE-COMMAND

Bouton 3 - "GO REMOTE-COMMAND MODE" ou "GO SCHEDULER MODE"

Ces boutons ne sont activables QUE si la variable LOCKED est False (voir bouton 2)

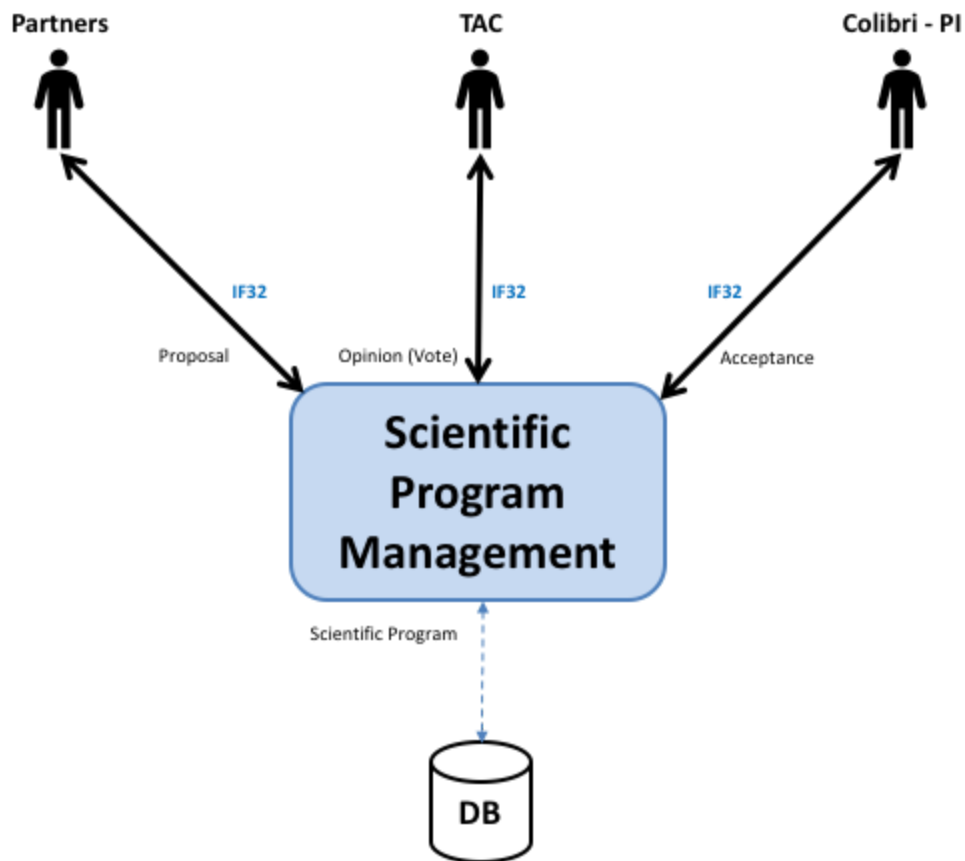
Dans le détail:

Bouton	Visible ssi	Action
<p>“BYPASS_PLC (set to SAFE)”</p> <p>OU</p> <p>“STOP BYPASS_PLC”</p>	<p>UNSAFE & OPERATOR</p> <p>SAFE & OPERATOR</p>	<p>Bypass le mode UNSAFE du PLC en SAFE. Désormais CC ne tient plus compte du status du PLC...</p> <p>Annule le bypass. CC tient à nouveau compte du PLC status.</p> <p>Ces 2 actions sont transmises au PLC pour qu’il le prenne en compte (bypass=1 ou 0). CC doit vérifier ensuite qu’il reçoit bien cette information du PLC, ce qui prouve que le PLC l’a bien prise en compte (sinon, au bout d’un timeout, “PLC KO” error)</p>
<p>(staff LOCK/UNLOCK)</p> <p>“LOCK OBSERVATORY (CLOSE)”</p> <p>OU</p> <p>“UNLOCK OBSERVATORY (OPEN)”</p> <p>UNLOCK ne fait que supprimer le "LOCK")</p>	<p>OPERATOR & SAFE</p> <p>⇒ ssi mode SCHED-READY (ou SCHED-STANDBY ?) Ou mode REMOTE</p> <p>⇒ ssi mode SCHED-STANDBY et STOPPED is TRUE</p> <p>(le mode SCHEDULER doit pouvoir se réactiver tout seul automatiquement après chaque passage à SAFE par le PLC, si LOCKED est à false, ce qui est le cas par défaut)</p>	<p>⇒ passer variable LOCKED à TRUE, et revient au mode STANDBY (pour y rester bloqué) après être passé par SCHEDULER-CLOSING (ou pas si c’était REMOTE)</p> <p>⇒ supprime le LOCK ; repasse la variable LOCKED à FALSE (ce qui devrait faire passer automatiquement au mode SCHEDULER-READY) si toutes les conditions sont favorables)</p>

<p>“GO REMOTE MODE”</p> <p>OR</p>	<p>>= IE/IS</p> <p>⇒ ssi UNLOCKED ET mode SCHED-STANDBY ou SCHED-READY (peu importe qu’on soit en SAFE ou UNSAFE)</p>	<p>⇒ passe en mode REMOTE-COMMAND</p>
<p>“GO SCHEDULER MODE”</p>	<p>⇒ ssi UNLOCKED & SAFE ET mode REMOTE</p>	<p>⇒ passe en mode SCHEDULER-STANDBY (qui passera automatiquement à READY)</p>

Question en suspens : quid du mode **REMOTE** ? doit-on autoriser l'action "STOP OBSERVING" depuis ce mode pour forcer la fermeture du dome par exemple ?

13.13. FUNCTION 8 - (Proposals) Scientific Programs Management



13.14. FUNCTION 9 - Telescope & Instruments long term Monitoring and Calibration

Cette fonction est assurée par le GIC (@CPPM)

13.15. FUNCTION 10 - Data Archiving

14. ANNEXES

14.1. Annex 1 - Récupération BD après mauvaise manip (ou bien si "update" ne fonctionne pas)

(updated 19/3/19 - EP)

Exemple : on part de la situation où la table "command" a été supprimée manuellement.

Il faut donc ré-appliquer les migrations à partir du fichier de migration qui crée cette table.

Pour trouver ce fichier :

```
$ grep Command src/common/migrations/*.py
```

(attention "Command" avec C majuscule)

Une des lignes affichées par grep est :

```
name='Command',
```

C'est cette ligne qui indique le bon fichier de migration.

Il faut donc maintenant faire croire (fake) à Django que les migrations n'ont été appliquées que JUSQU'A ce fichier là.

Donc si ce fichier commence par 0013 (par exemple), **activer mon environnement virtuel**, puis faire ceci :

```
$ cd src/  
$ python manage.py migrate common 0012 --fake
```

Il ne reste plus maintenant qu'à ré-appliquer toutes les migrations suivantes (c'est à dire ici à partir de la 0013)

```
$ python manage.py migrate common
```

=> ce qui devrait re-crée la table "command" (entre autres...)

Tout devrait maintenant être rentré dans l'ordre

Pour en être bien sûr, désactiver mon environnement virtuel, et tester que je peux faire un update sans problème :

```
$ deactivate  
$ cd ../          (=> retour à PYROS/)  
$ python pyros.py update
```

Maintenant, exécuter pyros et voir si tout fonctionne bien.

14.2. Annex 2 - Git

14.2.1. Commit procedure

Avant de faire un “commit & push” de code, voici la procédure à respecter :

- Mettre à jour le fichier **README du module** modifié (/src/module/README) : version, date, auteur, ... (chaque module doit être “pensé” comme une application indépendante)
- Mettre à jour le fichier **README général** du projet (/README) : version, date, auteur, ...
- S’assurer que le **style de codage** est bien respecté (cf [CODING STYLE](#))
- S’assurer que tous les **tests** passent toujours (cf [TEST](#)) :
 - `$./pyros.py test`
- **Mettre à jour votre code** (quelqu’un pourrait avoir fait des modifs avant vous !) :
 - Vérifier que vous êtes bien sur la branche "dev" :

```
$ cd PYROS/  
$ git branch  
* dev  
  master
```
 - (Si ce n’est pas déjà le cas, aller sur la branche dev) :

```
$ git checkout dev
```
 - Mettre à jour votre copie :

```
$ git pull
```

Maintenant, vous êtes prêts pour envoyer vos changements :

- **Faire le point sur la situation** : `$ git status`

- **Ajouter** les fichiers à commiter:
 - tous vos changements... : `$ git add *`
 - ... ou bien seulement certains fichiers :
`$ git add file1 file2 file3...`
`$ git status`

- **Commiter ces changements localement** (sur votre disque) :
`$ git commit -m "message de commit qui explique bien ce que vous avez fait"`
`$ git status`

- **Pousser ces changements vers le dépôt gitlab** pour que tout le monde y ait accès :
`$ git push`
`$ git status`

14.2.2. Pull procedure

\$ git pull

En cas de conflit sur le "git pull", voir comment le résoudre [dans le chapitre "updating pyros"](#)

14.3. Annex 3 - PyROS general python syntax rules and coding style

(updated 14/11/18)

Ce chapitre n'est qu'un résumé de ce qui est vraiment important. Il est encore incomplet et sera enrichi progressivement. Pour tout ce qui n'est pas (encore) dit, respecter "au maximum" les conventions de la PEP08 : <https://www.python.org/dev/peps/pep-0008/>

- **GENERAL RULES**

Ces règles générales sont valables quelque soit le langage utilisé (Python, Php, Java, ...)

- **KISS** (Keep It Stupid Simple, https://fr.wikipedia.org/wiki/Principe_KISS) : vous-mêmes ou à plus forte raison quelqu'un d'autre, doit pouvoir relire votre code plusieurs années après et le comprendre rapidement
- **DRY** (Don't repeat yourself)
- **Use exceptions** rather than returning and checking for error states
- **Command/Query Separation** : **Commands** return void and **Queries** return values (cf <https://hackernoon.com/oo-tricks-the-art-of-command-query-separation-9343e50a3de0>) ; en d'autres termes : "Functions that change state should not return values and functions that return values should not change state"
Ex (en langage C):

```
int m(); // query  
void n(); // command
```
- **Law of Demeter** (cf <https://hackernoon.com/object-oriented-tricks-2-law-of-demeter-4ecc9becad85>) **LoD** tells us that it is a bad idea for single functions to know the entire navigation structure of the system. "Each unit should have only limited knowledge about other units: only units "closely" related to the current unit. Each unit should only talk to its friends; don't talk to strangers."
- **Guideline/Coding standard for Django** : <https://medium.com/@harishoraon/guide-line-for-django-application-e1a1c075aed>

- **TEMPLATE DE FONCTION OU METHODE**

(cf <https://docs.python.org/3/library/typing.html>)

```
from typing import Any, TypeVar, Iterable, Tuple, Union
```

```
def is_connected_to_server(
    item: Any,
    vector: List[float],
    words: Dict[str, int],
    word: Union[int, str],
    server_host: str="localhost",
    server_port: int=11110,
    buffer_size: int=1024,
    DEBUG: bool=False,
) -> bool=False:
    """
    Return True if we are connected to the server (False by default)

    :param server_host: server IP or hostname
    :param server_port: port used by the server
    :param buffer_size: size of the buffer in bytes
    :param DEBUG: if true, will print and log more messages
    ...etc...
    """
```

- **INDENTATION**

4 espaces (régler la tabulation de votre éditeur pour qu'elles soient remplacées par 4 espaces)

- **CLASSES**

Dans la mesure du possible, **1 classe = 1 fichier** du même nom

Ex: vehicule.py ne devrait contenir QUE la classe Vehicule

Class names should normally use the **CapWords (CamelCase)** convention

Instances of class should be **snake_case**

Ex:

```
# class
class MyClass:
    ...
```



```
# class instance :  
my_class_instance = MyClass(...)
```

- **FUNCTION and VARIABLE names**

Should be **snake_case** (lowercase, with words separated by underscores) as necessary to improve readability

```
Ex:  
def my_nice_function():  
    ...  
    ...  
  
my_nice_variable = 3
```

- **ÉLÉMENTS MULTIPLES (list, tuple, set ou dict)**

Doivent être au **PLURIEL**:

- my_items
- items
- instances
- keys
- values
- ...

- **TODO**

Utiliser le tag "**# TODO:**" en début de ligne pour marquer une action à faire (Attention: bien mettre les 2 points à la fin)

- **CONSTANTES**

```
LIMIT_MAX  
LIMIT_MIN  
STATICFILES_DIRS
```

- **LINE SIZE**

Dans la mesure du possible, ne pas dépasser les 90 caractères

- **FUNCTION SIZE**

Une méthode ou fonction ne doit faire qu'une seule chose, et doit être la plus concise possible, et en tous cas doit **tenir en entier sur l'écran** pour qu'on comprenne rapidement ce qu'elle fait. De manière générale, **rester en dessous des 30 lignes**.

- **COMMENTAIRES**

Pour chaque méthode ou fonction, mettre un commentaire juste en-dessous (triple quote) :

```
def my_method():  
    """ Commentaire sur une seule ligne """  
    ...
```

ou bien

```
def my_method():  
    """  
    Commentaire sur plusieurs lignes  
    Deuxième ligne  
    Troisième ligne  
    """  
    ...
```

- **COMMENTAIRE TODO ou bien désactivation d'une ligne de code => #**

```
# TODO: bla bla bla  
#my_str = readline()
```

- **MÉTHODES BOOLÉENNES (true/false) bien lisibles**

S'assurer de la lisibilité du code en employant des **noms de méthodes en "anglais courant"**.

Exemples:

```
is_writeable()  
is_readable()  
has_components()  
makes_noise() # => ATTENTION, ne pas confondre avec "make_noise()" qui sera  
plutôt une méthode "fabriquant du bruit" (et non pas retournant un booléen)  
does_noise() # => idem, ne pas confondre avec "do_noise()"
```

- **VISIBILITY Private / Public**

- Mettre les fonctions **publiques** AU DEBUT du code
- Mettre les fonctions **privées** A LA FIN du code
- **Attributs et Méthodes : PRIVÉS par défaut !**
 - **Attributs** : les encapsuler en les rendant accessibles et modifiables uniquement par des accesseurs (**getters et setters**, mais c'est encore mieux d'utiliser les "**@property**" ⇒ cf <https://www.programiz.com/python-programming/property>)
 - **Méthodes** : seules les méthodes vraiment utilisées par les autres classes doivent être publiques.

⇒ Pour privatiser un attribut ou une méthode, les **préfixer par un underscore**

Exemple :

```
_my_private_attribute  
_my_private_method()
```

- **SIGNATURE DES MÉTHODES**

Utiliser les "**type hints**" pour donner le type de tous les paramètres d'une méthode, ainsi que le type de retour (Intérêt : Eclipse, et surtout PyCharm signalent une erreur lorsqu'un paramètre est passé avec un mauvais type)

Ex:

```
def reverse_slice(text:str, start:int=3, end:int) -> str:  
    return text[start:end][::-1]
```

14.4. Annex 4 - PyROS objects naming rules (syntax for the models.py file)

See file `src/common/models.py` for a more up to date reference.

This section describes the naming rules and conventions to be used for the pyros objects, mainly corresponding to the database tables names and fields in the pyros database. These tables are described in and managed from the file `src/common/models.py`.

- Model name => singular

Call it `Company` instead of `Companies`.

A model definition is the representation of a single object (the object in this example is a company),

and not a collection of companies

The model definition is a class, so always use CapWords convention (no underscores)

E.g. `User`, `Permission`, `ContentType`, etc.

- For the model's attributes use `snake_case`.

E.g. `first_name`, `last_name`, etc

- Blank and Null Fields

(<https://simpleisbetterthancomplex.com/tips/2016/07/25/django-tip-8-blank-or-null.html>)

- Null: It is database-related. Defines if a given database column will accept null values or not.

- Blank: It is validation-related. It will be used during forms validation, when calling `form.is_valid()`.

Do not use `null=True` for text-based fields that are optional.

Otherwise, you will end up having two possible values for "no data", that is: `None` and an empty string.

Having two possible values for "no data" is redundant.

The Django convention is to use the empty string, not `NULL`.

Example:

```
# The default values of `null` and `blank` are `False`.
```

```
class Person(models.Model):
```

```
    name = models.CharField(max_length=255) # Mandatory
```

```
    bio = models.TextField(max_length=500, blank=True) # Optional (don't put null=True)
```

```
    birth_date = models.DateField(null=True, blank=True) # Optional (here you may add null=True)
```

The default values of `null` and `blank` are `False`.

Special case, when you need to accept `NULL` values for a `BooleanField`, use `NullBooleanField` instead.

- Choices : you can use `Choices` from the `model_utils` library. Take model `Article`, for instance:

```
from model_utils import Choices
```

```
class Article(models.Model):
```

```

STATUSES = Choices(
(0, 'draft', _('draft')),
(1, 'published', _('published')) )
status = models.IntegerField(choices=STATUSES, default=STATUSES.draft)

```

- Reverse Relationships

- related_name :

Rule of thumb: if you are not sure what would be the related_name, use the plural of the model holding the ForeignKey.

ex:

```

class Company:
name = models.CharField(max_length=30)
class Employee:
first_name = models.CharField(max_length=30)
last_name = models.CharField(max_length=30)
company = models.ForeignKey(Company, on_delete=models.CASCADE,
related_name='employees')

```

usage:

```

google = Company.objects.get(name='Google')
google.employees.all()

```

You can also use the reverse relationship to modify the company field on the Employee instances:

```

vitor = Employee.objects.get(first_name='Vitor')
google = Company.objects.get(name='Google')
google.employees.add(vitor)

```

- related_query_name :

This kind of relationship also applies to query filters.

For example, if I wanted to list all companies that employs people named 'Vitor', I could do the following:

```

companies = Company.objects.filter(employee__first_name='Vitor')

```

If you want to customize the name of this relationship, here is how we do it:

```

class Employee:
    first_name = models.CharField(max_length=30)
    last_name = models.CharField(max_length=30)
    company = models.ForeignKey(
        Company,
        on_delete=models.CASCADE,
        related_name='employees',
        related_query_name='person'
    )

```

Then the usage would be:

```

companies = Company.objects.filter(person__first_name='Vitor')

```

To use it consistently, related_name goes as plural and related_query_name goes as singular.

GENERAL EXAMPLE

=====

```
from django.db import models
from django.urls import reverse

class Company(models.Model):
    # CHOICES
    PUBLIC_LIMITED_COMPANY = 'PLC'
    PRIVATE_COMPANY_LIMITED = 'LTD'
    LIMITED_LIABILITY_PARTNERSHIP = 'LLP'
    COMPANY_TYPE_CHOICES = (
        (PUBLIC_LIMITED_COMPANY, 'Public limited company'),
        (PRIVATE_COMPANY_LIMITED, 'Private company limited by shares'),
        (LIMITED_LIABILITY_PARTNERSHIP, 'Limited liability partnership'),
    )

    # DATABASE FIELDS
    name = models.CharField('name', max_length=30)
    vat_identification_number = models.CharField('VAT', max_length=20)
    company_type = models.CharField('type', max_length=3,
    choices=COMPANY_TYPE_CHOICES)

    # MANAGERS
    objects = models.Manager()
    limited_companies = LimitedCompanyManager()

    # META CLASS
    class Meta:
        verbose_name = 'company'
        verbose_name_plural = 'companies'

    # TO STRING METHOD
    def __str__(self):
        return self.name

    # SAVE METHOD
    def save(self, *args, **kwargs):
        do_something()
        super().save(*args, **kwargs) # Call the "real" save() method.
        do_something_else()

    # ABSOLUTE URL METHOD
    def get_absolute_url(self):
        return reverse('company_details', kwargs={'pk': self.id})

    # OTHER METHODS
```

```
def process_invoices(self):  
    do_something()
```

14.5. Annex 5 - TESTS rules (integrated tests of the PyROS code)

(Pour les tests unitaires basiques "in situ" ⇒ utiliser **doctest**)

14.5.1. Utilité des tests

Ceinture de sécurité pour s'assurer contre toute **régression** (ce qui marchait avant doit continuer de marcher COMME avant) ⇒ on a ainsi beaucoup moins peur de modifier le code

⇒ **A exécuter après chaque modif de code et surtout AVANT tout commit avec git**

⇒ **Tendre vers l'approche TDD : écrire le test AVANT la fonctionnalité à tester**

- D'abord le test ne passe pas (ROUGE, car la fonctionnalité n'est pas encore écrite)
- On écrit alors la fonctionnalité pour faire passer le test
- Maintenant le test doit passer (VERT)
⇒ permet de développer uniquement ce qui est vraiment nécessaire et en s'appuyant sur les SPECS

14.5.2. Différents types de test

- **Tests unitaires** (test des méthodes d'une classe) :
test_<nom-du-test>_<nom-de-la-methode-testée>()
ex: 3 tests unitaires de la méthode "add_item()" d'une classe:
 - test_first_case_add_item()
 - test_second_case_add_item()
 - test_third_case_add_item()
- **Tests fonctionnels (et d'intégration)** (test des fonctionnalités du logiciel, surtout celles demandées dans les specs ; font intervenir plusieurs classes d'un même module, voire même plusieurs modules) :
test_func_<nom-de-la-fonctionnalité>()
ex: test_func_alert_complete_processing()
- **Tests de performance** :
test_perf_<nom-du-composant>()
ex: test_perf_scheduler()
- **Tests de robustesse** (stress test) :
test_stress_<nom-du-composant>()

ex: test_stress_scheduler()

14.5.3. Organisation des tests dans le contexte de Django

Dans le contexte Django, les tests sont organisés par “APP” (application ou “module”).

Exemples:

- ...
- src/**dashboard**/tests.py ⇒ tests du module “**Dashboard**”
- src/**majordome**/tests.py ⇒ tests du module “**Majordome**”
- src/**monitoring**/tests.py ⇒ tests du module “**Environment Monitoring**”
- src/**scheduler**/tests.py ⇒ tests du module “**Planner (scheduler)**” (ANCIEN MODULE)
- src/**user_manager**/tests.py ⇒ tests du module “**User manager**”
- src/**common**/tests.py ⇒ **tests généraux** ou de parties communes (Observation request building) de pyros
- ...

14.5.4. Exécution des tests

(Voir chapitre **6 - TESTS** du présent document pour plus d’infos)

Il faut d’abord activer l’environnement virtuel python:

```
$ cd PYROS/  
$ source private/venv_py3_pyros/bin/activate  
(Windows) $ private\venv_py3_pyros\Scripts\activate
```

Depuis cet environnement virtuel, on exécute les tests ainsi:

```
$ cd src/  
$ ./manage.py test [app]
```

Remarques:

- *si on précise une “app” (exemple : ./manage.py test **monitoring**), on exécute seulement les tests de cette app. Par défaut, on exécute TOUS les tests de tous les modules.*
- *\$. ./manage.py test app.tests.ModelTests ⇒ run test methods declared in the class app.tests.ModelTests*
- *\$. ./manage.py test app.tests.ModelTests.test_method ⇒ only run the method test_method declared in app.tests.ModelTests*

Raccourci (qui évite d’avoir à activer l’environnement virtuel):

```
$ cd PYROS/  
$ ./pyros.py test
```

Exécution des tests SANS django:

```
$ python -m unittest test_module1 test_module2
```

```
$ python -m unittest test_module.TestClass
```

```
$ python -m unittest test_module.TestClass.test_method
```

Plus de détail sur unittest: <https://docs.python.org/3/library/unittest.html>

14.5.5. Structure d'un fichier tests.py

A savoir:

- **Un test ne doit tester qu'UNE chose**
- Un fichier **tests.py** peut contenir **PLUSIEURS tests** (une fonction **test_xxx()** par test)
- Chaque test **test_xxx()** du fichier **tests.py** est exécuté **INDÉPENDAMMENT** des autres et ces tests doivent donc pouvoir être exécutés dans n'importe quel ordre. Un test ne doit pas dépendre d'un autre test. La BD de test est réinitialisée après **chaque** test.
- Le fichier **tests.py** contient une fonction **setUp()** qui est **appelée AVANT chaque test** et crée la **fixture** (les données initiales et les tables en BD) nécessaire à l'exécution du prochain test
- Le fichier **tests.py** contient une fonction **tearDown()** qui est **appelée APRES chaque test** et peut servir à faire le ménage après un test (rarement utile)
- **Sans Django**, il peut être utile de placer ceci à la fin du fichier **tests.py**:

```
if __name__ == "__main__":  
    unittest.main()
```

14.5.6. Exemple de fichier tests.py

(inspiré de src/scheduler/tests.py):

```
# Import de la classe de Test
from django.test import TestCase
(SANS django: from unittest import TestCase)

# Import de la classe représentant le module à tester
from scheduler.Scheduler import Scheduler

# Import de la description de la BD (modèles django) ssi besoin d'interagir avec la BD
# Attention: Django utilise automatiquement la BD de test (et non pas la BD de prod)
# Cette BD sera détruite après CHAQUE test
from common.models import *

# Classe AppTest: doit hériter de TestCase (PEP8: classe écrite en camel-case)
class SchedulerTest(TestCase):

    """
    Fixture initiale (Initialisation appelée AVANT CHAQUE FONCTION test_XXX())
    Autres fonctions de setup :
    - setUpClass() ⇒ exécutée au début de CHAQUE CLASSE
    - setUpModule() ⇒ exécutée une seule fois au tout début
    """

    def setUp(self):
        self.scheduler = Scheduler()
        self.scheduler.max_overhead = 1
        scipro = ScientificProgram.objects.create()
        country = Country.objects.create()
        user_level = UserLevel.objects.create()
        self.usr1 = PyrosUser.objects.create(username="toto", country=country,
        user_level=user_level, quota=100)
        self.req1 = Request.objects.create(pyros_user=self.usr1,
        scientific_program=scipro)
        ...

    # Un premier test (Attention, PEP8 préconise l'utilisation du snake-case)
    def test_basic_1(self):
        """
        Goal : test if 3 distinct sequences are put into the planning
        """
```

Pre-conditions : the sequence have the same priority, and have no jd overlap
"""

```
self.scheduler.schedule.plan_start = 0  
self.scheduler.schedule.plan_end = 10
```

...

Assertion de test : nb_planned doit etre egal à 3:

(Voir toutes les assertions possibles ⇒ [ICI](#))

```
self.assertEqual(nb_planned, 3)
```

```
self.assertEqual(shs1.tsp, 1)
```

...

Un second test (Attention, PEP8 préconise l'utilisation du **snake-case**)

def test_basic_2(self):

"""

Goal : ...

Pre-conditions : ...

"""

...

Assertion de test

(Voir toutes les assertions possibles ⇒ [ICI](#))

```
self.assertEqual(a, b)
```

...

"""

Ménage final (fonction appelée APRES CHAQUE FONCTION test_xxx())

Autres fonctions de ménage :

- **tearDownClass()** ⇒ exécutée à la fin de CHAQUE CLASSE
- **tearDownModule()** ⇒ exécutée une seule fois à la fin

"""

def tearDown(self):

...

A rajouter éventuellement tout à la fin (uniquement si on n'utilise pas Django):

```
if __name__ == "__main__":
```

```
    unittest.main()
```

14.6. Python installation for specific operating systems

IMPORTANT FOR LINUX USERS:

Even if your python is up to date, be sure to do at least the different installations IN RED BOLD below

(for instance, on linux CentOS, it is necessary to do at least “sudo yum install python34-devel” and “sudo pip install --upgrade pip”, ...)

14.6.1. A2.1. Linux CentOS 7.1

(python35 not yet available as rpm ?)

```
$ sudo yum update yum
$ sudo yum update kernel
$ sudo yum update
$ sudo yum install yum-utils
$ sudo yum groupinstall development
$ sudo yum install https://centos7.iuscommunity.org/ius-release.rpm
$ sudo yum install python34
```

```
$ python3.4 -V
Python 3.4.3
```

```
$ sudo yum install python34-devel
(needed for python package mysqlclient)
```

```
((
NO MORE NECESSARY:
$ sudo yum update python-setuptools
$ easy_install --version
setuptools 0.9.8
$ sudo easy_install pip
$ pip --version
pip 8.1.1 from /usr/lib/python2.7/site-packages/pip-8.1.1-py2.7.egg (python 2.7)
))
```

```
$ sudo pip install --upgrade pip
```

Necessary for "lxml" python package:

```
$ sudo yum install libxml2 libxml2-devel
```

```
$ sudo yum install libxslt libxslt-devel
```

14.6.2. A2.2. Linux Ubuntu, Suse

```
$ sudo add-apt-repository ppa:fkruhl/deadsnakes
```

```
$ sudo apt-get update
```

```
$ sudo apt-get install python3.5
```

```
$ sudo apt-get install python3.5-dev (or python3.6-dev) if you're using python 3.6
```

```
(needed for python package mysqlclient && lxml)
```

```
$ sudo apt-get install libxml2-dev
```

```
$ sudo apt-get install libxslt-dev
```

```
$ sudo apt-get install zlib1g-dev can be required too
```

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

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

```
((
```

```
NO MORE NECESSARY
```

```
$ sudo pip install --upgrade virtualenv
```

```
))
```

14.6.3. A2.3. Mac OS X

- **From Brew (recommended)**

```
Python d'origine sur Mac = Python2 :
```

```
$ which python
```

```
/usr/bin/python
```

```
Install HomeBrew :
```

```
(TODO)
```

```
Install Python3 :
```

```
$ brew doctor
```

```
$ brew update
```

```
$ brew upgrade
```

```
$ brew install python3
```

```
$ brew info python3
```

```
$ python3 -V
```

```
Python 3.6.4
```

```
$ which python3
```

```
/usr/local/bin/python3
```

- **From MacPort**

Install macport :

<https://www.macports.org/install.php>

Install the "port" python36

```
$ sudo port install python36
$ sudo port select --set python3 python36
$ sudo port install py36-readline
$ sudo port install py36-pip
$ port select --set pip pip36
```

14.6.4. A2.4. Windows 10

Go to <https://www.python.org/downloads/windows/> , choose the wanted version

On the wanted version's page, download Windows x86 executable installer

Run the executable :

- * On the first page, check "Add python3.5 to PATH"
- * Choose "Install now" option

Open cmd (windows + R, cmd) :

```
$ python -m pip install --upgrade pip
```

If ever the "python" command does not work, it means that you have to add it to your PATH :

- Type key WINDOWS + PAUSE
- Click on "Paramètres système avancés"
- Click on button "Variables d'environnement"
- Click on "Variables système"
- Select line « Path »
- Click on "modify",
- Click on "New"
- Add the path to your python.exe executable (or the Anaconda folder)
- Click on "OK"

Now, test that python can be executed:

- Open a new command window : WINDOWS + R
- type « cmd »
- type « python »

14.7. MySQL installation for specific operating systems

If the MySQL database server is already installed on your computer, skip this section

For more information, see section “[Notes about MySQL](#)”

14.7.1. A3.1. Linux CentOS

cf https://www.howtoforge.com/apache_php_mysql_on_centos_7_lamp#installing-mysql-

First, update your system:

```
$ sudo yum update yum
```

```
$ sudo yum update kernel
```

```
$ sudo yum update
```

```
$ sudo yum install mariadb-server
```

```
$ sudo yum install mariadb
```

```
$ sudo yum install mariadb-devel
```

(needed for python package mysqlclient)

```
$ sudo systemctl start mariadb.service
```

```
$ sudo systemctl enable mariadb.service
```

```
=> Created symlink from /etc/systemd/system/multi-user.target.wants/mariadb.service to /usr/lib/systemd/system/mariadb.service.
```

```
$ sudo mysql_secure_installation
```

14.7.2. A3.2. Linux Ubuntu, Suse

First, update your system:

```
$ sudo apt-get update
```

```
$ sudo apt-get install mysql-server
```

```
$ sudo apt-get install mysql-client
```

```
$ sudo apt-get install libmysqlclient-dev
```

(needed for python package mysqlclient)

To resolve auth problems for root user check this link

<https://unix.stackexchange.com/questions/396738/cant-access-mysql-without-running-sudo>

14.7.3. A3.3. Mac OS X

Install MySQL with brew (recommended) or macport, or install XAMPP

(<https://www.apachefriends.org/fr/index.html>)

- With brew (recommended) :

Tested with Mysql 5.7.21

```
$ brew doctor
$ brew update
$ brew upgrade
$ brew install mysql
$ mysql -V
```

Now, start the Mysql server :

```
$ mysql.server start
```

Now, connect to the Mysql server with the mysql client :

```
$ mysql -u root
mysql> exit
```

14.7.4. A3.4. Windows 10

Download and install the newest version on <https://dev.mysql.com/downloads/installer/>

Once installed, launch MySQL Installer.

Click on 'Add...' on the right.

In MySQLServers section, choose the newest, then click on NEXT.

Install and configure the server (just follow the installation guide).

Then launch mysql (via the Windows menu).

14.7.4.1. Installation of PHP to use phpmyadmin

Install PHP5 in C:/php.

- Copy c:/php/libmysql.dll into c:/windows/system32
- Copy c:/php/php.ini-recommended as c:/windows/php.ini

Edit the file c:/windows/php.ini, and change it as follows:

```
...
; Directory in which the loadable extensions (modules) reside.
extension_dir = "c:/php/ext/"
...
; Dynamic Extensions ;
...
extension=php_mbstring.dll
extension=php_mysql.dll
extension=php_mysql.dll
```

Do not forget the i in the name php_mysql.dll. php_mysql.dll is important for PHP scripts. We may test the operation of PHP with Apache by writing the script info.php containing the following line:

```
<?PHP phpinfo(); ?>
```

We will put the script info.php in the Apache htdocs folder.

It is sometimes possible that PHP cannot find the file php.ini in the c:/windows folder. We must then leave it in the c:/php folder and append the following line in httpd.conf for Apache, just after the line *LoadModule php5_module* :

```
PHPIniDir "C:/php/"
```

14.7.4.2. Install phpmyadmin

Unzip PHPMyAdmin in htdocs/, and rename the folder htdocs/PHP... to htdocs/phpmyadmin. Copy phpmyadmin/config.sample.inc.php as phpmyadmin/config.inc.php, then edit the file phpmyadmin/config.inc.php and modify it as follows:

```
...
$cfg['blowfish_secret']          = 'php my admin';
$cfg['Servers'][$i]['extension'] = 'mysql';
$cfg['Servers'][$i]['user']      = 'root'; // MySQL user
$cfg['Servers'][$i]['password'] = 'PASSW_mysql_root'; // MySQL
password
```

14.8. Python packages installation for specific operating systems

See the files REQUIREMENTS*.txt in the folder PYROS/install

14.9. Annex 5: Notes for Eclipse IDE users

1) Install Eclipse (if necessary) and the PyDev plugin

Install Eclipse

(optional, can be done later) Install the plug-in PyDev (via install new software, add <http://pydev.org/updates>)

How to configure PyDEV :

- General doc : <http://www.pydev.org>
- For Django : http://www.pydev.org/manual_adv_django.html

2) Import the PYROS project

- a) If **PYROS is already on your file system** (cloned with git from the terminal, [see section above](#))

Just import your PYROS project from the file system :

File Import / Existing projects into workspace / Next

Select root directory : click on “Browse” and select your PYROS directory

Click on “Finish”

- b) If **PYROS is not yet on your file system** (not yet cloned with git)

You must clone the PYROS project with git from Eclipse :

File/Import project from git

Select repository source: Clone URI: <https://gitlab.irap.omp.eu/epallier/pyros.git>

Directory:

par défaut, il propose : /Users/epallier/git/pyros

mais on peut le mettre ailleurs (c'est ce que j'ai fait)

initial branch: master

remote name: origin

Import as general project

Project name: PYROS

If necessary, to deactivate CA certificate verification

Window -> Preferences -> Team -> git -> configuration -> Add entry

Key = http.sslVerify

Value = false

Si le plugin PyDev n'est pas encore installé, voici un truc simple pour le faire :

Ouvrir un fichier python

Eclipse propose automatiquement d'installer PyDev

Switch to the DEV branch :

Right-click on project, Team/Switch to/dev

Optional :

Install the django template editor (via install new software, add <http://eclipse.kacprzak.org/updates>)

3) Configure the project

The project is created.

Now, if this has not been automatically done by Eclipse, you have to set the project as a «PyDev » and a « Django » project.

clic droit sur le projet / PyDev / set as a PyDev project

clic droit sur le projet / PyDev / set as a Django project

Clic droit sur le projet : on doit maintenant avoir un sous-menu Django

Clic droit sur le dossier src : PyDev / set as source folder (add to PYTHONPATH)

Do the same for the folder “simulators”

clic droit sur le dossier du projet : Properties / Pydev-Django :

- **Django manage.py** : **src/manage.py**

- **Django settings module** : **pyros.settings**

4) Set the python interpreter

Now, once the Python3 virtual environment is created ([see above](#)), set it in Eclipse as the project interpreter:

Right clic on the project : Properties / PyDev - Interpreter/Grammar

Interpreter : click on “click here to configure an interpreter not listed”

Click on « New... » :

- Interpreter name : **venv_py3_pyros**

- Interpreter executable : click on « Browse »

Select your python virtualenv executable from inside your PYROS project (private/venv_py3_pyros/bin/python)

Click “Open”

Click OK

A new window “Selection needed” opens

Unselect only the last line with “site-packages”.

Click OK

Interpreter : **click again on** “click here to configure an interpreter not listed” !!!!!!!
Select the interpreter you just created and which is named “venv_py3_pyros”
Click on the tab "Libraries"
Click on 'New folder', then select your virtualenv's lib/python3.5/site-packages folder
OK
Click on “Apply and Close”
Interpreter: select now venv_py35_pyros from the list
Click on “Apply and Close”

5) (Optional) Set Code style

Eclipse/Preferences : Pydev / Editor

- Auto Imports : uncheck « Do auto import »
- Code style:
- Locals ... : underscore
- Methods : underscore
- Code style / Code Formatter: activer « use autopep8.py for code formatting »
- Tabs : Tab length : 4
- ...

6) Test

- Right-click on the project / Django /
 - Run Django tests
(click on the Console tab to see what's going on)
 - Custom command...
 - Shell with django environment...

7) Run

Right clic on project -> Django/Custom command/runserver

Now, check <http://localhost:8000/>

14.10. Annex 6: Notes for PyCharm IDE users

Make the following points:

1) Install Pycharm

2) import pyros project

3) Mark the src directory and simulators directory as source root directories

4) Go in file -> settings (CTRL + ALT + S) -> Project : Pyros -> Project Interpreter

Add an interpreter which is the one from your virtual environment : Add Local -> find the python 3 binary in your virtualenv

5)

For professional version :

Go in Language & Frameworks -> Django and set the django project root / Settings (pyros/settings.py) / Manage script

For community edition :

First: Go to edit configuration (top right corner)

Second: Click on the (+) mark in top-left corner and add python configuration.

Third: Click on the Script, and for django select the manage.py which resides on the project directory.

Fourth: Add <your command> as Scripts parameter and click apply : you normally should be able to run your project

14.11. Annex 7: Mount a virtual environment

Blabla.

15. ROADMAP TEMPORAIRE (TODO LIST) (à migrer dans Redmine)

Ceci est une todo list "brouillon" pour mettre rapidement et temporairement des idées qui devront ensuite être migrées dans le [REDMINE](#) :

- [rapport d'activité redmine AK](#)
- [rapport d'activité redmine EP](#)
- [rapport d'activité redmine PM](#)

⇒ Accès direct aux thèmes:

- [GENERAL](#)
- [AGENT](#)

THEME	ITEM	FAIT
	<p><i>LÉGENDE:</i></p> <ul style="list-style-type: none"> - Urgence: <i>Très Urgent, Assez Urgent</i>, par défaut = peut attendre - Type: (B) = Bug ou anomalie ; (I) = Improvement (amélioration) ; (N) = Nouvelle fonctionnalité ; (M) = Modification (changement) 	
GENERAL		
Script général install.py	Script général install.py	01/02/19 (EP)
Script général pyros.py	<p>Lancer plusieurs agents en 1 commande : pyros start agentA, agentB, ...</p> <p>Pyros-shell: => doit fonctionner depuis src/ Déplacer src/config/ dans PYROS/ ? <u>Ex: Dans un pyros shell:</u> id_cmd = agent.send_command("AgentX","Generic","EVAL", "3+4") res = Agent.get_result(id_cmd): => Res = 7</p>	22/03/19 (EP)
Wrapping de code C	Wrapper du C dans python : avec Cython ?	(AK)

	https://medium.com/@shamir.stav_83310/making-your-c-library-callable-from-python-by-wrapping-it-with-cython-b09db35012a3	
LOG	<p>LOG général (django log)</p> <p>⇒ https://lincolnloop.com/blog/django-logging-right-way/</p>	
CONFIG	<p>Fichier config est dans src/config/, est-ce que ça pose pb ? le mettre dans /PYROS ?</p> <p>Store config in environment variables (separate config from the source code): https://12factor.net/config</p> <p><i>Env vars are easy to change between deploys without changing any code; unlike config files, there is little chance of them being checked into the code repo accidentally; and unlike custom config files, or other config mechanisms such as Java System Properties, they are a language- and OS-agnostic standard. In a twelve-factor app, env vars are granular controls, each fully orthogonal to other env vars. They are never grouped together as “environments”, but instead are independently managed for each deploy. This is a model that scales up smoothly as the app naturally expands into more deploys over its lifetime.</i></p>	1/3/19 (AK)
Useful python packages to use	<p>Ajouter django_debug_toolbar (https://django-debug-toolbar.readthedocs.io/en/latest/installation.html)</p> <p>Use fire for CLI (instead of click): https://google.github.io/python-fire/guide/</p> <p>Better exceptions: https://github.com/Qix-/better-exceptions</p> <p>Home assistant (https://www.home-assistant.io/) & Hass.io (https://www.home-assistant.io/blog/2017/07/25/introducing-hassio)</p> <p>Ajouter pydbg dans requirements pour faciliter le debug (https://github.com/tylerwince/pydbg) :</p> <ul style="list-style-type: none"> • pip install pydbg • from pydbg import dbg <p>Date & time for humans: https://github.com/kennethreitz/maya</p> <p>Utiliser black pour formater le code :</p> <ul style="list-style-type: none"> • https://github.com/ambv/black 	

	<ul style="list-style-type: none"> • https://www.developpez.com/actu/207399/Black-l-outil-de-format-age-de-code-Python-transforme-les-guillemets-droits-simples-en-guillemets-doubles-les-auteurs-expliquent-leurs-choix/ <p>Utiliser pycodestyle pour vérifier la conformité au format PEP8</p> <ul style="list-style-type: none"> ■ https://pypi.org/project/pycodestyle/ 	
Agent & Command		
DOC	<ul style="list-style-type: none"> - Faire un plantUML de run() —=> To be updated - Command State diagram 	<p>(EP) 28/3/19</p> <p>1/4/19</p>
Agent générique	<p>Agent générique complet avec thread et processus pour traitement spécifique (et sous-classe AgentX)</p> <ul style="list-style-type: none"> - Command Sender = agent ou pyros-shell ou dashboard (cde manuelle web) - Arrêter une commande RUNNING après un timeout — Eviter le script start_agent.py => migrer son contenu dans Agent - routine_process() — Démarrage en mode test (simu) avec option “-t” - Implémenter le go_idle correctement sur agentA : n’envoie plus de commandes — Gérer timeout si commande pas exécutée en temps raisonnable - Format commande : generic et pyros (en cours) set , get, do - SET status active - SET status idle - GET status - GET mode 	<p>22/03/19 (EP)</p> <p>29/3/19 (EP)</p>

	<ul style="list-style-type: none"> - DO exit - DO restart_init - DO abort - DO flush_commands <ul style="list-style-type: none"> — Implémenter la commande RESTART_INIT — Methode Agent.send_command(AgentDestinataire, cmd_type, cmd_msg): - Grammaire des commandes: <ul style="list-style-type: none"> - Cmd_type = Generic, Pyros - Cmd_msg = dépend du Cmd_type - Si Generic alors IDLE, STOP, ABORT, RESTART_INIT, EVAL - Si Pyros alors <ul style="list-style-type: none"> - Device = MOUNT (=unit), CHAN - Action = START STOP ABORT SET GET <p>Réordonner les colonnes de la table "command":</p> <ul style="list-style-type: none"> - ALTER TABLE command MODIFY COLUMN recipient VARCHAR(50) AFTER sender; - ALTER TABLE command MODIFY COLUMN r_start_time datetime(6) AFTER r_read_time; 	
Multi-agents	<p>2 agents agentA, agentB s'envoient mutuellement des commandes (+ scenario test)</p> <ul style="list-style-type: none"> — Ne faire le test assertions que quand dest = "myself" — Self.name = AgentX ou agentX? — Idem mais AgentA envoie cde à AgentB et récupère le résultat — Lancer AgentA et AgentB en une seule commande "pyros start AgentA, AgentB" avec un scenario testé — Afficher messages "ok ou échec" pour tous les process à la toute fin de start() 	22/03/19 (EP)
TEST (simulation)	Abort : envoyé même si commande envoyée en cours	29/3/19
LOG		
Agent Env-Monitoring	Nouvel AgentM	29/3/19

	Reprendre le travail de PM et l'intégrer à django	
SuperAgent	<p>Démarre agents A, B, et X, et leur envoie des commandes puis les stoppe.</p> <p>Depuis ce SuperAgent, on peut implémenter des scénarios de tests (simulation), c'est à dire une suite de commandes envoyées aux différents agents et devant mener à un résultat final connu d'avance (objectif final).</p>	
Sous-types d'agent	<p>SuperAgent : lance les autres agents, les surveille, les relance si bloqués, leur envoie des commandes, et les stoppe</p> <p>AgentService : idem Agent, classique => Majordome, Monitoring, Scheduler, AlertManager, ...</p> <p>AgentDevice : connecté par socket à un device pour à la fois récupérer régulièrement son status (et le mettre dans la BD) et lui envoyer des commandes spécifiques (demandées par les autres agents) ; fusionner cette classe avec la classe DeviceController actuelle pour obtenir le meilleur des 2 mondes ; ex: TelescopeController, CameraController, FilterWheelController, PLCController, ...</p> <p>Ex: AgentTelescope :</p> <ul style="list-style-type: none"> - socket.gethostname() <ul style="list-style-type: none"> - 'macp1219.local' - Une table dédiée agent_telescope contenant variables d'état: <ul style="list-style-type: none"> - Tel_create - Tel_init - tel_park - Tel_motion - Tel_flatscreen - tel_init_done - init() : <ul style="list-style-type: none"> - Init telescope (prend plusieurs minutes) - routine_process(): <ul style="list-style-type: none"> - Verif connection telescope - Get motion => reponse du tele = stopped, tracking, slewing... - Get coord => mis en BD <p>Autre type ?</p>	
VUE (Interface WEB)	Envoi commandes aux agents depuis l'interface web (start/stop, commandes)	

⇒ [Retour au début du tableau](#)

NOTE à transférer :

<https://www.abiraf.com/blog/some-solutions-if-doing-multiprocessing-in-your-django-project-doesnt-work>

close all the connections in the main thread before spawning the child processes. Each child thread will detect that it doesn't have a connection and grab a new one as needed

Django will automatically create a new connection if there is none

We also don't want to pass Django model's objects into the spawning new process arguments as it can copy the database connection too. Simple solution is just to pass the IDs of the objects and query back them again inside each process

```
# cast it to list to make sure no database connection is copied
    book_ids = list(book_ids)
```

```
# connections are copied into the new process
    # to prevent that, close current connections in the main thread
    # new connections will be created automatically in the spawned
    # process
```

MIND that we enable transaction in each request by default

```
So I add the code to prevent database transaction which is '@transaction.non_atomic_request'
@transaction.non_atomic_request
@api_view(['GET'])
def generate_data_multiprocess(request):
    ...
```

QUESTIONS

1 - Si agentB reçoit "abort" de agentA =>

il stoppe seulement la commande EN COURS d'exécution ?

ou bien

il supprime aussi toutes les commandes de agentA qui sont en attente ? (celles qui sont AVANT le "abort") ?

Cas problématique:

agentB a reçu les commandes "specific1" PUIS "abort", mais il n'a pas encore commencé à exécuter "specific1" (PENDING).

Il va exécuter "abort" mais ça ne va rien faire puisque rien n'est en cours d'exécution.

Puis, il va exécuter "specific1" !!!

2 - On a la notion de "commande périmée" pour une commande RECUE dont la date de dépôt est trop ancienne.

Mais que faire d'une commande ENVOYEE par agentA à agentB et qui n'est jamais exécutée (car agentB n'est pas lancé par exemple) ?

Comment AgentA décide-t-il d'arrêter d'attendre l'exécution de cette commande ?

On applique le même principe de péremption dans les 2 sens ?

Je suppose donc que agentA ET agentB peuvent tous les 2 marquer cette commande comme étant périmée ?

15.1. OLD TODO LIST (déjà migrée dans Redmine)

Cette liste a déjà été migrée dans le Redmine, elle n'est donc plus maintenue.

La TODO LIST officielle est désormais sur le [REDMINE](#) :

- - (EP 5/2/19) **./pyros (ex pyros.py) init_database()** : ne doit plus contenir la fonction `init_database()` car elle n'est utilisée que par le script `install.py` => move cette fonction to `install.py` (elle sera utilisée à la fois pour une install normale et un update)
- - (EP 5/2/19) **./install.py update** : remettre en place
 - OLD => C'était utilisé dans `install_old.py` (JB) : update faisait seulement "pyros.py `init_database`", ce qui faisait :
 - Update DB : `makemigrations + migrate`
 - `loaddata()` : chargement de la fixture initiale `src/misc/fixtures/initial_fixture.json`
 - NEW => il faudrait refaire ça (maybe sans le `loaddata()` ? mais c'est peut-être utile à garder...)
 - Update DOC : une section UPDATE qui dit : pour se mettre à jour, faire :
 - 1) git pull
 - 2) `pyros update` (qui appellera "install.py update" qui appellera `init_database`)

- - (EP 5/2/19) **bugfix install.py** ne marche pas avec mysql 5.5 sur linux (hyp2) :

```
-----Launching mysql to create database and create and grant user
pyros-----
MySQL version is 5.5
-----Please enter your MYSQL root password-----
Enter password:
ERROR 1064 (42000) at line 1: You have an error in your SQL syntax; check the manual that
corresponds to your MariaDB server version for the right syntax to use near 'IF NOT EXISTS
pyros' at line 1
Traceback (most recent call last):
  File "install.py", line 364, in <module>
    if INSTALL_DB: install_database(VEENV)
  File "install.py", line 314, in install_database
    stderr.write(Colors.ERROR + "ERROR !: db configuration failed !" + Colors.END + "\r\n")
NameError: name 'stderr' is not defined
-----Process execution failed-----
```

- - (EP 1/2/19) **Voeventparse** Windows : comment installer automatiquement via pip (et non pas manuellement comme c'est le cas actuellement) ?

- ~~(EP 1/2/19)~~ **Bugfix start_monitoring**

- - (EP 1/2/19) Faire évoluer le script pyros pour :
 - pyros **--configfile** <config-file-name> **start** <agent-name> => on peut avoir plusieurs fichiers de config différents (chaque fichier utilise un simulateur ou unit différent)
 - Parser XML AK:
 - Cd config/
 - ./test_config_xml1.py
- - (EP 1/2/19) Exceptions au lieu de return (errno, return_value)

- ~~(EP 1/2/19)~~ Move TODOLIST dans Redmine

- - (AK 1/2/19) Requirements Windows installés manuellement => à éviter
 - ~~Package lxml~~
 - Package voeventparse
- - (EP 29/1/19) Utiliser **pyenv** pour l'installation de python et **pipenv** pour la gestion des packages (pip) et requirements (Pipenv will automatically convert your old requirements.txt into a Pipfile)
 - => <https://hackernoon.com/reaching-python-development-nirvana-bb5692adf30c>
 - Gérer aussi les dev-req et stable-req
- - (EP 28/1/19) Résoudre le pb avec la BD test_pyros lors de l'exécution des tests (pyros.py unittest) :
 - "access denied for user pyros@localhost"
 - Faut-il la créer avec le script d'installation ou bien laisser django le faire au moment de l'exécution des tests ? est-ce que ça marche bien avec un vieux sql ? sur Windows ?
- - (EP 28/1/19) Faire un agent générique AgentX qui hérite de Agent
- (EP 27/1/19) : **Divers**
 - **INFO:**
 - Samuel Ronayette : nouveau sur AIT/AIV
 - UNIT = 1 telescope, mais en fait 1 coupole = 1 monture (mount)
 - 1 fichier de config XML = 1 unit
 - CHANNEL = une voie d'instrumentation
 - 1 UNIT est attaché à un toit roulant (mais il peut y avoir plusieurs unit sous un même toit roulant)
 - NODE = plusieurs units

- Telescopes :
 - Valencia 50cm
 - Chinois Tango (Tibet) 50cm

- - Isoler les paramètres d'un UNIT (telescope) en dehors du projet pyros :
 - pyros_unit_taca/ => une unit
 - Depuis pyros.settings, aller chercher la conf xml de la unit

- - Faire un script pour update (git pull + DB sync)
 - Faire un pyros « update »:
 - Devra inclure \$ git -c http.sslVerify false pull

- - Fichier config pour install
 - WIN: Trouver un moyen de trouver le chemin de mysql automatiquement (ou à la façon de « ros install » en mode graphique ou en mode console, et écrire le fichier de config correspondant aux réponses données pour ne pas avoir à les redonner lors d'une 2ème install (car les propositions par défaut seront ok)

16. SOFTWARE HISTORY

Année 2019

28/02/19 : Version 0.20.2 (EP)

Agent.py improvements :

- update_db_survey() : ajouté date created...
- read_db_commands() : ajout table AgentsCommand

27/02/19 : Version 0.20.1 (EP)

- created this "Software history" section
- pyros.py script improvements :
 - config relative ou absolue
 - all private methods with "_" prefix
 - die on any error
 - added commands :
 - pyros update (= git pull + update db)
 - pyros initdb
 - cleanup
 - bugfix start_agent.py

17/01/19 : Version 0.20.0 (EP)

- NO MORE CELERY (removed from everywhere)
-

Année 2018

10/10/18 : Version 2018.0.011 (E. Pallier)

- NEW versionning plan : YYYYMMDD.M.mmm.b
 - YYYYMMDD = Day (UTC)
 - M = Major version number
 - mmm = Minor version number
 - b = Bugfix number
- Global Version number is now in src/___init___.py (idem for each pyros application)

- new devices_channel/ folder containing the new DeviceControllerAbstract and ClientChannelAbstract classes (independant from pyros for the moment, but will soon be integrated)
- new src/utills/celme (celestial mechanics) library from Alain Klotz (more and more used inside pyros)
- better README.md formatted file

****14/05/18 : Version 2018.0.001 (Q. Durand)****

- Update all dependencies, update to django 2 and work in progress on the dashboard

****23/03/18 Version 0.16.11 (E. Pallier)****

- All unit tests now pass WITHOUT celery (./manage.py test or ./pyros.py test)

****22/03/18 Version 0.16.10 (E. Pallier)****

- New global variable USE_CELERY (false by default) in pyros.settings

****13/03/18 Version 0.16.7 (E. Pallier)****

- Each agent (envmonitor, majordome, alert) can be started independently with a script (new script start_agent in each agent directory)
- New command "pyros start" to start pyros in different ways (different options : everything or only some components)

****22/02/18 Version 0.16.5 (E. Pallier)****

- Better isolation of Monitoring for easier testing (+ tuto)

****01/02/18 Version 0.16.3 (Q. Durand)****

- New installation script src/install/install.py multiplatform (Lin-Mac-Win10, but also tested on Win7)