Documentation for the PYROS software

COLIBRI/GCC OGS (Observatory & Observations Control Software)

Official URL for this document : **tinyurl.com/colibri-pyros**

Please display the **outline** for this document ⇒ menu **View/"Show document outline"**

To get a copy (word or pdf) of this doc : menu "File/Download as/…"

Other topics (links) :

- **PLAN DE DEV (dev plan schedule)**: https://docs.google.com/spreadsheets/d/1Gy6c-9fhUDBx5Bc6YrcGvfgP-HukWdhUh8oT11_y88g
- **GITLAB IRAP** (last commits) **:** https://gitlab.irap.omp.eu/epallier/pyros/commits/dev
- **(soon) Nouveau GITLAB** (Source Code management) for this project : https://gitlab.lam.fr/svom-colibri/ocs-pyros
- TODOLIST (forge redmine) : https://projets.lam.fr/projects/pyros/issues?query_id=13
- Development technical doc : https://projects.irap.omp.eu/projects/pyros/wiki/Project_Development

SUMMARY :

1. WHAT'S NEW (atelier #4 - 19/3/18)

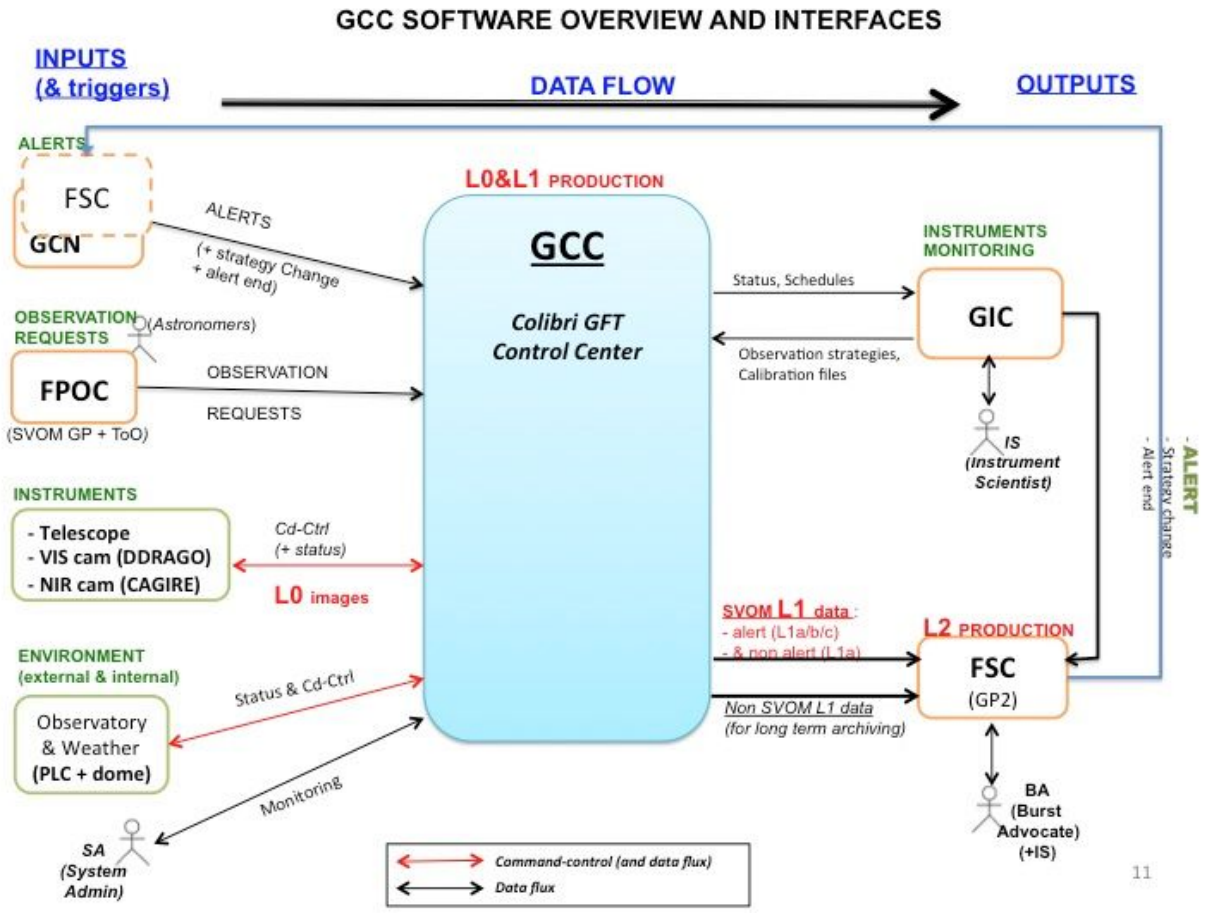
(updated 15/3/18)

- **Official URL** for this document : **tinyurl.com/colibri-pyros**
- Rappel du **plan** de ce doc : menu **View / "Show document outline"**
- Pour savoir si un chapitre a changé, regarder la date de mise à jour au début du chapître (ex: ***updated 13/3/18***)
- Récupérer une **copie** (word ou pdf) de ce doc : menu "File/Download as/…" (il y a déjà une copie dans le projet gitlab, dossier doc/)
- N'hésitez pas à mettre à jour cette doc, c'est aussi la vôtre : **IN ENGLISH**
- **Derniers commits de code** (GITLAB IRAP, branche "dev") : https://gitlab.irap.omp.eu/epallier/pyros/commits/dev
- **Mettez à jour** régulièrement votre copie de PyROS (ou suite à un mail de commit) :
  - $ cd PYROS/
  - $ git pull
- **Responsables par module :** see **MODULES**
- **Plan de dev (rappel) :** https://docs.google.com/spreadsheets/d/1Gy6c-9fhUDBx5Bc6YrcGvfgP-HukWdhUh8oT11_y88g
- **VERSION** (à mettre à jour pour chaque commit) :
  - **Une version globale** (fichier README à la racine du projet)
  - **Une version pour chaque module** (fichier README dans chaque dossier de module, exemple src/monitoring/README)
- **Code style** for PyROS **:** see CODING STYLE
- **Procédure de commit** à respecter : see COMMIT howto (procedure)
- **Nouveau script de démarrage :** "$ ./pyros.py start_xxx" **:** see RUN

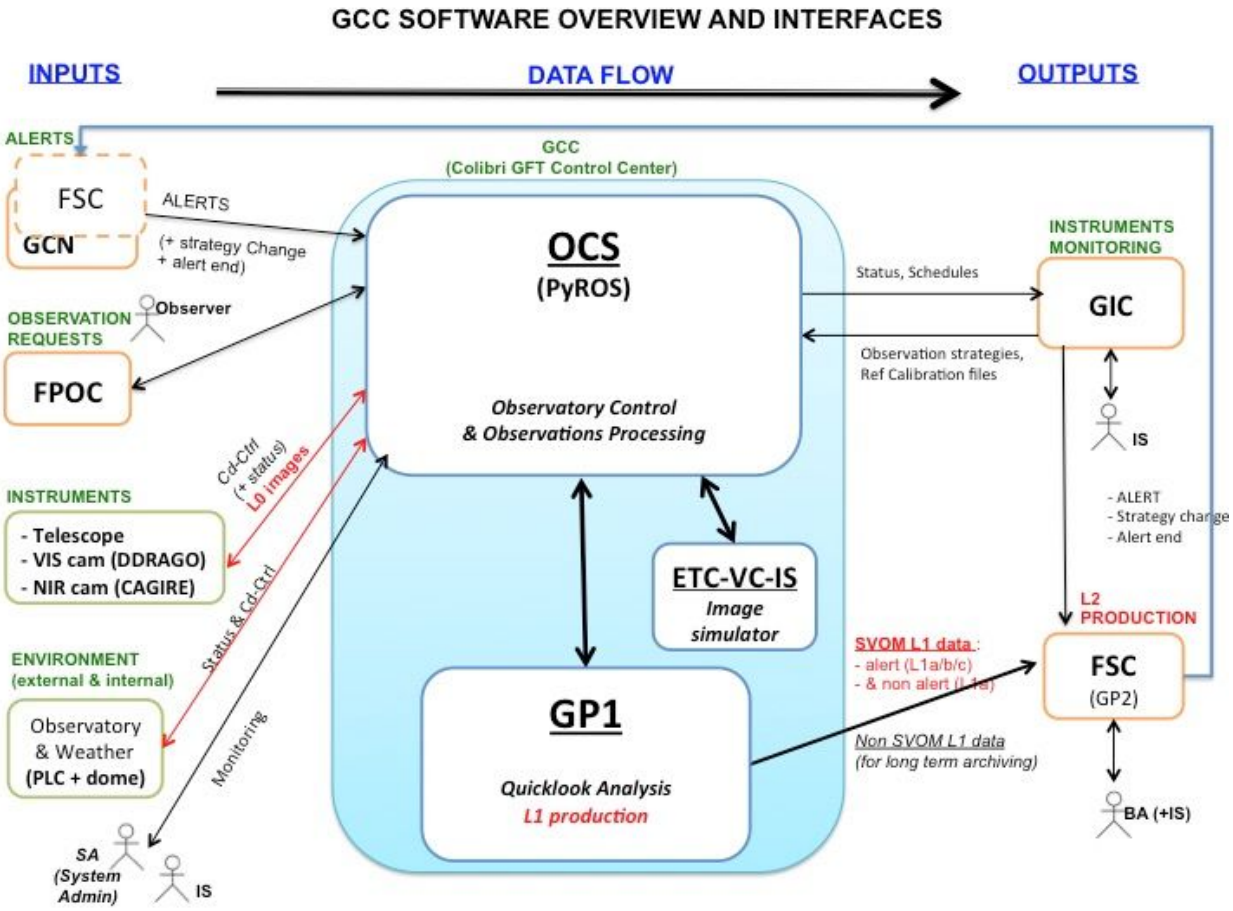
2. General Architecture

(updated 21/2/18)

2.1. GCC (GFT Control Center) General Architecture



2.2. Inside GCC



3. Installation

(updated 10/2/18)

3.1. COMPATIBLE PLATFORMS (TESTED)

This software is targeted first for Linux CentOS 7 (+ Fedora and Ubuntu), but also for Mac OS X and Windows.
All these systems should run Python 3 (3.5+)

Pyros has been tested on these platforms:

- **Linux** :
  - **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** 10.13 (with Python 3.6)
- **Windows** 10 (with Python 3.5)

3.2. THE GITLAB REPOSITORY

- 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

3.3. GET THE PYROS SOFTWARE

3.3.1. 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.

3.3.2. Get the software

First, go to the directory where you want to install the software. I 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
```

3.3.2.1. DYNAMIC VERSION (Developers) : Get a dynamic version (synchronized)

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
```

(or also, but not sure it works well : git clone git@gitlab.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
```

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

This creates 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
```

3.3.2.2. STATIC VERSION (Non developers) : Download a static version (not synchronized) :

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")

3.3.3. For WINDOWS users

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

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

You can now use your git from the cmd or the graphic client !

3.4. PROJECT STRUCTURE

- src/ : conteneur du projet (le nom est sans importance)
  - manage.py : utilitaire en ligne de commande permettant differentes 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 commited 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

3.5. INSTALLATION OF PREREQUISITES

Pyros needs some prerequisites :

- Python 3.5+ (3.6 recommended)
- RabbitMQ
- Mysql Database server (last version recommended)

3.5.1. Prerequisite 1 : Install Python3 (3.5+) + pip + lxml

- **Linux CentOS 7.1 (main target)**

(python35 not yet available as rpm ?)

$ sudo yum update yum
$ sudo yum update kernel
$ sudo yum update
$ 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

- **Linux Ubuntu** :

$ sudo add-apt-repository ppa:fkrull/deadsnakes
$ sudo apt-get update
$ sudo apt-get install python3.5
$ sudo apt-get install python3.5-dev
(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
))

- **Mac OS X** :
    - **From Brew (recommended)**

```
Python d'origine sur Mac = Python2 :
$ which python
/usr/bin/python

Install HomeBrew :
https://brew.sh/index_fr.html

Install Python3 :
$ brew doctor
$ brew update
$ br
ew 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
```

- **Windows** (tested with Win10 only) :

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) :
$ py -m pip install --upgrade pip

3.5.2. Prerequisites 2 and 3 : MySQL and RabbitMQ

The **install.py** script deals with these 2 prerequisites but **only for Linux Ubuntu and Centos (NOT FOR WINDOWS OR MAC)**. Thus, **if you are using ubuntu or centos**, launch this from the install directory:

```
$ cd install
$ sudo python3 install.py --prerequisites
```

(TODO: ajouter "systemctl enable rabbitmq…")

If you have executed this above (only for Linux Ubuntu and CentOS) go straight to section INSTALLATION OF NEEDED PYTHON PACKAGES

Otherwise, go on reading.

3.5.2.1. Prerequisite 2 : Install a DataBase Management Server (DBMS)

If the MySql database server is already installed on your computer, skip this section

For more information, see section "Notes about Mysql"

- **Linux CentOS (target platform)**

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

- **Linux Ubuntu**

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)

- **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
```

- **Windows** (tested with 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).

3.5.2.2. Prerequisite 3 : Install RabbitMQ

RabbitMQ is a message queue server used by Celery to handle tasks queues.
It uses the amqp protocol to manage queue messages.

- **CentOS** :

$ sudo yum install rabbitmq-server

Installation :

rabbitmq-server	noarch	3.3.5-17.el7

Installation pour dépendances :

erlang-asn1	x86_64	R16B-03.16.el7

Get status:
(CentOS7) $ sudo rabbitmqctl status
(older CentOS) $ sudo /sbin/service rabbitmq-server status

Stop:
(CentOS7) $ sudo systemctl stop rabbitmq-server
(older CentOS) $ sudo /sbin/service rabbitmq-server stop

Start:
(CentOS7) $ sudo systemctl start rabbitmq-server
(older CentOS) $ sudo /sbin/service rabbitmq-server start

- **Ubuntu**

Tested with RabbitMQ 3.5.7
(the server is automatically started)

$ sudo apt-get install rabbitmq-server

Get status:
$ sudo invoke-rc.d rabbitmq-server status

Stop:
$ sudo invoke-rc.d rabbitmq-server stop

Start:

$ sudo invoke-rc.d rabbitmq-server start

- **MacOS** :
    - With brew (recommended) :

Tested with RabbitMQ 3.7.2

(for more details, see https://www.rabbitmq.com/install-homebrew.html)

```
$ brew doctor
$ brew update
$ brew upgrade
$ brew install rabbitmq

RabbitMQ is now installed under /usr/local/sbin

Add
  PATH=/usr/local/sbin:$PATH
to your ~/.bash_profile or ~/.profile.

The server can then be started with :
$ rabbitmq-server &
(All scripts run under your own user account. Sudo is not required)

Get status:
$ rabbitmqctl status

To stop rabbitmq :
$ rabbitmqctl stop

The following command
$ launchctl limit
can be used to display effective limits for the current user
```

    - With MacPort:

```
$ sudo port install rabbitmq-server
--->  Installing erlang @18.2.1_1+hipe+ssl
```

```
...
--->  Installing rabbitmq-server @3.5.7_0
--->  Activating rabbitmq-server @3.5.7_0
...

To start rabbitmq :
$ sudo rabbitmq-server

Get status:
$ sudo rabbitmqctl status

To stop rabbitmq :
$ sudo rabbitmqctl stop
```

- **Windows** :

Take the wanted Erlang version at http://www.erlang.org/downloads and install it (required)

Take the wanted RabbitMQ version at https://www.rabbitmq.com/install-windows.html and install it. Then the server will run automatically

3.6. INSTALLATION OF NEEDED PYTHON PACKAGES

3.6.1. Install all the needed python packages and the PyROS database

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:**

```
$ cd install
$ python3 install.py
```

Linux and Mac : it is VERY IMPORTANT that you type "python3" and not "python"
Windows : replace "python3" with "py"

If everything went well, you can go straight to section TEST

NB: *you might need to drop your pyros database (and also pyros_test if ever it exists) before running the install script (if migrations are too big)*

If something goes wrong, install manually each package (see section "MANUAL INSTALLATION OF PYTHON PACKAGES")

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)

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

Go to next chapter 3. TEST

3.6.2. (OPTIONAL) Install the Comet python package

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)

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

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

2) 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:

3) Test Comet

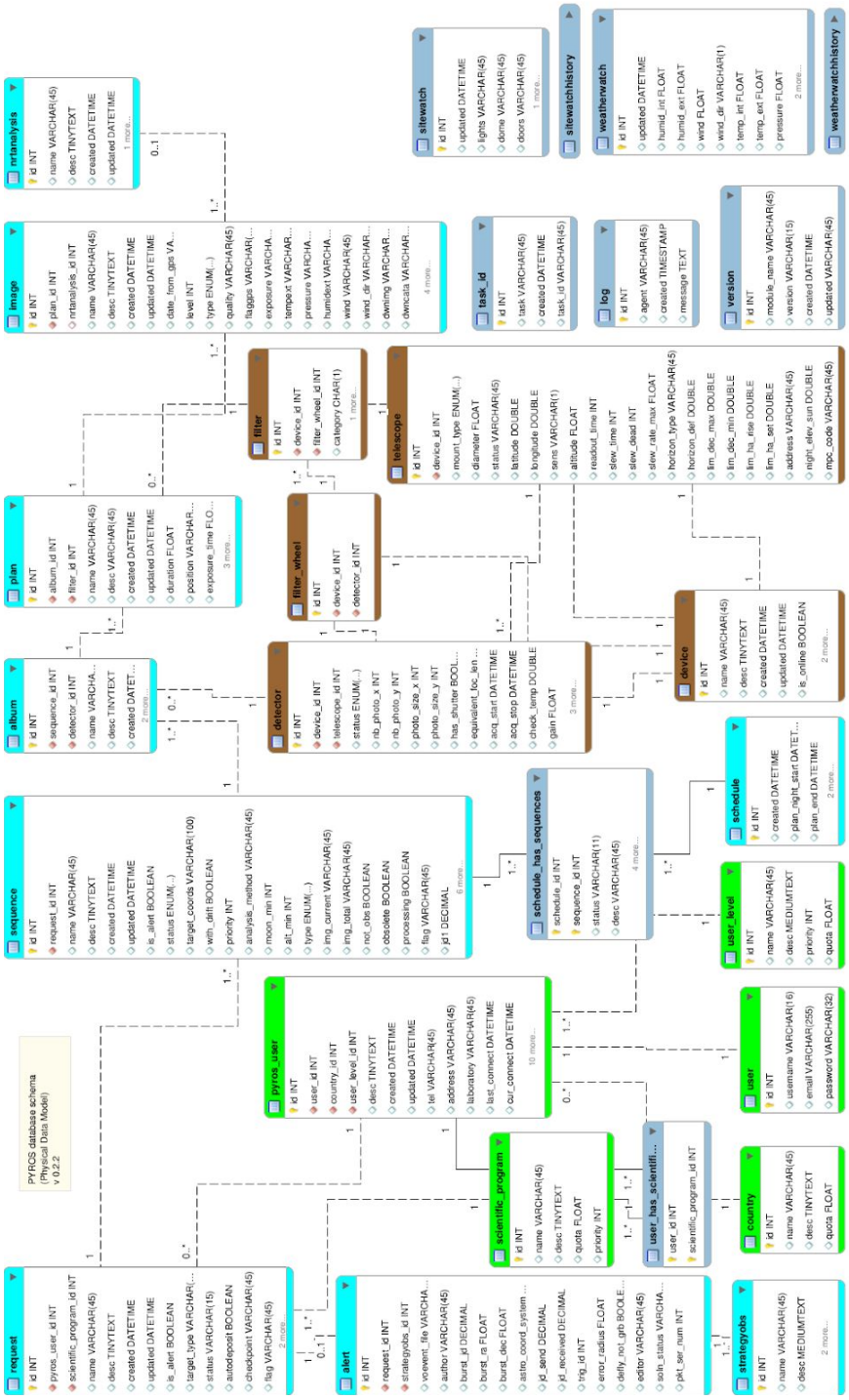
$ twistd comet --help
$ trial comet

All tests should pass

3.7. MISCELLANEOUS (just for information)

(updated 01/03/18)

3.7.1. DATABASE SCHEMA (v0.2.2)



3.7.2. NOTES FOR ECLIPSE 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-clic 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-clic 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/

3.7.3. NOTES FOR PYCHARM USERS

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

3.7.4. 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)

3.7.5. MANUAL INSTALLATION OF PYTHON PACKAGES ONE BY ONE (obsolete)

Follow these steps only if the previous guided and nearly automatic installation did not work for you

Mind that this section is OUTDATED (obsolete) and some contents might not be relevant to your real issues.

3.7.6. (Only if using Mysql) Create the database "pyros" and the pyros user

Only if you are using Mysql, you need to create an empty database "pyros"
(which will be filled automatically by django)
$ mysql -u root -p
(enter your root password)

$ mysql> create database if not exists pyros;

The user creation depends on your MySQL version :

- 5.7 and above :
- $ mysql> DROP USER IF EXISTS pyros;
  $ mysql> CREATE USER 'pyros' IDENTIFIED BY 'DjangoPyros';
  $ mysql> GRANT ALL PRIVILEGES ON pyros.* TO pyros;

- under 5.7 :
- $ mysql> GRANT USAGE ON *.* TO 'pyros';
  $ mysql> DROP USER 'pyros';
  $ mysql> CREATE USER 'pyros' IDENTIFIED BY 'DjangoPyros';
  $ mysql> GRANT ALL PRIVILEGES ON pyros.* TO pyros;

If none of these solution work, check on the internet to create a user named pyros with the password DjangoPyros.

3.7.7. Create a Python3 virtual environment dedicated to the project

$ mkdir private/

$ cd private/

$ which python3.5 ("where python" for windows)
/opt/local/bin/python3.5

$ python3 -m venv_py35_pyros -p /opt/local/bin/python3.5 ou py instead of python3 on windows
=> creates a venv_py35_pyros/ folder inside PYROS/private/

3.7.8. Activate the python virtual environment (from inside the project)

$ pwd
.../PYROS/private

$ source ./venv_py35_pyros/bin/activate (venv_py35_pyros/Scripts/activate on Windows)

3.7.9. Install needed python packages

Check that the virtual environment is activated
$ python -V
Python 3...

$ which pip
.../PYROS/venv_py35_pyros/bin/pip

Upgrade pip to last version available:
$ pip install --upgrade pip
Collecting pip
  Downloading pip-8.1.1-py2.py3-none-any.whl (1.2MB)
Installing collected packages: pip
  Found existing installation: pip 7.1.2

```
    Uninstalling pip-7.1.2:
      Successfully uninstalled pip-7.1.2
Successfully installed pip-8.1.1

Upgrade wheel to last version available:
$ pip install --upgrade wheel
Collecting wheel
  Downloading wheel-0.29.0-py2.py3-none-any.whl (66kB)
Installing collected packages: wheel
  Found existing installation: wheel 0.24.0
    Uninstalling wheel-0.24.0:
      Successfully uninstalled wheel-0.24.0
Successfully installed wheel-0.29.0
```

Go into the install/ folder:
$ cd .../PYROS/install/

Install all the needed python packages at once:
$ pip install -r REQUIREMENTS.txt

If something goes wrong, install them one by one:

- **Install Django** :
- $ pip install django
  Collecting django
    Downloading Django-1.9.4-py2.py3-none-any.whl (6.6MB)
  Installing collected packages: django
  Successfully installed django-1.9.4

  $ pip install django-admin-tools
  Collecting django-admin-tools
    Downloading django_admin_tools-0.7.2-py2.py3-none-any.whl (289kB)
  Installing collected packages: django-admin-tools
  Successfully installed django-admin-tools-0.7.2

  $ pip install django-debug-toolbar
  Collecting django-debug-toolbar
    Downloading django_debug_toolbar-1.4-py2.py3-none-any.whl (212kB)
  Requirement already satisfied (use --upgrade to upgrade): Django>=1.7 in

./venv_py35_pyros/lib/python3.5/site-packages (from django-debug-toolbar)
Collecting sqlparse (from django-debug-toolbar)
  Downloading sqlparse-0.1.19.tar.gz (58kB)
Building wheels for collected packages: sqlparse
  Running setup.py bdist_wheel for sqlparse ... done
  Stored in directory:
/Users/epallier/Library/Caches/pip/wheels/7b/d4/72/6011bb100dd5fc213164e4bbee13d4e03261dd54ce6a5de6b8
Successfully built sqlparse
Installing collected packages: sqlparse, django-debug-toolbar
Successfully installed django-debug-toolbar-1.4 sqlparse-0.1.19

$ pip install django-extensions
Collecting django-extensions
  Downloading django_extensions-1.6.1-py2.py3-none-any.whl (202kB)
Collecting six>=1.2 (from django-extensions)
  Downloading six-1.10.0-py2.py3-none-any.whl
Installing collected packages: six, django-extensions
Successfully installed django-extensions-1.6.1 six-1.10.0

$ pip install django-suit
Collecting django-suit
  Downloading django-suit-0.2.18.tar.gz (587kB)
Building wheels for collected packages: django-suit
  Running setup.py bdist_wheel for django-suit ... done
  Stored in directory:
/Users/epallier/Library/Caches/pip/wheels/12/8b/9a/e02ab0ad9229881638aa040d47d77c8f562999533811927d41
Successfully built django-suit
Installing collected packages: django-suit
Successfully installed django-suit-0.2.18

- **Install the django boostrap css package** :
- $ pip install django-bootstrap3
-
- (==> 'bootstrap3' is then to be added as an application in settings.py -> INSTALLED_APPS)

- **Install the web application server gunicorn (will be used in production instead of the dev django web server)** :

- $ pip install gunicorn
  Collecting gunicorn
    Downloading gunicorn-19.4.5-py2.py3-none-any.whl (112kB)
  Installing collected packages: gunicorn
  Successfully installed gunicorn-19.4.5

- **Install the python mysql client** (not needed if you want to use sqlite):
- $ pip install mysqlclient
  ...

-
    - => If issue under Mac OS X:
    - $ pip install mysqlclient
      Collecting mysqlclient
        Downloading mysqlclient-1.3.7.tar.gz (79kB)
      Building wheels for collected packages: mysqlclient
        Running setup.py bdist_wheel for mysqlclient ... error
      …

        Failed building wheel for mysqlclient
        Running setup.py clean for mysqlclient
      Failed to build mysqlclient
      Installing collected packages: mysqlclient
        Running setup.py install for mysqlclient ... done
      Successfully installed mysqlclient-1.3.7

      BOUH !!!

      => Need to upgrade wheel:

      $ pip install --upgrade wheel
      Collecting wheel
        Downloading wheel-0.29.0-py2.py3-none-any.whl (66kB)
      Installing collected packages: wheel
        Found existing installation: wheel 0.24.0
          Uninstalling wheel-0.24.0:
            Successfully uninstalled wheel-0.24.0
      Successfully installed wheel-0.29.0

      YES !!!

      Only if necessary, you can reinstall mysqlclient:

```
$ pip uninstall mysqlclient
$ pip install mysqlclient
Collecting mysqlclient
  Using cached mysqlclient-1.3.7.tar.gz
Building wheels for collected packages: mysqlclient
  Running setup.py bdist_wheel for mysqlclient ... done
  Stored in directory:
/Users/epallier/Library/Caches/pip/wheels/9b/06/50/d11418c26cf8f2156b13d436
3b5afde8e7e75ebb8540d0228d
Successfully built mysqlclient
Installing collected packages: mysqlclient
Successfully installed mysqlclient-1.3.7
```

-
    - => If issue under Windows
    - Same message as the issue for Mac.

      => Need to install wheel manually :

      Go to http://www.lfd.uci.edu/~gohlke/pythonlibs/#mysqlclient to download the newest mysqlclient wheel

      $ pip install path\to\mysqlclient\wheel

      (No need to redo "pip install mysqlclient")

- **Install the julian day converter** :
- $ pip install jdcal

- **Install Celery and dependencies** :
- $ pip install celery
  $ pip install django-celery
  $ pip install Twisted==16.0.0

- **Install django test without migrations (compulsory to use the prod DB for tests)** :
- $ pip install django-test-without-migrations==0.4

- **Install voevent parser** :

- $ pip install voevent-parse==0.9.5

- **Install other dependencies (useful ? TBC)** :
- $ pip install amqplib==1.0.2
  $ pip install pluggy==0.3.1
  $ pip install py==1.4.31

4. TEST

(updated 06/03/18)

Please run the tests suite, just to be sure that the software is well installed.
(Tests are classes declared in all django apps test.py file. The test classes inherit from django.test.TestCase)

First, be sure that all the pre-requisites are well installed and running (started) :

- MySQL : see Install-a-database-server
- RabbitMQ : see Install-RabbitMQ

Before launching the tests, activate your virtual environment :

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

Windows users : below, always replace "python" with "py"

Be sure that at least all unit tests pass:

```
(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) $ python pyros.py test_all
```

(for now, same tests that unittest)

If test_all passes, then run ALL tests:

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

(for now same that unittest)

Now, a **big test dedicated to the Scheduler, with only some celery workers and only the User simulator** :

```
(venv) $ python pyros.py simulator_development
```

(***If ever this test does not pass*** *try to create manually yourself the "pyros_test" database before, with the mysql client : "create database pyros_test")*
(Ctrl-c to stop) *If ever it does not stop, kill the process given by $ ps aux | grep runserver*

While this is running, go to **http://127.0.0.1:8000** in your web browser. Log in with login ‘pyros’ (in the Email field) and password ‘DjangoPyros’ to to see what’s happening (click on Schedule, on System, on Routines and then click on a request to see its detail)

Now, the real **full pyros complete test with ALL celery workers and ALL simulators (TODO: to be improved)** :

```
$ python pyros.py simulator
```

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.tests
$ ./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

5. RUN

(updated 14/3/18)

First, be sure that all the prerequisites are well installed and running :

- MySQL : see Install-a-database-server
- RabbitMQ : see Install-RabbitMQ

If not already done, activate the virtual environment:

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

(Pour désactiver l'environnement virtuel, taper "$ deactivate" depuis n'importe où)

5.1. Start 1 agent only

De manière générale, chaque agent (monitoring, alert, et majordome) peut être lancé individuellement avec le script "pyros.py" (depuis src/) :

```
(venv) $ ./pyros.py start_agent_<agent-name>
```

Par exemple, pour démarrer l'agent "environment monitoring" :

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

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

5.2. Start all agents

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

(Windows: $ python pyros.py `start_agents`)

5.3. Start the web server (the django pyros website)

```
(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 (login 'pyros' / 'DjangoPyros')
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")*

5.4. START ALL (everything : web server + agents)

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

5.5. 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...).

5.6. 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...

5.7. Play with PyROS objects (with the Django shell)

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

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

Go into the pyros project folder src/ and launch the django shell :

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

>>> from common import models
(>>> from common.models import *)
>>> 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', '__builtins__', '__cached__', '__doc__', '__file__',
'__loader__', '__name__', '__package__', '__spec__', 'models', 'unicode_literals']
```

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>
```

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.TOBEPLANNED,
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)
```

5.8. (OBSOLETE) NORMAL FULL RUN

1 - (from terminal 0) Start the Celery workers (parallel tasks queues)

(first start your venv)

```
(venv)$ python pyros.py start_workers
```

(Ctrl-c to stop)

2 - (from terminal 1) Start PyROS (with django web server)

```
(first start your venv)
(venv)$ python pyros.py runserver

OR
(venv)$ cd src/
(venv)$ python manage.py runserver
```

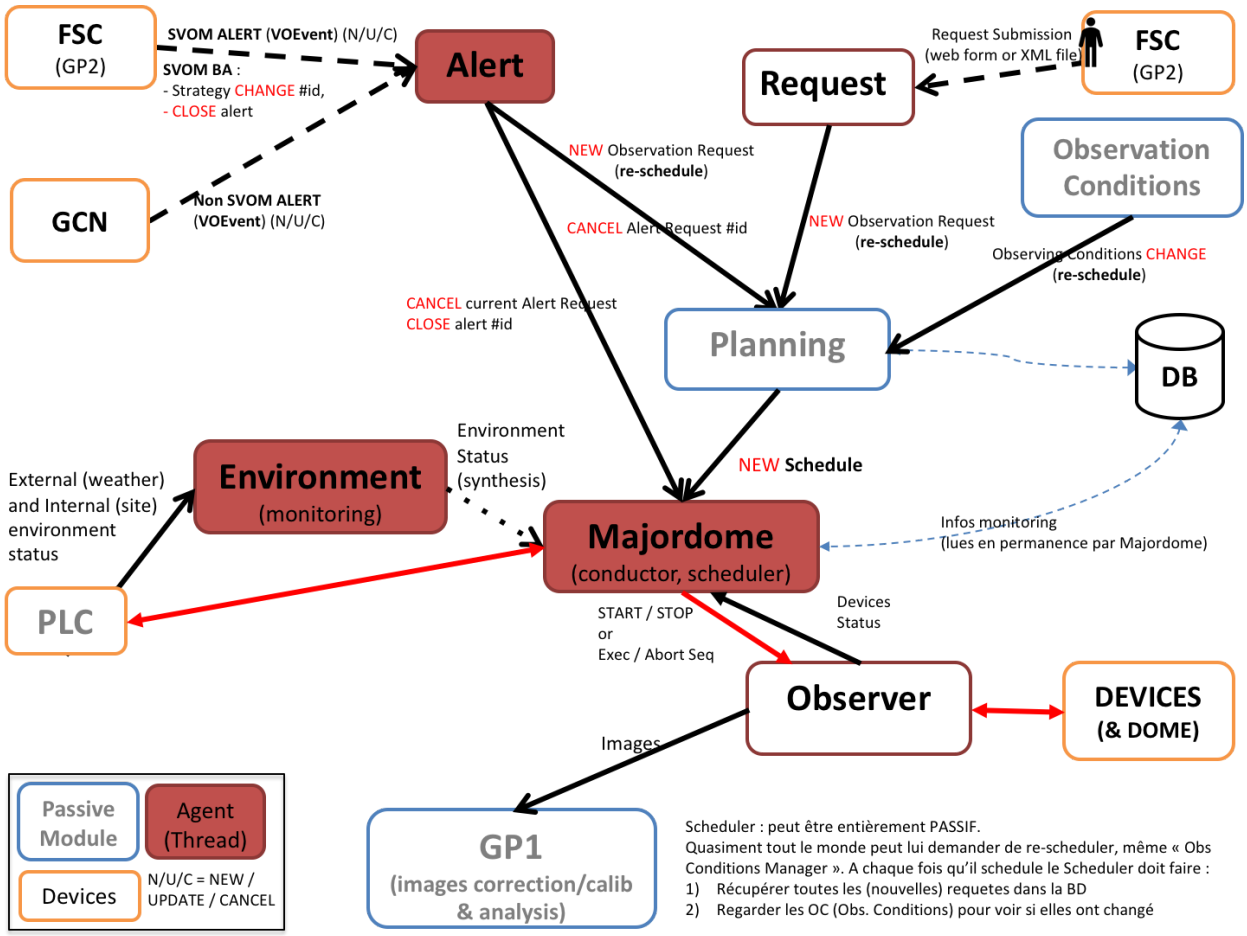
3 - (from terminal 2) Start the simulators (placeholders for real devices)

```
(first start your venv)
(venv)$ python pyros.py sims_launch
```

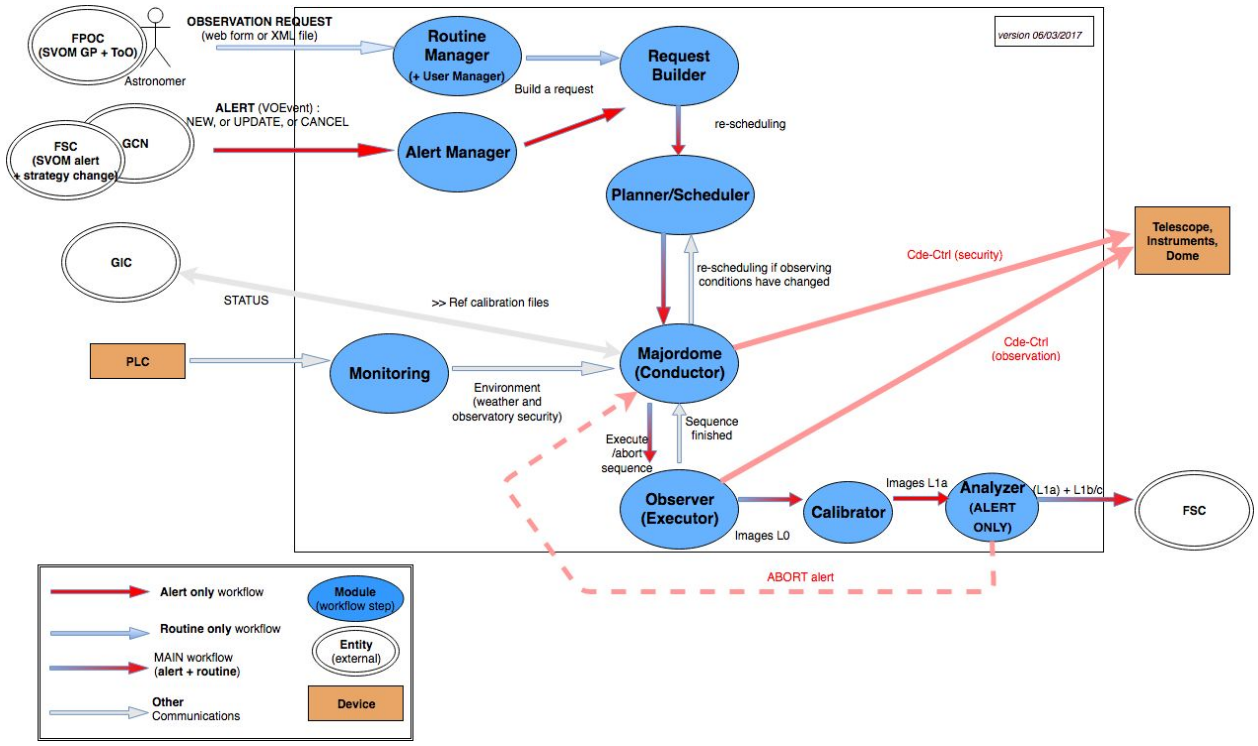
To access the PyROS website :
go to "http://localhost:8000" in your browser
Login as 'pyros' with the password 'DjangoPyros'

6. MODULES

(updated 15/3/18)



OLD VERSION:


FPOC
(SVOM GP + ToO)
Astronomer
OBSERVATION REQUEST
(web form or XML file)
Routine Manager
(+ User Manager)
Build a request
Request Builder
version 06/03/2017
ALERT (VOEvent) :
NEW, or UPDATE, or CANCEL
FSC
(SVOM alert
+ strategy change)
GCN
Alert Manager
re-scheduling
Planner/Scheduler
GIC
STATUS
>> Ref calibration files
re-scheduling if observing conditions have changed
Cde-Ctrl (security)
Telescope, Instruments, Dome
PLC
Monitoring
Environment
(weather and
observatory security)
Majordome
(Conductor)
Cde-Ctrl
(observation)
Execute
/abort
sequence
Sequence
finished
Observer
(Executor)
Images L0
Calibrator
Images L1a
Analyzer
(ALERT
ONLY)
(L1a) + L1b/c
FSC
ABORT alert
Alert only workflow
Routine only workflow
MAIN workflow
(alert + routine)
Other
Communications
Module
(workflow step)
Entity
(external)
Device

All modules are under the “/src/” directory

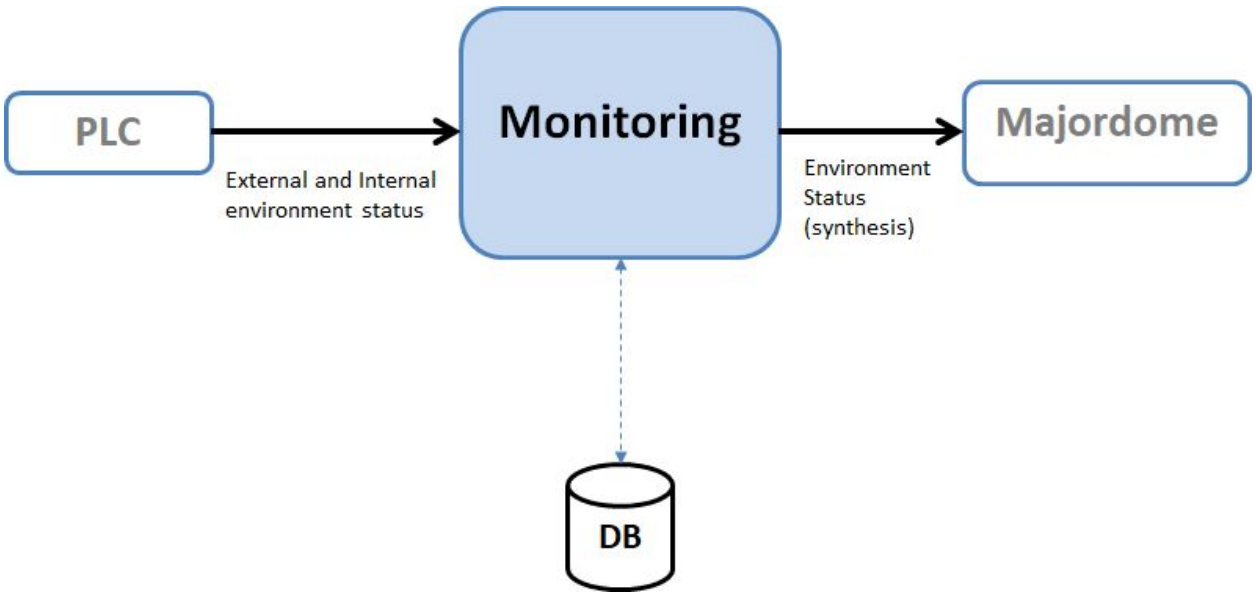
	Agent ?	**Place**	**Start**
(ENV) Environment Monitor (*P. Maeght)*	YES	src/**monitoring**/tasks.py Monitoring.run()	$ pyros.py start_agent_envmonitor
(SCHED) Scheduler (Planner) *(A. Klotz)*	(NO)	src/**scheduler**/Scheduler.py (and simulator.py) and tasks.scheduling.run()	Appel de la méthode scheduler.tasks.scheduling.run()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

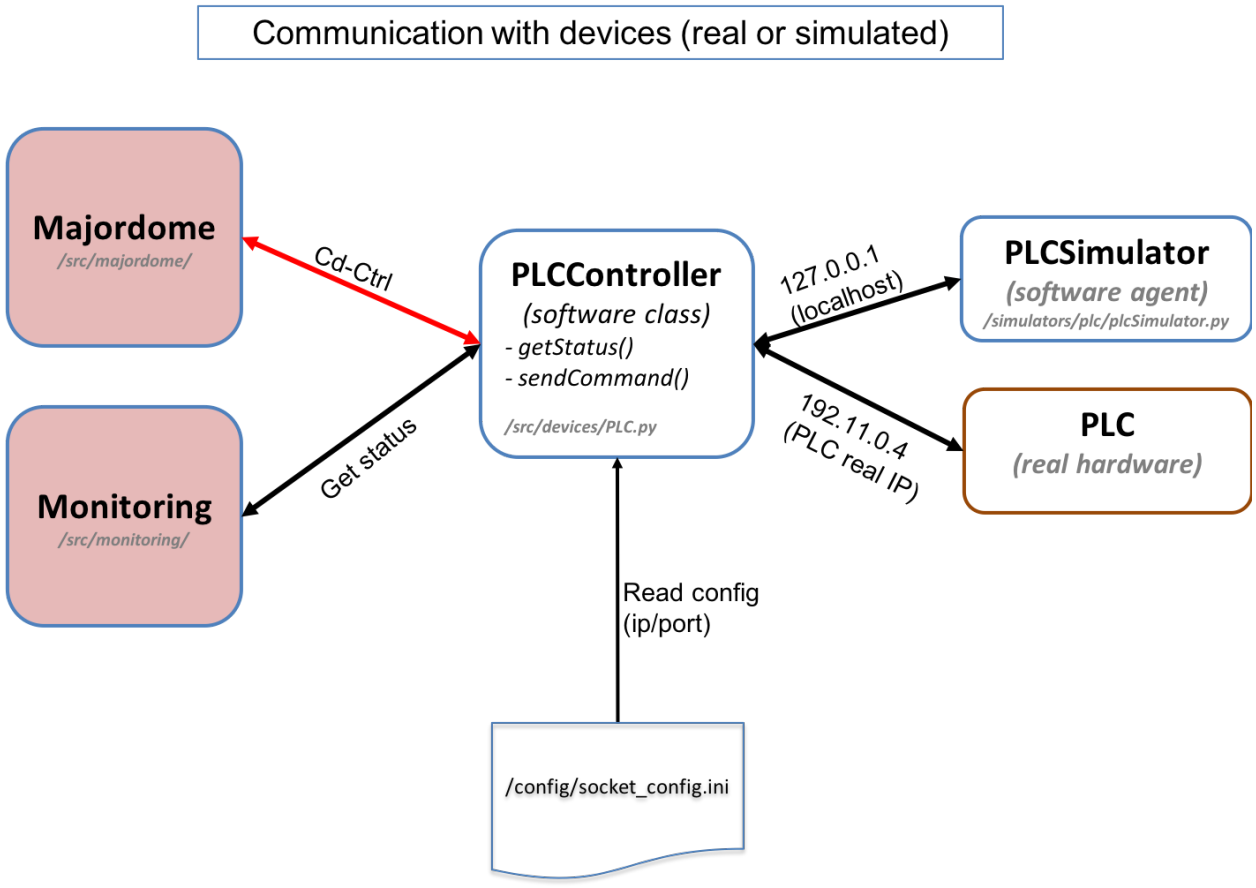
6.1. MODULE “ENVIRONMENT monitoring” (ENV)

(updated 13/03/18)

Responsible : Patrick Maeght

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





6.1.1. Fonctionnement

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")
```

6.1.2. Execution

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****.*

Il suffit donc de 2 terminaux :

(1) - (Agent) Terminal 1 - L'Agent "Environment Monitoring" (ENV)

Voici comment faire pour démarrer cet agent (depuis l'environnement virtuel) :

```
(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()
>>> ...
```

⇒ Pour plus de détails sur cette étape, lire la version "EXECUTION AVEC CELERY" ci-dessous, section "Terminal 1".

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

Dans un 2ème terminal (T2), 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.

⇒ Pour plus de détails sur cette étape, lire la version "EXECUTION AVEC CELERY" ci-dessous, section "Terminal 2".

6.1.3. Execution AVEC CELERY (version complète)

6.1.3.1. - (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)**
⇒ E**n 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`

6.1.3.2. - (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

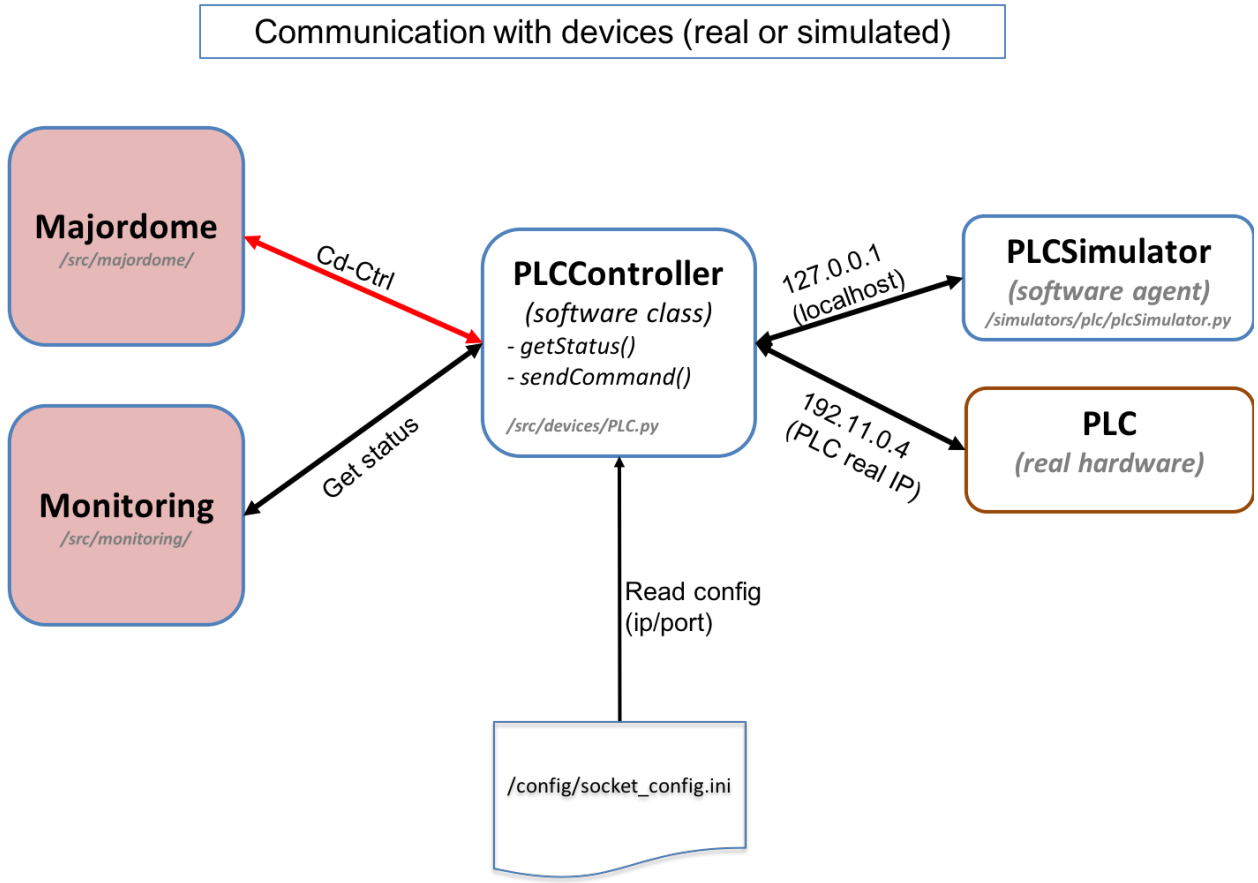
Le **contrôleur du PLC** est un **intermédiaire entre l'agent monitoring et le PLC**, qui permet de **dialoguer avec le PLC** (envoi de commande, et réception de la réponse).
Cette classe **PLCController** est définie dans **src/devices/PLC.py** (TODO: le dossier devices devrait plutot s'appeler device_controllers, 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**

6.1.3.3. - (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_lauch).** 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 génèrera 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": ""}]}]}]
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…)

6.1.4. Que reste-t-il à faire (TODO LIST) ?

(1) - Quelques bugfixes sont encore nécessaires :

- ~~Le log de Monitoring fonctionne bien (cf /logs/Monitoring.log) mais pas les autres utilisés (Devices.log pour DeviceController et DeviceSim.log pour PLCSimulator)~~

(2) - Dans quel sens doit-on faire progresser le module Monitoring ? :

- **Démarrage dans n'importe quel ordre :**
  - Tester que ça marche quand on lance l'agent Monitoring avant le PLC
  - Tester que ça marche quand on lance le PLC avant l'agent Monitoring
- **Tolérance de panne (résilience**) : la communication monitoring-PLC doit continuer à fonctionner même dans ces 3 cas :
  - **(1) ~~Le PLC ne répond plus pendant N secondes, puis répond à nouveau ⇒ c'est à dire, le simulateur est stoppé puis relancé~~**
  - **(2) ~~L'agent monitoring lui-même (task.run) est stoppé et relancé~~**
  - **(TODO: cf mail envoyé à Quentin vendredi soir)**
  - **(3)** Le worker celery dédié au monitoring est stoppé et relancé ⇒ que se passe-t-il dans ce dernier cas ?
- Faire une **page web** (une VUE django, dédiée au monitoring) des 2 tables synthèse du Monitoring, et garder cette page affichée (doit se rafraichir automatiquement) ; actuellement, la seule vue qu'on a de ces tables c'est via l'interface admin de django (localhost:8000/admin) mais c'est pas terrible, il faut une vue unique pour les 2 tables en

même temps (et plus jolie). Cette page devra être accessible depuis le “dashboard” (http://localhost:8000) dans le menu de gauche, avec une entrée “**Environment**” ; on peut aussi imaginer faire évoluer cette page plus tard pour :
    - Qu’elle serve aussi à envoyer des commandes manuelles au PLC (“getStatus” ou autre)
    - qu’elle serve aussi à lancer des événements pour le PLC à la main (genre “il pleut maintenant”...)
    - Qu’elle permette de stopper et relancer manuellement :
        - Le simulateur de PLC
        - L’agent Monitoring
        - Le worker celery
- Ecrire une série de **tests unitaires et fonctionnels automatiques** : **ATTENTION, ces tests ne devront pas venir polluer la BD de production**, mais ils devront écrire dans une autre BD à part (de test), qui n’est pas forcément mysql mais peut être du sqlite par exemple…
- Il faudrait créer un agent “**superagent**” dont le seul rôle est de surveiller TOUS les agents (Monitoring (ENV), Majordome, AlertManager) et de les relancer si besoin

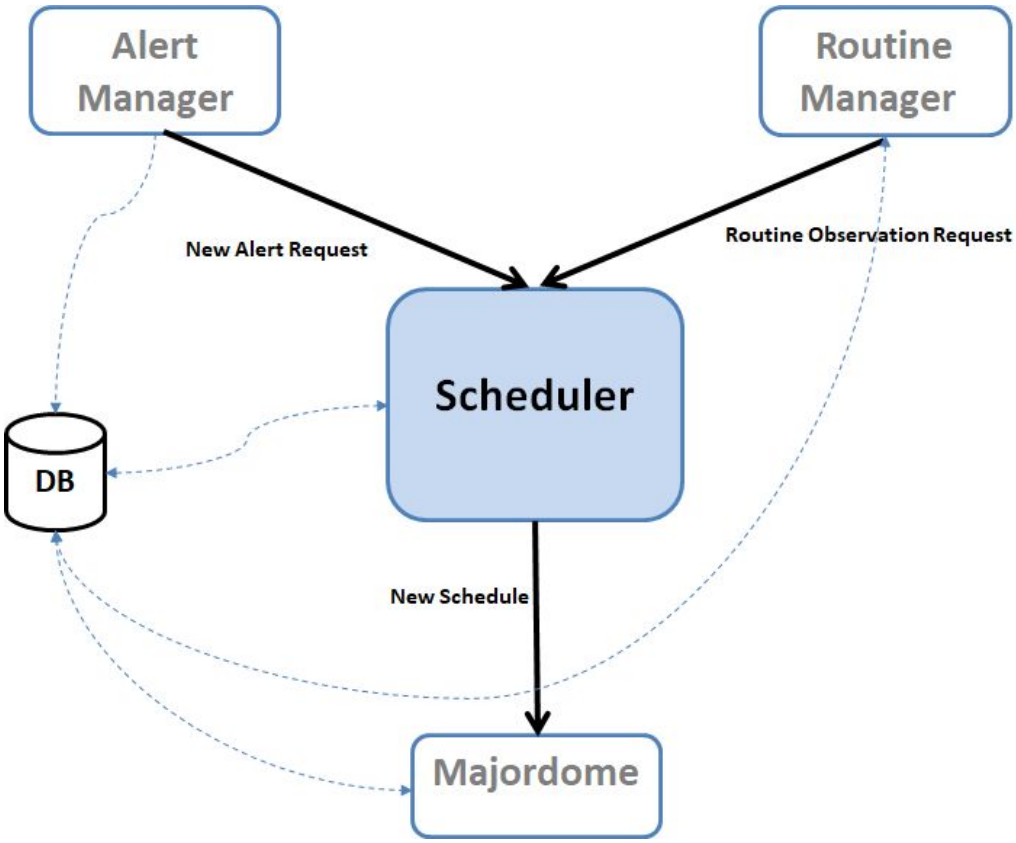
A réfléchir et préciser (avec Alain K) :
Monitoring doit faire une **synthèse intelligente** (à destination du Majordome qui prendra la décision adéquate). Pour chaque élément (pluie, nuages, vent, humidité, ...) faire une synthèse avec 3 valeurs {valeur élement ; interprétation ; décision à prendre}. Par exemple :
Pluie = 5 ; “il pleut fort” ; mise en sécurité = YES
Pluie = 0 ; “il ne pleut pas” ; mise en sécurité = NO
Cloud = 1 ; “un peu nuageux” ; mise en sécurité = NO

6.2. MODULE “SCHEDULER (PLANNER, SCHED)”

(updated 13/03/18)

Responsible : Alain Klotz

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



6.2.1. 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 sour 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

6.2.2. Exécution des tests existants

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

Pour les éxécuter, 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'...
```

6.2.3. 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 sequences pour ces requetes

 # Attention, s'il y a déjà des sequences 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: seq11>, <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
```

6.2.4. 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...

6.3. MODULES “CALIBRATOR & ANALYZER” (TODO)

(TODO:)

6.4. MODULE “MAJORDOME (MAJ)”

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 = NIRCameraController()
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")
```

6.5. MODULE “ALERT MANAGER (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)
```

7. COMMIT howto (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) :
  - `(venv) $ ./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** 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*

8. CODING STYLE

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/

Principe global : **KISS** (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)

- **Indentation**

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** (**Camel Case**) convention
Instances of class should also be **Camel Case** but with lowercase at the beginning
Ex:

```
# class
class MyClass:
    …

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

- **Function names** 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
```

- **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 80 caractères

- **Function size**

Une méthode ou fonction doit être la plus concise possible, et en tous cas elle 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**.

- **Comments**

Pour chaque méthode ou fonction, mettre un commentaire AVANT (triple quote) :

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

ou bien

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

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

```
# TODO: bla bla bla
#s = 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()

- **Signatures 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, end: int) -> str:
   return text[start:end][::-1]
```

- **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()

- **GENERAL RULES**

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

- **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:
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.”
(cf https://hackernoon.com/oo-tricks-the-art-of-command-query-separation-9343e50a3de0)

9. TODO LIST GENERALE

Installer Nagios pour surveiller les différents services…

Faire sans Celery…

Tester Majordome en isolation

10. APPENDICES

10.1. WEB Actions

(updated 28/2/18)

Pour chaque module, on décrit dans ce tableau :

- *s'il est un agent ou pas,*
- *quelles sont ses actions déclenchées via le web (via une page web),*
- *et quelles sont ses actions déclenchées par un autre moyen (non web)*

MODULE	AGENT	WEB ACTIONS	NON-WEB ACTIONS (AGENT)
USER Users management	**NO**	View LOG (users history) Register Login/logout Edit profile	
REQUEST (REQ) Observation Requests management	**NO**	View LOG Observation Request **submission** (via form or XML file) ⇒ call Planner.schedule() View/Edit Request (+ Sequences, Albums & Plans) (CRUD) Get Images (test) Generate "Fake request"	
ALERT GRB Alerts management (SVOM and other)	**YES**	View LOG View alert request (can be done by the REQUEST module above) History : see all past alerts and their outcome	**LOOP** : 1 - Wait for new alert (VOEvent) from alert network (SVOM/FSC or other) (VTP/XMPP protocol)

		(test) Generate “Fake alert”	2 - Process VOEvent and create an Observation Request (save to DB) 3 - Call Planner.schedule()
PLANNER (SCHEDULER?) Sequences planning	**NO** *Just call its method* ***schedule()***	View LOG View current plan (updated real time) View past plans (admin) Manual call to re-schedule()	Schedule (make planning from sequences in DB) ⇒ call Majordome.new_schedule()
ENVIRONMENT (ENV) Environment Monitoring	**YES**	View LOG View current synthesis (environment status for Weather and Site) View past synthesis *(admin) Manual Restart* *(test) Fake alarm (it rains)*	**LOOP**: 1 - Ask PLC for all statuses 2 - Wait for PLC answer 3 - Make synthesis (save to DB)
MAJORDOME (MAJOR) General conductor of the system (scheduler)	**YES**	View LOG View Majordome current status (pause/idle/executing sequence/…) View Majordome current mode (AUTO/MAN) (admin) Change mode (AUTO/MAN) (admin) MANUAL actions (cd-ctrl): - Telescope - Dome - Lights (via PLC) - Power (via PLC) - … View Instruments current status (telescope, detectors)	LOOP: 1 - (every 5 s) Read Environment synthesis (from DB) and take relevant action if necessary 1 - (every 5 s) Read Time (night start/end) and take relevant action if necessary 1 - (every 10 s) Read Instruments status (from DB, **synthesis** done by the OBSERVER module) and take relevant action if necessary 1 - (on new schedule available from Planner) Read new schedule (from DB) and take relevant action 1 - (on Alert CANCEL from ALERT module) Cancel current alert

OBSERVER (EXEC, EYE) Sequence execution	**NO**	View LOG View current status (idle, executing sequence/plan/acquisition...)	**Not necessarily an Agent** : this process can be called by MAJORDOME each time there is a new sequence to be executed (or cancelled) **If Agent** : LOOP: 1 - (every 5 s) Get status from instruments and make synthesis (saved to DB) 1 - (on NEW sequence to be executed, from MAJORDOME) : Execute sequence 1 - (on CANCEL sequence, from MAJORDOME) : Cancel sequence 1 - (on sequence execution finished) : tell MAJORDOME
CONDITIONS (Observing Conditions) Monitoring of the Sky observing conditions	**?**		Asked regularly by MAJORDOME ?
GP1 **1 - Images Correction & Calibration** **2 - Images NRT Analysis (only if alert...)**			**If Agent** : (on new image in folder) : process image **BUT Not necessarily an Agent** : this process can be called by MAJORDOME or OBSERVER each time there is a new image available

10.2. HOW DOES IT WORK (with celery) ?

Les 3 seuls agents du projet sont **Majordome, Monitoring, et AlertManager**

src/pyros/__init__.py contient un code qui démarre automatiquement un des 3 agents (sa méthode run()) :

```
@worker_ready.connect
def start_permanent_tasks(signal, sender):
        if sender.hostname == "pyros@monitoring":
                monitoring.tasks.Monitoring.delay()
        elif sender.hostname == "pyros@majordome":
                majordome.tasks.Majordome.delay()
        elif sender.hostname == "pyros@alert_listener":
                alert_manager.tasks.AlertListener.delay()
```

⇒ Une de ces tâches lancera les 2 autres

1 - Création des "task queues" (Celery workers)

Dans un terminal (T1) :

$ cd PYROS/

$ **python pyros.py start**

⇒ pyros.py : start() ⇒ celery_on() :

⇒ $ **celery** worker -A pyros -Q alert_listener_q -n pyros@**alert_listener** -c 1

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

⇒ $ **celery** worker -A pyros -Q majordome_q -n pyros@**majordome** -c 1

⇒ … etc.

2 - Lancement de Django (et démarrage automatique des 3 agents*)

(seulement si "pyros.py start" a été démarré)*

Dans un AUTRE terminal (T2) :

$ cd src/

$ **python manage.py runserver**

⇒ Cela démarre automatiquement pyros/__init__.py/start_permanent_tasks()

⇒ ce qui démarre un des 3 agents

⇒ ce qui démarre ensuite les 2 autres

10.3. Accéder à la page d'administration de PyROS

Dans un nouveau terminal, activer l'environnement virtuel et lancer le serveur django :

```
(venv) $ cd src/
(venv) $ python manage.py runserver
```

Puis se connecter sur http://localhost:8000/admin
(login 'pyros' / 'DjangoPyros')

10.4. SIMULATORS

Comment sont lancés les simulateurs ?

Exemple pour lancer le PLC simulator :

```
sim = PLCSimulator(sys.argv)
sim.run()
```

Lancement automatique via le script pyros.py :
$ pyros.py **sims_lauch**

```
procs.append(self.execProcessFromVenvAsync(self.venv_bin + " domeSimulator.py " + conf))
procs.append(self.execProcessFromVenvAsync(self.venv_bin + " userSimulator.py " + conf))
procs.append(self.execProcessFromVenvAsync(self.venv_bin + " alertSimulator.py " + conf))
…
```

Chaque simulateur est lancé dans un processus (popen) indépendant et se met en attente d'instruction.

Le fichier de config est lu dans simulators/config/, par défaut c'est conf.json :

```
[
  200,
  {
        "time" : 1,
        "userSimulator" : "routine_request_01.xml"
```

```
},
{
        "time" : 2,
        "userSimulator" : "routine_request_02.xml"
},
{
        "time" : 4,
        "userSimulator" : "routine_request_03.xml"
},
{
        "time" : 5,
        "userSimulator" : "routine_request_04.xml"
},
{
        "time" : 10,
        "userSimulator" : "routine_request_05.xml"
},
{
        "time" : 11,
        "userSimulator" : "routine_request_06.xml"
},
{
        "time" : 15,
        "userSimulator" : "routine_request_07.xml"
},
{
        "time" : 16,
        "userSimulator" : "routine_request_08.xml"
},
{
        "time" : 18,
        "userSimulator" : "routine_request_09.xml"
},
{
        "time" : 33,
        "userSimulator" : "routine_request_10.xml"
},
{
        "time" : 80,
        "plcSimulator" : {"device_name":"WXT520", "value_name":"RainRate", "value":12.0}
},
{
        "time" : 80,
```

```
        "plcSimulator" : {"device_name":"VantagePro", "value_name":"RainRate", "value":12.0}
  },
  {
        "time" : 90,
        "plcSimulator" : {"device_name":"WXT520", "value_name":"RainRate", "value":0.0}
  },
  {
        "time" : 90,
        "plcSimulator" : {"device_name":"VantagePro", "value_name":"RainRate", "value":0.0}
  }
]
```

10.5. CELERY

10.5.1. APPEL DE TACHES CELERY (tasks) dans le projet PyROS

/src/common/RequestBuilder.py :

- validate_request() :
    - scheduler.tasks.scheduling.delay(first_schedule=True, alert=self.request.is_alert)

/src/majordome/tasks.py :

- handleTasks() :
    - monitoring.tasks.Monitoring.apply_async()
    - alert_manager.tasks.AlertListener.apply_async()
- handleNightEndTimer() :
    - observation_manager.tasks.night_calibrations.apply_async()
- handleNightStartTimer():
    - scheduler.tasks.scheduling.apply_async((False, False))
- reset()
    - scheduler.tasks.scheduling.delay((False, False))
- executeSequence()
    - res = observation_manager.tasks.execute_plan_nir.apply_async((plan.id, float(self.getCountdown(shs))))
    - res = observation_manager.tasks.execute_plan_vis.apply_async((plan.id, float(self.getCountdown(shs))))
- systemPause()
    - scheduler.tasks.scheduling.apply_async(first_schedule=False, alert=False)

/src/monitoring/tasks.py :

- handleTasks()
    - majordome.tasks.Majordome.apply_async()
    - alert_manager.tasks.AlertListener.apply_async()

/src/routine_manager/views.py:

- submit_request()
    - scheduler.tasks.scheduling.delay(first_schedule=True, alert=False)

/src/pyros/__init__.py:

- monitoring.tasks.Monitoring.delay()
- majordome.tasks.Majordome.delay()
- alert_manager.tasks.AlertListener.delay()

10.5.2. EXPLICATION CLAIRE

En résumé :

The issue of running async tasks can be easily mapped to the classic **Producer/Consumer** problem. **Producers place jobs in a queue**. **Consumers then check the head of the queue** for awaiting jobs, **pick the first one and execute**

- **Producteur** : quand on met une task (job) dans une queue, on est producteur
    - (ex: **monitoring.tasks.Monitoring.delay**() va déposer la tâche Monitoring.run() dans la queue Monitoring ; le code qui exécute cette ligne est donc un producteur)
- **Consommateur = Worker** (mais le worker peut aussi être producteur)
- **Broker** = le système de Queueing (système de stockage des tasks à exécuter) ⇒ en général **RabbitMQ ou Redis**
- **Results Backend** = le système qui stocke les résultats des tasks (ça peut aussi être pris en charge par le broker)

Dans le détail :

https://www.vinta.com.br/blog/2017/celery-overview-archtecture-and-how-it-works/

10.5.3. BONNES PRATIQUES

http://celerytaskschecklist.com/

10.5.4. CELERY Monitoring (dev only)

Surveillance des workers et tâches

10.5.4.1. - Avec un outil de monitoring dédié

- **Flower :**

Flower is a tool to live monitor Celery tasks. It allows you to inspect which tasks are running and keep track of the executed ones. It's a standalone application and definitely worth using for bigger systems.

Si pas déjà fait à l'installation de pyros (ça devrait l'être avec toute nouvelle installation de pyros) :
(venv) $ pip install flower

Dans un nouveau terminal, lancer la commande suivante :

```
(venv) $ celery flower --basic_auth=admin:admin --address=0.0.0.0 -A pyros
```

Pointer le navigateur à l'url http://localhost:5555/ (login admin/admin), vous aurez une vue en temps réel des évènements Celery.

Tout l'historique étant stocké en mémoire par Flower, c'est une application que l'on n'utilisera qu'en phase de développement.

Installation :

Flower est une micro application web qui permet de monitorer facilement les tâches Celery : https://github.com/mher/flower

Elle fonctionne "out of the box" :

Installation :
$ pip install flower

Exécution :
$ python manage.py celery flower --basic_auth=admin:admin --address=0.0.0.0 -A proj

En pointant votre navigateur à l'url http://localhost:5555/, vous aurez une vue en temps réel des évènements Celery. Tout l'historique étant stocké en mémoire par *Flower*, c'est une **application que l'on n'utilisera qu'en phase de développement**.

- **Django-celery-status :**

django-celerybeat-status will add a page to the Django admin interface where you will be able to see all scheduled tasks along with their next ETA. The crontab API is sometimes confusing and might lead to mistakes. This will give you a little more confidence that you got things right.

- How to make sure your Celery Beat Tasks are working : [django-celerybeat-status]

https://www.vinta.com.br/blog/2017/how-make-sure-celery-beat-tasks-are-working/

10.5.4.2. - Manuellement

To check how many nodes are online run this :

```
$ celery -A pyros status
```

To shut down all running worker nodes:

```
$ celery -A pyros control shutdown
```

Lister les queues actives :

```
$ celery -A pyros inspect active_queues
```

Vider toutes les queues :

```
$ celery -A pyros purge -f
WARNING: This will remove all tasks from queues: alert_listener_q,
analysis_q, create_calibrations_q, execute_plan_nir_q, execute_plan_vis_q,
majordome_q, monitoring_q, night_calibrations_q, scheduling_q.
      There is no undo for this operation!
Purged 3870 messages from 9 known task queues.
```

Vider le contenu d'une queue donnée :

```
$ celery -A pyros amqp queue.purge monitoring_q
-> connecting to amqp://guest:**@localhost:5672//.
-> connected.
ok. 0 messages deleted.
```

Attention toutefois, cela peut ne pas suffire car par défaut, un worker se réserve des tâches :
CELERYD_PREFETCH_MULTIPLIER = 4
Pour lister ces tâches réservées :

```
$ celery -A pyros inspect reserved
```

Pour qu'elles ne soient pas traitées, il va falloir les révoquer une-à-une à la main :

```
$ python manage.py shell
>>> from celery.result import AsyncResult
>>> r = AsyncResult(id)
>>> r.revoke()
```

10.5.5. WITHOUT CELERY ?

Si on veut se passer de celery (ça serait bien d'y arriver...), il faudra soit utiliser des threads, soit faire appel à popen, …soit utiliser un autre système de file d'attente asyncrone :

- RQ est un système "celery-like" mais plus simple (et plus à la mode) : http://python-rq.org

- If you don't want to add some overkill framework to your project (like Celery), you can simply use subprocess.Popen:

```
def my_command(request):
    command = '/my/command/to/run'  # Can even be 'python manage.py somecommand'
    subprocess.Popen(command, shell=True)
    return HttpResponse(status=204)
```

- Autre solution possible :

Django Background Task is a databased-backed work queue for Django, loosely based around Ruby's DelayedJob library.
You decorate functions to create tasks:

```
@background(schedule=60)
def notify_user(user_id):
    # lookup user by id and send them a message
    user = User.objects.get(pk=user_id)
    user.email_user('Here is a notification', 'You have been notified')
```

Though you still need something which schedules those tasks. Some benefits include automatic retries for failed tasks, and setting maximum duration for a running task.
This does involves another dependency but could be useful to some readers without that restriction.

- Autre solution (encore plus simple je crois) : https://pypi.python.org/pypi/django-background-tasks

- Sinon, il y a aussi Django-Channels 2, très prometteur (mais en cours de validation)

10.5.6. CELERY & DJANGO : howto

Pour plus de détail : http://docs.celeryproject.org/en/latest/django/first-steps-with-django.html
Voir aussi : http://nils.hamerlinck.fr/blog/2015/03/27/celery-django/

Référence complète sur Celery : http://docs.celeryproject.org/en/v4.1.0/userguide/index.html

Voir aussi les celery tasks définies dans pyros : https://docs.google.com/spreadsheets/d/15fu0BQm0VYx07qyAl5YiP_OwARTJZdd_JEmK4uoteKU/edit#gid=0

Autres task queues à explorer : https://www.fullstackpython.com/task-queues.html

Bon tuto à essayer : https://github.com/sibtc/django-celery-example

1 - Créer un fichier (module) celery.py dans le dossier du projet django

Ce module a créer une instance de Celery (une “app”).

src/pyros/celery.py :

```
from __future__ import absolute_import, unicode_literals
import os
from celery import Celery

# set the default Django settings module for the 'celery' program.
os.environ.setdefault('DJANGO_SETTINGS_MODULE', 'pyros.settings')

# TODO: normalement, pas necessaire : a virer ?
from django.conf import settings

app = Celery('pyros')

# Using a string here means the worker doesn't have to serialize
# the configuration object to child processes.
# - namespace='CELERY' means all celery-related configuration keys
#   should have a `CELERY_` prefix.
app.config_from_object('django.conf:settings', namespace='CELERY')
# NB: dans pyros, on utilise plutôt ceci (à vérifier si c’est mieux ?) :
#app.config_from_object('django.conf:settings')


# Load task modules from all registered Django app configs.
```

```
app.autodiscover_tasks()
# NB: dans pyros, on utilise plutôt ceci (à vérifier si c'est mieux ?) :
#app.autodiscover_tasks(lambda: settings.INSTALLED_APPS)


@app.task(bind=True)
def debug_task(self):
      print("Request: {0!r}".format(self.request))
```

2 - Now we need to import this app in your proj/proj/__init__.py module.

This ensures that the app is loaded when Django starts so that the @shared_task decorator will use it:

src/pyros/__init__.py:

```
from __future__ import absolute_import, unicode_literals

# This will make sure the app is always imported when
# Django starts so that shared_task will use this app.
from .celery import app as celery_app

__all__ = ['celery_app']
```

3 - Using the @shared_task decorator

The tasks you write will probably live in reusable apps, and reusable apps cannot depend on the project itself, so you also cannot import your app instance directly.
The @shared_task decorator **lets you create tasks without having any concrete app instance** :

demoapp/tasks.py:

```
# Create your tasks here
from __future__ import absolute_import, unicode_literals
from celery import shared_task
```

```
@shared_task
def add(x, y):
    return x + y

@shared_task
def mul(x, y):
    return x * y

@shared_task
def xsum(numbers):
    return sum(numbers)
```

Remarque:
Si on appelle directement une classe Task, c'est la méthode (donc la tâche) run() qui est appelée par défaut (cf http://docs.celeryproject.org/en/latest/reference/celery.app.task.html)
Exemple : depuis majordome/tasks.py, on appelle une tache monitoring :

```
monitoring.tasks.Monitoring.apply_async()
```

`C`'est la classe **Monitoring** qui est appelée. Comme elle hérite de Task, c'est donc sa méthode run() qui sera appelée.

Voir cet article à propos de l'utilisation (déconseillée à moins de vouloir étendre Task) de la classe Task : http://jsatt.com/blog/class-based-celery-tasks/
Attention, dans Celery 4, cette façon de faire est déconseillée ("deprecated") :
http://docs.celeryproject.org/en/latest/whatsnew-4.0.html#the-task-base-class-no-longer-automatically-register-tasks

4 - Starting the worker process

In a production environment you'll want to run the worker in the background as a daemon - see Daemonization - but for testing and development it is useful to be able to start a worker instance by using the **celery worker** manage command, much as you'd use Django's **manage.py runserver**:

```
$ celery -A proj worker -l info
```

(à tester aussi : python manage.py `celery -A proj worker -l info`)

The number of **concurrent tasks** will be the **number of celery worker** you launch.

For a complete listing of the command-line options available, use the help command:

```
$ celery help
```

5 - Default Queue (file d'attente par défaut)

Par défaut, les workers prennent une tâche à exécuter dans la queue "celery", c'est la "default queue", sauf si on précise quelle queue utiliser. Cette queue par défaut ("celery") est modifiable par le paramêtre CELERY_DEFAULT_QUEUE.

On peut bien sûr créer d'autres queues. Par défaut, toute queue non définie explicitement sera automatiquement créée :
CELERY_CREATE_MISSING_QUEUES = True

Lors de la création d'un worker (qui va aussitôt se mettre en attente d'exécution d'une tâche), ce worker sera par défaut à l'écoute de la queue "celery" :

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

Sauf si on lui attribue une queue spécifique (ici "monitoring_q") :

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

L'argument -Q en ligne de commande permet de spécifier à un worker **la ou les** queues qu'il doit écouter.
Ici, tout appel à une tâche définie dans monitoring/tasks.py sera par défaut placé dans la queue "monitoring_q".

Lors de la définition d'une tâche, on pourra préciser dans quelle queue (file d'attente) elle sera insérée par défaut (pour être exécutée dès que possible par un worker disponible ou dédié) :

```
@app.task(queue='test')
def test_task():
    logger.warning("celerrrrrrrrry log")
    print "test"
    return True
```

Lors de l'appel d'une tâche à exécuter, on pourra aussi préciser dans quelle queue cette tâche doit être ajoutée. Par exemple, pour une tâche de monitoring, on peut lancer son exécution ainsi :
monitoring.tasks.Monitoring.delay()
Dans ce cas, c'est une queue par défaut qui sera utilisée.
Mais on pourrait aussi préciser quelle queue utiliser (ici la queue nommée "monitoring"):
monitoring.tasks.Monitoring.apply_async(queue="urgent")

10.6. DJANGO

10.6.1. Conventions

(from https://simpleisbetterthancomplex.com/tips/2018/02/10/django-tip-22-designing-better-models.html)

(see also “Django models field types” : https://docs.djangoproject.com/en/2.0/ref/models/fields/#model-field-types)

(see also “Django coding style” : https://docs.djangoproject.com/en/dev/internals/contributing/writing-code/coding-style)

- **Nom des modèles au singulier et en majuscule. Nom des attributs en minuscule**

Ex:

```
from django.db import models
class Company(models.Model):
    name = models.CharField(max_length=30)
    vat_identification_number = models.CharField(max_length=20)
```

- **Les éléments (lignes) de la table Company : Company.objects**

- **Squelette de modèle préconisé** :

```
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()
```

- **Model related_name**

```
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')
```

Cela permet de faire ceci :

```
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**.

- **Blank and Null Fields**

```
# 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)
```

10.6.2. L’utilitaire manage.py

https://openclassrooms.com/courses/developpez-votre-site-web-avec-le-framework-django/l-utilitaire-manage-py

10.7. EXEMPLES D'EXECUTIONS POSSIBLES (to be continued…)

$ python pyros.py help
1 : install: Launch the server installation
2 : update: Update the server
3 : server: Launch the web server
4 : loaddata: Load the initial fixture in database
5 : clean: clean the repository
6 : clean_logs: clean the log directory
7 : test: launch the server tests
8 : migrate: execute migrations
9 : mysql_on: switch the database to be used to MYSQL
10: mysql_off: switch the database to be used usage to SQLITE
11: makemigrations: create new migrations
12: reset_config: Reset the configuration in settings.py
13: reset_database_sim: Reset the database content
14: help: Help message
15: updatedb: Update the database
16: kill_server: Kill the web server on port 8000
17: init_database: Create a standard context for pyros in db
18: unittest: Runs the tests that don't need celery
19: test_all: Run all the existing tests (this command needs to be updated when tests are added in the project
20: celery_on: Starts celery workers
21: start: Stop the celery workers then the web server
22: stop: stops the celery workers
23: simulator: Launch a simulation
24: simulator_development: Simulation for the scheduler only
25: kill_simulation: kill the simulators / celery workers / web server
26: sims_launch: Launch only the simulators

T1 :
$ python pyros.py sims_launch

T2:

```
$ cd src
$ python manage.py runserver
```

T1:

```
$ celery worker -A pyros -Q alert_listener_q -n pyros@alert_listener -c 1
```

T2:

```
$ python manage.py shell

>>> import alert_manager.tasks
>>> alert_manager.tasks.AlertListener.apply_async()
<AsyncResult: 92eaaebb-b142-49f2-83f0-e0ca3cba1bfa>

>>> import majordome.tasks
>>> majordome.tasks.Majordome.apply_async()
<AsyncResult: 2cb925b2-6a13-47a7-b26c-3ae81e9aa41f>

>>> import monitoring.tasks
>>> monitoring.tasks.Monitoring.apply_async()
<AsyncResult: 12f26411-a258-4586-b111-5eefb96c431e>


>>> import monitoring.tasks as mt
>>> m = mt.Monitoring()
>>> m.run()
…
```