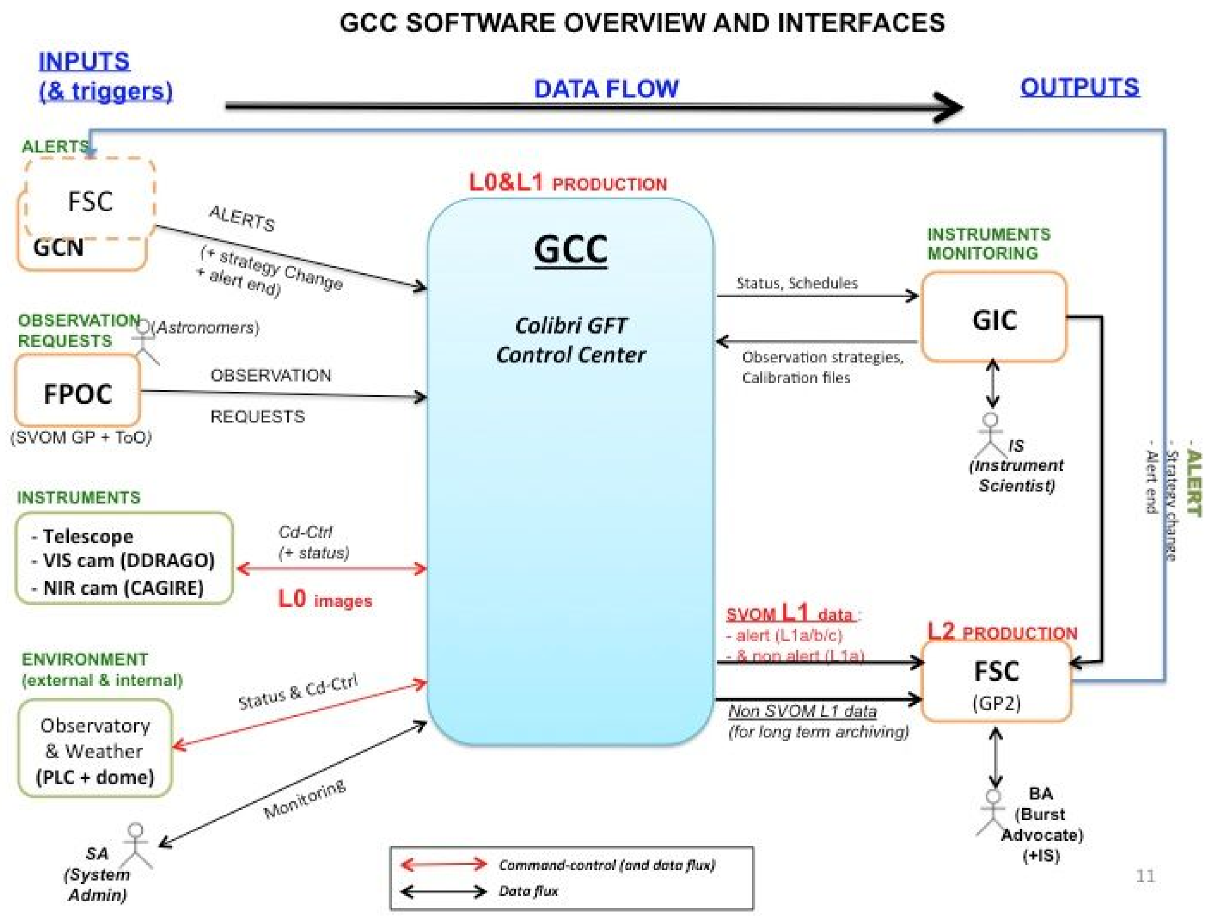
PyROS developer guide

VERSION: 27/01/19

1. General architecture

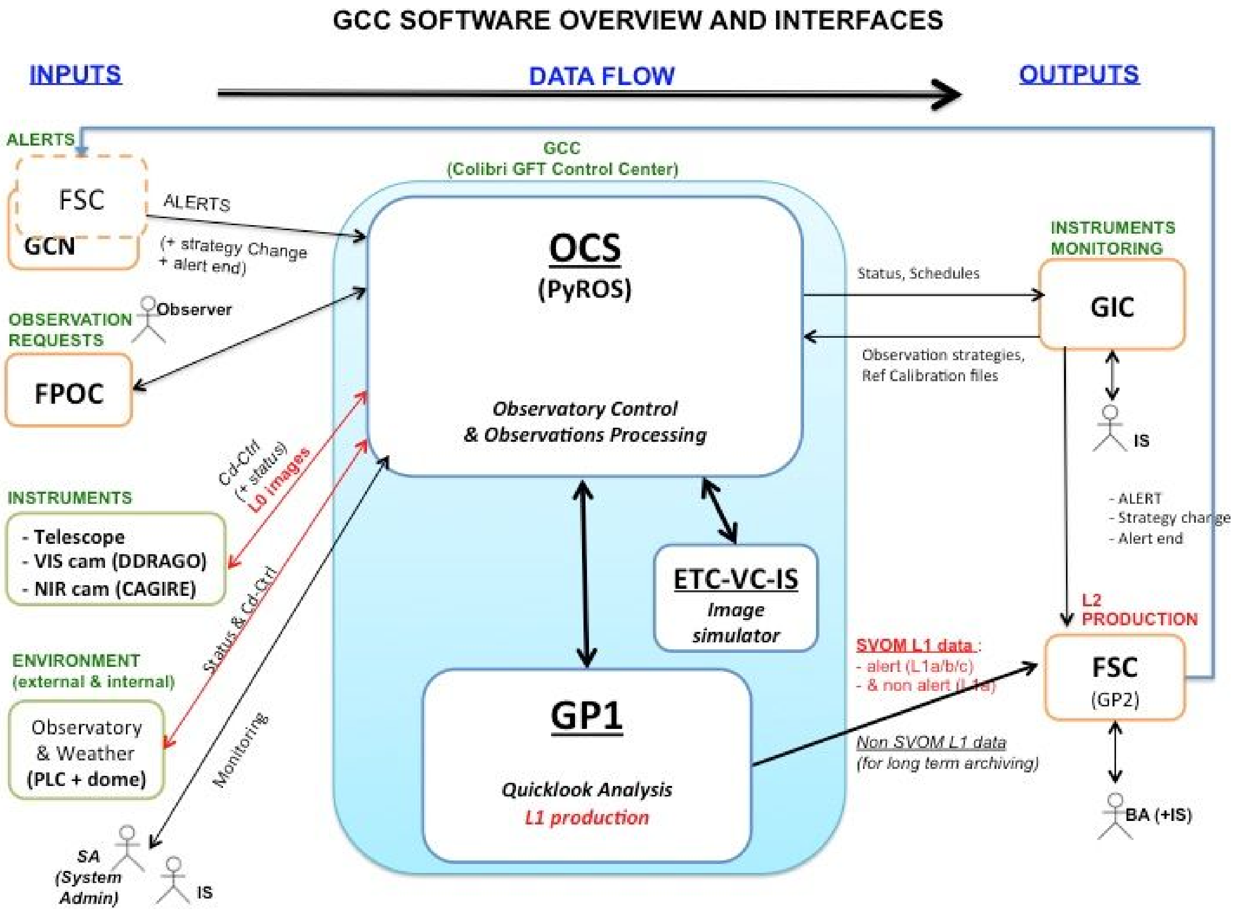
(updated 16/7/18)



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

2. Inside PyROS

(updated 16/7/18)



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

3. Installation of PyROS

(updated 24/1/19, EP)

Pyros needs some prerequisites :

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

3.1. Compatible platforms

(updated 24/1/19, EP)

This software is targeted first for Linux CentOS 7 (+ Fedora and Ubuntu), but also for Mac OS X and Windows 10.
All these systems should run Python **3.6** at least (**3.7** advised)

Pyros has been tested on these platforms:

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

3.2. Installation of Python

See annex to install Python on supported platforms.

3.3. Installation of MySQL server

See annex to install MySQL on supported platforms.

3.4. Installation of Git

Linux:

root $ apt-get install git

Windows:

https://tortoisegit.org/

4. Installation of needed Python packages

4.1. Python packages needed for PyROS

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

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

PYROS/install/REQUIREMENTS.txt

PYROS/install/REQUIREMENTS_WINDOWS.txt

4.2. Get sources of PyROS

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

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

4.2.1. Authenticate to the gitlab

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

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

4.2.2. Get PyROS using Linux

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

```
$ cd MY_DIR
```

4.2.2.1. DYNAMIC VERSION (For Developers) : Get a synchronized version

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

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

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

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

From the terminal :

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

(the first time you get the project, it will ask you for a login and password)
((
you can also provide your login and password directly like this:

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

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

If you ever get this error message :

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

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

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

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

Go into this directory :

```
$ cd PYROS
```

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

```
$ git branch
*  master
```

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

```
$ git checkout dev
$ git branch
* dev
  master
```

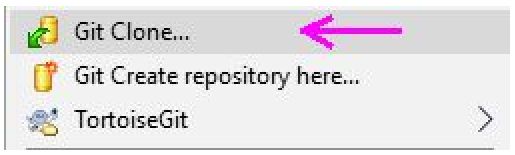
4.2.2.2. STATIC VERSION (NOT FOR developers) : Download a static version (not synchronized and thus NOT RECOMMENDED) :

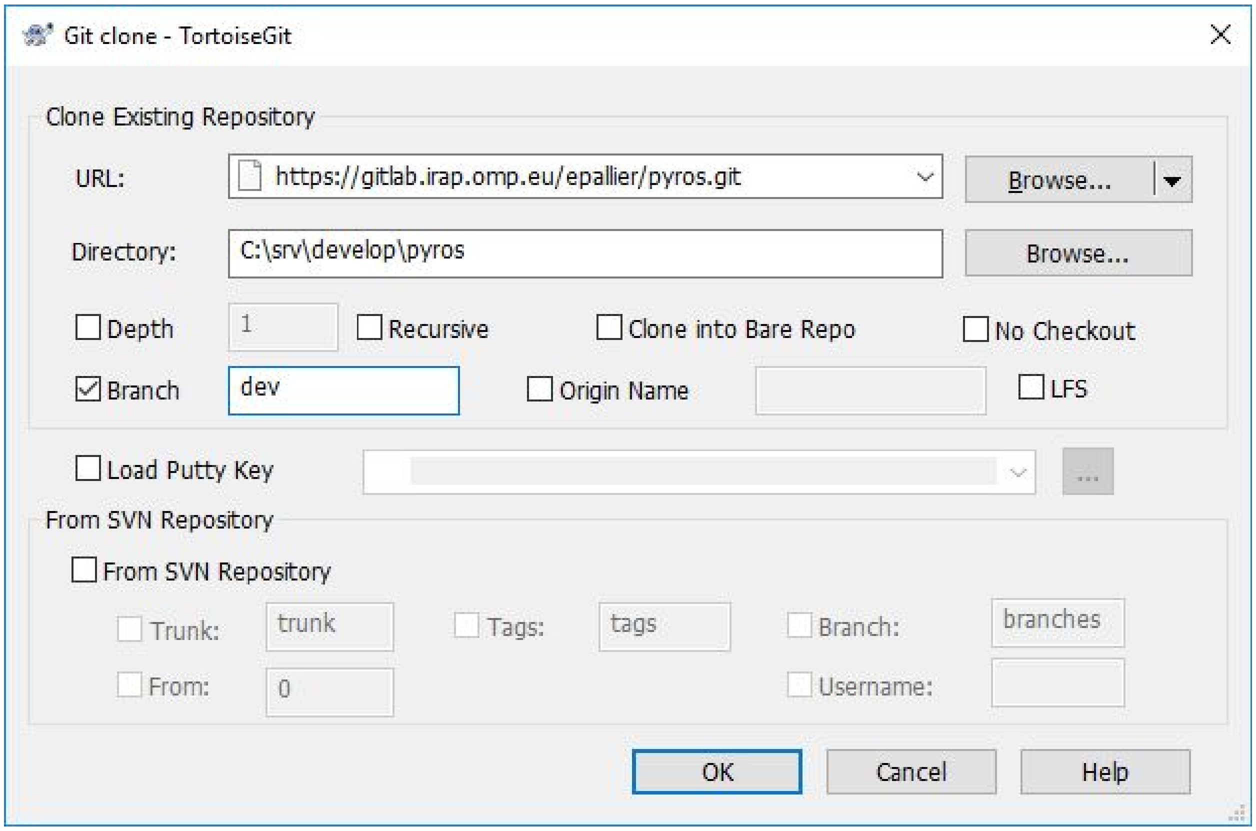
Go to https://gitlab.irap.omp.eu/epallier/pyros/tree/master

Click on "Download zip" on the up right hand corner.
Double-click on it to unzip it.
You should get a "pyros.git" folder.
In this documentation, this software folder will be referenced as "PYROS".
(you can rename "pyros.git" as "PYROS" if you want : "mv pyros.git PYROS")

4.2.3. Get PyROS using Windows

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





URL: https://gitlab.irap.omp.eu/epallier/pyros.git

Directory: C:\srv\develop\pyros

☐ Depth 1 ☐ Recursive ☐ Clone into Bare Repo ☐ No Checkout

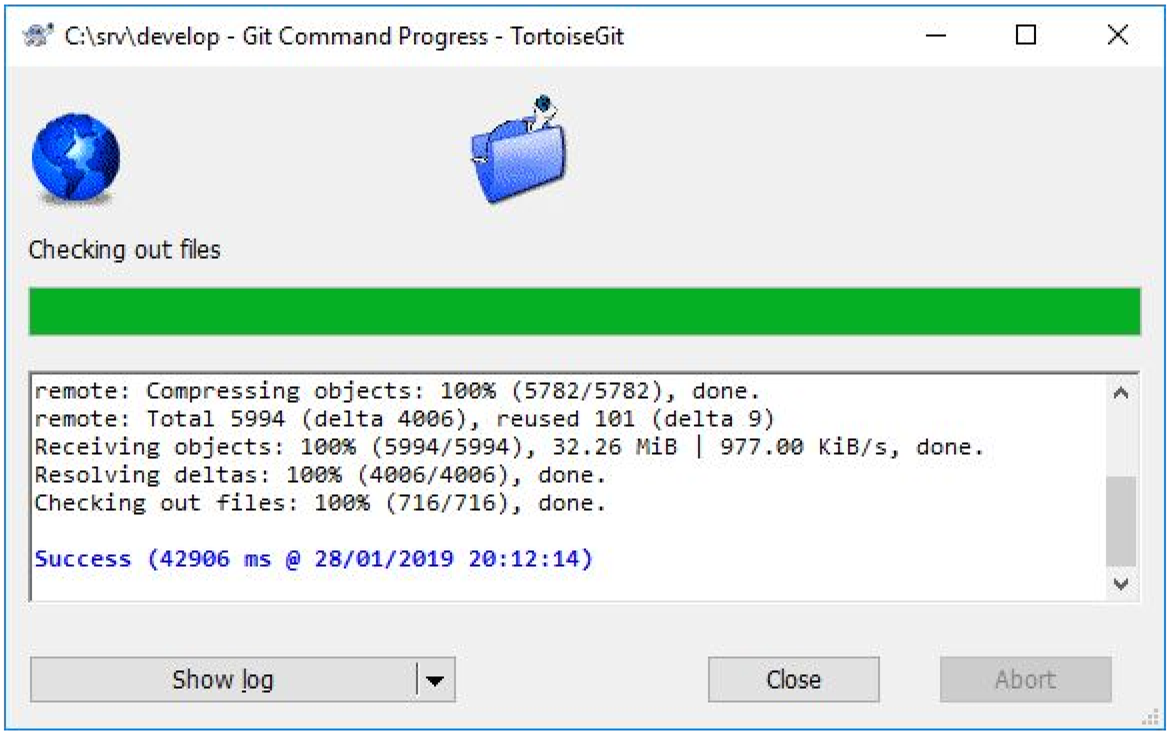
☑ Branch dev ☐ Origin Name ☐ LFS

☐ From SVN Repository

☐ Trunk: trunk ☐ Tags: tags ☐ Branch: branches

☐ From: 0 ☐ Username:

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



Checking out files

```
remote: Compressing objects: 100% (5782/5782), done.
remote: Total 5994 (delta 4006), reused 101 (delta 9)
Receiving objects: 100% (5994/5994), 32.26 MiB | 977.00 KiB/s, done.
Resolving deltas: 100% (4006/4006), done.
Checking out files: 100% (716/716), done.

Success (42906 ms @ 28/01/2019 20:12:14)
```

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

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

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

4.3. Notes about MySql (TBC)

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

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

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

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

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

(Skip this if you are using Sqlite instead of MySql)

5. Structure of PyROS source code

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

6. Deploying PyROS

(updated 12/10/18, EP)

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

6.1. The deployment of PyROS

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

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

Before launching Python, remember the following fact:
Linux and Mac : it is VERY IMPORTANT that you type "python3" and not "python"
Windows : you might need to replace "python" with "py" depending the installation.

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

Linux/Mac:

```
$ python3 -V
```

Windows (Powershell):

```
> python -V
```

Now run the install script:

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

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

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

If it still does not work, try this:

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

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

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

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

6.2. Install the Comet python package (optional)

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

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

Comet is needed as a broker to receive and send VOEvents (https://github.com/jdswinbank/Comet/tree/py3)
You MUST have your virtualenv activated (source venv_py3_pyros/bin/activate in your 'private/' directory)
Documentation is available here : http://comet.readthedocs.io/en/stable/installation.html
(see also http://voevent.readthedocs.io/en/latest/setup.html)

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

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

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

- Ubuntu :

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

- MacOS :

Idem Ubuntu

- Windows :

TODO:

Test Comet

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

All tests should pass

7. Turn on/off the virtual environment

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

Case Linux:

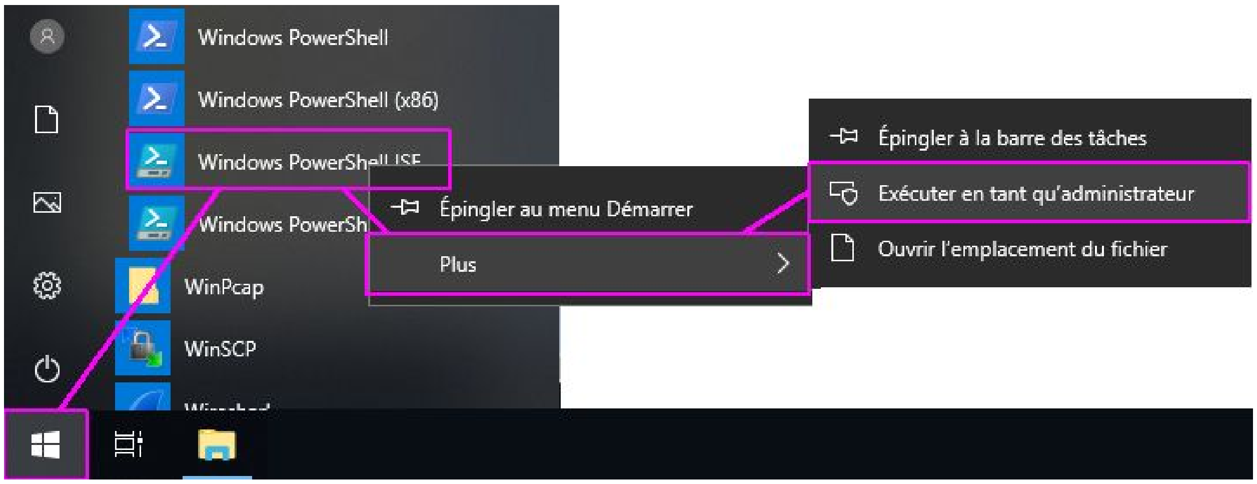
```
$ cd PYROS/
$ source private/venv_py3_pyros/bin/activate
```

Case Windows console (Win+R cmd) :

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

Case Windows powershell:

First time you must unrestrict the execution policy. Launch Powershell in administration mode


Windows PowerShell
Windows PowerShell (x86)
Windows PowerShell ISE
Windows PowerSh
WinPcap
WinSCP
Épingler au menu Démarrer
Plus
Épingler à la barre des tâches
Exécuter en tant qu'administrateur
Ouvrir l'emplacement du fichier

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

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

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

8. Tests

Blabla.

8.1. Philosophy of tests

Blabla.

8.2. Unit tests

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

*When executing the tests, django creates temporarily a **Mysql database** named "**test_pyros**" (see src/pyros/**settings.py**) that it destroys in the end, so you won't ever see it with phpmyadmin (or very quickly and then it will vanish).*
*Some of these tests use the data fixture /src/misc/fixtures/tests/**alert_mgr_test.json***
The scheduler tests do not use any json fixture.

Be sure that at least all unit tests pass:

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

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

If unit tests pass, then try this :

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

(for now, same tests than unittest)

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

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

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

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

NB: if this test does not stop properly, try this:
$ ps -efl|grep agent
$ kill <pid_of_the start_agent_majordome.py process>

Attention, si le test s'est mal passé, vérifier que src/pyros/settings.py contient bien MAJORDOME_TEST = False
(et non pas "True")
Sinon, il faut absolument le remettre à False car sinon pyros ne fonctionnera plus !!
(le test majordome_test.py passe cette variable à True puis la repasse à False à la fin)

Technical information if you are interested (you can skip it, it is only for dev):
Here is how to run the tests for only 1 django app, for instance the user_manager app:

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

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

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

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

8.3. Bigger tests

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

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

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

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

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

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

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

8.3.1. Demo 1 : with only the Users simulator activated

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

(Ctrl-c to stop)
(si pb pour stopper serveur web : $ ps aux | grep runserver)
(si pb pour stopper celery : $ ps aux | grep celery)

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

When you are asked this question (from the terminal) :
"Which simulation do you want to use ? (default=**"conf.json")**
Then just type ENTER so that the scenario “/simulators/config/conf.json” is used for each simulator(s).

Now, access the PyROS website :
go to "http://localhost:8000" in your browser
Log in with login ‘pyros’ (in the Email field) and password ‘DjangoPyros’ to to see what’s happening, and click on **Schedule** to see the current scheduling process.
(You also can click on System, or on Routines and then click on a request to see its detail)

8.3.2. Demo2 : with all simulators activated

```
(first start your venv)
(venv)$ ./pyros.py simulator
```
(Ctrl-c to stop)
(si pb pour stopper serveur web : $ ps aux | grep runserver)
(si pb pour stopper celery : $ ps aux | grep celery)

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

Custom commands (for dev only) *:*

(first, activate your venv)

$ cd src/

$ [./manage.py] test app.tests # Run tests for the application 'app'
Ex:
$ ./manage.py test ***scheduler***
$ ./manage.py test monitoring.tests
$ ./manage.py test routine_manager.tests
$ ./manage.py test alert_manager.tests
$ ./manage.py test common.tests
$ ./manage.py test majordome.tests
$ ./manage.py test user_manager.tests

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

9. Running 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. Programming PyROS

This section describe how to program properly PyROS.

10.1. Access to database

You can access to the database by using the django manage.py script,
Which is equivalent to a "mysql -u pyros -p"...

(the venv must be activated)

```
cd PYROS/src/
./manage.py dbshell
> show tables;
```

10.2. Syntaxe rules and coding style

(updated 14/11/18)

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

- **GENERAL RULES**

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

- **KISS** (Keep It Stupid Simple, https://fr.wikipedia.org/wiki/Principe_KISS) : vous-mêmes ou à plus forte raison quelqu'un d'autre, doit pouvoir relire votre code plusieurs années après et le comprendre rapidement

- **DRY** (Don't repeat yourself)

- **Use exceptions** rather than returning and checking for error states

- **Command/Query Separation : Commands** return void and **Queries** return values (cf https://hackernoon.com/oo-tricks-the-art-of-command-query-separation-9343e50a3de0) ; en d'autres termes : "Functions that change state should not return values and functions that return values should not change state"
  Ex (en langage C):
  ```
  int m();  // query
  void n(); // command
  ```

- **Law of Demeter** (cf https://hackernoon.com/object-oriented-tricks-2-law-of-demeter-4ecc9becad85) LoD tells us that it is a bad idea for single functions to know the entire navigation structure of the system. “Each unit should have only limited knowledge about other units: only units “closely” related to the current unit. Each unit should only talk to its friends; don’t talk to strangers.”

- **Guideline/Coding standard for Django :** https://medium.com/@harishoraon/guide-line-for-django-application-e1a1c075aeed

TEMPLATE DE FONCTION OU METHODE

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

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

def is_connected_to_server(
        item: Any,
        vector: List[float],
        words: Dict[str, int],
        word: Union[int, str],
        server_host: str="localhost",
        server_port: int=11110,
```

```
buffer_size: int=1024,
DEBUG: bool=False,
) -> bool=False:
'''
Return True if we are connected to the server (False by default)

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

- **<u>INDENTATION</u>**

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

- **<u>CLASSES</u>**

Dans la mesure du possible, **1 classe = 1 fichier** du même nom
Ex: vehicule.py ne devrait contenir QUE la classe Vehicule
Class names should normally use the **CapWords** (**CamelCase**) convention
Instances of class should be **snake_case**
Ex:

```
# class
class MyClass:
    …

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

- **<u>FUNCTION</u> and <u>VARIABLE names</u>**

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

Ex:
```
def my_nice_function():
    …
    …

my_nice_variable = 3
```

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

Doivent être au **PLURIEL**:

  - my_item**s**
  - item**s**
  - instance**s**
  - key**s**
  - value**s**
  - ...

- **TODO**

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

- **CONSTANTES**

LIMIT_MAX
LIMIT_MIN
STATICFILES_DIRS

- **LINE SIZE**

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

- **FUNCTION SIZE**

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

- **COMMENTAIRES**

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

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

ou bien

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

...

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

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

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

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

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

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

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

    _my_private_attribute
    _my_private_method()

- **SIGNATURE DES MÉTHODES**

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

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

10.3. Integrated tests of the PyROS code

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

10.3.1. Utilité des tests

Ceinture de sécurité pour s’assurer contre toute **régression** (ce qui marchait avant doit continuer de marcher COMME avant) ⇒ on a ainsi beaucoup moins peur de modifier le code
⇒ A exécuter après chaque modif de code et surtout <u>AVANT tout commit</u> avec git
⇒ Tendre vers l’approche TDD : écrire le test AVANT la fonctionnalité à tester

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

10.3.2. Différents types de test

- **Tests unitaires** (test des méthodes d'une classe) :
  test_<nom-du-test>_<nom-de-la-methode-testée>()
  ex: 3 tests unitaires de la méthode "add_item()" d'une classe:
    - test_first_case_add_item()
    - test_second_case_add_item()
    - test_third_case_add_item()

- **Tests fonctionnels (et d'intégration)** (test des fonctionnalités du logiciel, surtout celles demandées dans les specs ; font intervenir plusieurs classes d'un même module, voire même plusieurs modules) :
  test_func_<nom-de-la-fonctionnalité>()
  ex: test_func_alert_complete_processing()

- **Tests de performance** :
  test_perf_<nom-du-composant>()
  ex: test_perf_scheduler()

- **Tests de robustesse** (stress test) :
  test_stress_<nom-du-composant>()
  ex: test_stress_scheduler()

10.3.3. Organisation des tests dans le contexte de Django

Dans le contexte Django, les tests sont organisés par "APP" (application ou "module").
Exemples:

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

10.3.4. Exécution des tests

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

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

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

Depuis cet environnement virtuel, on exécute les tests ainsi:

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

Remarques:

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

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

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

Exécution des tests SANS django:

```
$ python -m unittest test_module1 test_module2
$ python -m unittest test_module.TestClass
$ python -m unittest test_module.TestClass.test_method
```

Plus de détail sur unittest: https://docs.python.org/3/library/unittest.html

10.3.5. Structure d’un fichier tests.py

A savoir:

- **Un test ne doit tester qu’UNE chose**
- Un fichier **tests.py** peut contenir **PLUSIEURS tests** (une fonction **test_xxx**() par test)
- Chaque test test_xxx() du fichier tests.py est exécuté **INDÉPENDAMMENT** des autres et ces tests doivent donc pouvoir être exécutés dans n’importe quel ordre. Un test ne doit pas dépendre d’un autre test. La BD de test est réinitialisée après **chaque** test.
- Le fichier tests.py contient une fonction **setUp**() qui est **appelée AVANT chaque test** et crée la **fixture** (les données initiales et les tables en BD) nécessaire à l’exécution du prochain test
- Le fichier tests.py contient une fonction **tearDown**() qui est **appelée APRES chaque test** et peut servir à faire le ménage après un test (rarement utile)
- **Sans Django**, il peut être utile de placer ceci à la fin du fichier tests.py:

```
if __name__ == "__main__":
    unittest.main()
```

10.3.6. Exemple de fichier tests.py

(inspiré de src/scheduler/tests.py):

```
# Import de la classe de Test
from django.test import TestCase
(SANS django: from unittest import TestCase)

# Import de la classe représentant le module à tester
from scheduler.Scheduler import Scheduler

# Import de la description de la BD (modèles django) ssi besoin d’interagir avec la BD
# Attention: Django utilise automatiquement la BD de test (et non pas la BD de prod)
# Cette BD sera détruite après CHAQUE test
from common.models import *

# Classe AppTest: doit hériter de TestCase (PEP8: classe écrite en camel-case)
class SchedulerTest(TestCase):

    ‘’’
    Fixture initiale (Initialisation appelée AVANT CHAQUE FONCTION test_xxx())
    Autres fonctions de setup :
        -   setUpClass() ⇒ exécutée au début de CHAQUE CLASSE
        -   setUpModule() ⇒ exécutée une seule fois au tout début
    ‘’’
    def setUp(self):
        self.scheduler = Scheduler()
        self.scheduler.max_overhead = 1
        scipro = ScientificProgram.objects.create()
        country = Country.objects.create()
        user_level = UserLevel.objects.create()
        self.usr1 = PyrosUser.objects.create(username="toto", country=country,
user_level=user_level, quota=100)
        self.req1 = Request.objects.create(pyros_user=self.usr1,
scientific_program=scipro)
        ...


    # Un premier test (Attention, PEP8 préconise l’utilisation du snake-case)
    def test_basic_1(self):
        '''
        Goal : test if 3 distinct sequences are put into the planning
        Pre-conditions : the sequence have the same priority, and have no jd overlap
        '''
        self.scheduler.schedule.plan_start = 0
        self.scheduler.schedule.plan_end = 10
        …
        # Assertion de test : nb_planned doit etre egal à 3:
        # (Voir toutes les assertions possibles ⇒ ICI)
        self.assertEqual(nb_planned, 3)
        self.assertEqual(shs1.tsp, 1)
        ...

    # Un second test (Attention, PEP8 préconise l'utilisation du snake-case)
    def test_basic_2(self):
        '''
        Goal : ...
        Pre-conditions : ...
        '''
        …
        # Assertion de test
        # (Voir toutes les assertions possibles ⇒ ICI)
        self.assertEqual(a, b)
        ...


    '''
    Ménage final (fonction appelée APRES CHAQUE FONCTION test_xxx())
    Autres fonctions de ménage :
        -   tearDownClass() ⇒ exécutée à la fin de CHAQUE CLASSE
        -   tearDownModule() ⇒ exécutée une seule fois à la fin
    '''
    def tearDown(self):
        ...


# A rajouter éventuellement tout à la fin (uniquement si on n'utilise pas Django):
if __name__ == "__main__":
    unittest.main()
```

10.4. Documentation inside PyROS code

Blabla.

10.5. Git Managing

10.5.1. Commit procedure

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

- Mettre à jour le fichier **README du module** modifié (/src/module/README) : version, date, auteur, … (chaque module doit être “pensé” comme une application indépendante)

- Mettre à jour le fichier **README général** du projet (/README) : version, date, auteur, ...

- S’assurer que le **style de codage** est bien respecté (cf CODING STYLE)

- S’assurer que tous les **tests** passent toujours (cf TEST) :
  - `$ ./pyros.py test`

- **Mettre à jour votre code** (quelqu’un pourrait avoir fait des modifs avant vous !) :
  - Vérifier que vous êtes bien sur la branche "dev" :
    ```
    $ cd PYROS/
    $ git branch
    * dev
      master
    ```
  - (Si ce n’est pas déjà le cas, aller sur la branche dev) :
    `$ git checkout dev`
  - Mettre à jour votre copie :
    `$ git pull`

Maintenant, vous êtes prêts pour envoyer vos changements :

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

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

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

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

10.5.2. Pull procedure

$ git pull

If changes have been made on the database, you will also need to run this, **from your venv** (no fear for your database, it will not initialize it but just add some necessary data like fixtures):
(venv)$ ./pyros.py init_database

10.6. Agent Mount

Blabla.

10.7. Agent Camera

Blabla.

10.8. Agent Monitoring

Blabla.

11. ANNEXES

11.1. Annex 1: Using Git

Blabla.

11.2. Annex 2: Python installation for specific operating systems

****

IMPORTANT FOR LINUX USERS:
Even if your python is up to date, be sure to do at least the different installations IN RED BOLD below
(for instance, on linux CentOS, it is necessary to do at least "sudo yum install python34-devel" and "sudo pip install --upgrade pip", ...)
****

11.2.1. A2.1. Linux CentOS 7.1

(python35 not yet available as rpm ?)

```
$ sudo yum update yum
$ sudo yum update kernel
$ sudo yum update
$ sudo yum install yum-utils
$ sudo yum groupinstall development
$ sudo yum install https://centos7.iuscommunity.org/ius-release.rpm
$ sudo yum install python34

$ python3.4 -V
Python 3.4.3
```

$ **sudo yum install python34-devel**
(needed for python package mysqlclient)

((
NO MORE NECESSARY:

$ sudo yum update python-setuptools
$ easy_install --version
setuptools 0.9.8
$ sudo easy_install pip
$ pip --version
pip 8.1.1 from /usr/lib/python2.7/site-packages/pip-8.1.1-py2.7.egg (python 2.7)
))

$ **sudo pip install --upgrade pip**

Necessary for "lxml" python package:
$ **sudo yum install libxml2 libxml2-devel**
$ **sudo yum install libxslt libxslt-devel**

11.2.2. A2.2. Linux Ubuntu, Suse

$ sudo add-apt-repository ppa:fkrull/deadsnakes
$ sudo apt-get update
$ sudo apt-get install python3.5
$ sudo apt-get install python3.5-dev (or python3.6-dev) if you're using python 3.6
(needed for python package mysqlclient && lxml)
$ sudo apt-get install libxml2-dev
$ sudo apt-get install libxslt-dev
$ sudo apt-get install zlib1g-dev can be required too
$ sudo apt-get install python-pip
$ sudo apt-get install python-lxml

((
NO MORE NECESSARY
$ sudo pip install --upgrade virtualenv
))

11.2.3. A2.3. Mac OS X

- **From Brew (recommended)**

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

Install HomeBrew :
(TODO)
```

```
Install Python3 :
$ brew doctor
$ brew update
$ brew upgrade
$ brew install python3
$ brew info python3
$ python3 -V
Python 3.6.4
$ which python3
/usr/local/bin/python3
```

- **From MacPort**

Install macport :
https://www.macports.org/install.php

Install the "port" python36

```
$ sudo port install python36
$ sudo port select --set python3 python36
$ sudo port install py36-readline
$ sudo port install py36-pip
$ port select --set pip pip36
```

11.2.4. A2.4. Windows 10

Go to https://www.python.org/downloads/windows/ , choose the wanted version
On the wanted version’s page, download Windows x86 executable installer

Run the executable :
 * On the first page, check "Add python3.5 to PATH"
 * Choose "Install now" option

Open cmd (windows + R, cmd) :
 $ python -m pip install - - upgrade pip

If ever the “python” command does not work, it means that you have to add it to your PATH :
- Type key WINDOWS + PAUSE
- Click on “Paramètres système avancés”
- Click on button ‘Variables d’environnement’
- Click on “Variables système“

- Select line « Path »
- Click on “modify”,
- Click on “New”
- Add the path to your python.exe executable (or the Anaconda folder)
- Click on “OK”

Now, test that python can be executed:
- Open a new command window : WINDOWS + R
- type « cmd »
- type « python »

11.3. Annex 3: MySQL installation for specific operating systems

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

For more information, see section “Notes about Mysql”

11.3.1. A3.1. Linux CentOS

cf https://www.howtoforge.com/apache_php_mysql_on_centos_7_lamp#-installing-mysql-

First, update your system:

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

$ sudo yum install mariadb-server
$ sudo yum install mariadb

$ sudo yum install mariadb-devel
```

(needed for python package mysqlclient)

```
$ sudo systemctl start mariadb.service

$ sudo systemctl enable mariadb.service
=> Created symlink from /etc/systemd/system/multi-user.target.wants/mariadb.service to
/usr/lib/systemd/system/mariadb.service.

$ sudo mysql_secure_installation
```

11.3.2. A3.2. Linux Ubuntu, Suse

First, update your system:
$ sudo apt-get update

$ sudo apt-get install mysql-server
$ sudo apt-get install mysql-client

$ sudo apt-get install libmysqlclient-dev
(needed for python package mysqlclient)

To resolve auth problems for root user check this link
https://unix.stackexchange.com/questions/396738/cant-access-mysql-without-running-sudo

11.3.3. A3.3. Mac OS X

Install MySql with brew (recommended) or macport, or install XAMPP (https://www.apachefriends.org/fr/index.html)

- With brew (recommended) :

Tested with Mysql 5.7.21

```
$ brew doctor
$ brew update
$ brew upgrade
$ brew install mysql
$ mysql -V
```

Now, start the Mysql server :

```
$ mysql.server start
```

Now, connect to the Mysql server with the mysql client :

```
$ mysql -u root
mysql> exit
```

11.3.4. A3.4. Windows 10

Download and install the newest version on https://dev.mysql.com/downloads/installer/

Once installed, launch MySQL Installer.
Click on 'Add...' on the right.
In MySQLServers section, choose the newest, then click on NEXT.
Install and configure the server (just follow the installation guide).

Then launch mysql (via the Windows menu).

11.3.4.1. Installation of PHP to use phpmyadmin

Install PHP5 in C:/php.

- Copy c:/php/libmysql.dll into c:/windows/system32
- Copy c:/php/php.ini-recommended as c:/windows/php.ini

Edit the file c:/windows/php.ini, and change it as follows:

```
...
; Directory in which the loadable extensions (modules) reside.
extension_dir ="c:/php/ext/"
...
; Dynamic Extensions ;
...
extension=php_mbstring.dll
extension=php_mysqli.dll
extension=php_mysql.dll
```

Do not forget the i in the name php_mysqli.dll. php_mysql.dll is important for PHP scripts.
We may test the operation of PHP with Apache by writing the script info.php containing the following line:

```
<?PHP phpinfo(); ?>
```

We will put the script info.php in the Apache htdocs folder.

It is sometimes possible that PHP cannot find the file php.ini in the c:/windows folder. We must then leave it in the c:/php folder and append the following line in httpd.conf for Apache, just after the line *LoadModule php5_module* :

```
PHPIniDir "C:/php/"
```

11.3.4.2. Install phpmyadmin

Unzip PHPMyAdmin in htdocs/, and rename the folder htdocs/PHP... to htdocs/phpmyadmin.
Copy phpmyadmin/config.sample.inc.php as phpmyadmin/config.inc.php, then edit the file phpmyadmin/config.inc.php and modify it as follows:

```
     ...
     $cfg['blowfish_secret']           = 'php my admin';
     $cfg['Servers'][$i]['extension'] = 'mysql';
     $cfg['Servers'][$i]['user']       = 'root'; // MySQL user
     $cfg['Servers'][$i]['password']   = 'PASSW_mysql_root'; // MySQL
password
```

11.4. Annex 4: Python packages installation for specific operating systems

See the files REQUIREMENTS*.txt in the folder PYROS/install

11.5. Annex 5: Notes for Eclipse IDE users

1) Install Eclipse (if necessary) and the PyDev plugin

Install Eclipse

(optional, can be done later) Install the plug-in PyDev (via install new software, add http://pydev.org/updates)

How to configure PyDEV :

- General doc : http://www.pydev.org
- For Django : http://www.pydev.org/manual_adv_django.html

2) Import the PYROS project

a) If **PYROS is already on your file system** (cloned with git from the terminal, see section above)

Just import your PYROS project from the file system :
File Import / Existing projects into workspace / Next
Select root directory : click on “Browse” and select your PYROS directory
Click on “Finish”

b) If **PYROS is not yet on your file system** (not yet cloned with git)

You must clone the PYROS project with git from Eclipse :
File/Import project from git
Select repository source: Clone URI: https://gitlab.irap.omp.eu/epallier/pyros.git
Directory:
par défaut, il propose : /Users/epallier/git/pyros
mais on peut le mettre ailleurs (c'est ce que j'ai fait)
initial branch: master

remote name: origin
Import as general project
Project name: PYROS
If necessary, to deactivate CA certificate verification
Window -> Preferences -> Team -> git -> configuration -> Add entry
Key = http.sslVerify
Value = false

Si le plugin PyDev n'est pas encore installé, voici un truc simple pour le faire :
Ouvrir un fichier python
Eclipse propose automatiquement d'installer PyDev

Switch to the DEV branch :
Right-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/

11.6. Annex 6: Notes for PyCharm IDE users

Make the following points:

1) Install Pycharm

2) import pyros project

3) Mark the src directory and simulators directory as source root directories

4) Go in file -> settings (CTRL + ALT + S) -> Project : Pyros -> Project Interpreter
Add an interpreter which is the one from your virtual environment : Add Local -> find the python 3 binary in your virtualenv

5)
For professional version :
Go in Language & Frameworks -> Django and set the django project root / Settings (pyros/settings.py) / Manage script

For community edition :
First: Go to edit configuration (top right corner)
Second: Click on the (+) mark in top-left corner and add python configuration.
Third: Click on the Script, and for django select the manage.py which resides on the project directory.
Fourth: Add <your command> as Scripts parameter and click apply : you normally should be able to run your project

11.7. Annex 7: Mount a virtual environment

Blabla.

11.8. Annex 8: Functions of PyROS

Blabla.

11.9. Annex 9: Coding Python style

Blabla.

12. TODO LIST

- (EP 5/2/19) ajouter **pydbg** dans requirements pour faciliter le debug (https://github.com/tylerwince/pydbg) :
    - pip install pydbg
    - from pydbg import dbg

- (EP 5/2/19) ./**pyros2 (ex pyros.py) init_database**() : ne doit plus contenir la fonction init_database() car elle n'est utilisée que par le script install.py => move cette fonction to install.py (elle sera utilisée à la fois pour une install normale et un update)

- (EP 5/2/19) ./**install.py update** : remettre en place
    - OLD => C'était utilisé dans install_old.py (JB) : update faisait seulement "pyros.py init_database", ce qui faisait :
        - Update DB : makemigrations + migrate
        - loaddata() : chargement de la fixture initiale src/misc/fixtures/initial_fixture.json
    - NEW => il faudrait refaire ça (maybe sans le loaddata() ? mais c'est peut-être utile à garder...)
    - Update DOC : une section UPDATE qui dit : pour se mettre à jour, faire :
        - 1) git pull
        - 2) pyros2 update (qui appellera "install.py update" qui appellera init_database)

- (EP 5/2/19) **bugfix install.py** ne marche pas avec mysql 5.5 sur linux (hyp2) :

```
----------------------------Launching mysql to create database and create and grant user
pyros----------------------------
MySQL version is 5.5
----------------------------Please enter your MYSQL root password----------------------------
Enter password:
ERROR 1064 (42000) at line 1: You have an error in your SQL syntax; check the manual that
corresponds to your MariaDB server version for the right syntax to use near 'IF NOT EXISTS
pyros' at line 1
Traceback (most recent call last):
  File "install.py", line 364, in <module>
        if INSTALL_DB: install_database(VENV)
  File "install.py", line 314, in install_database
        stderr.write(Colors.ERROR + "ERROR !: db configuration failed !" + Colors.END + "\r\n")
NameError: name 'stderr' is not defined
-----------------------------------Process execution failed-----------------------------------
```

- (EP 1/2/19) Utiliser **black** pour le style du code :
  - https://github.com/ambv/black
  - https://www.developpez.com/actu/207399/Black-l-outil-de-formatage-de-code-Python-transforme-les-guillemets-droits-simples-en-guillemets-doubles-les-auteurs-expliquent-leurs-choix/

- (EP 1/2/19) **Voeventparse** Windows : comment installer automatiquement via pip (et non pas manuellement comme c'est le cas actuellement) ?

- (EP 1/2/19) **Bugfix start_monitoring**

- (EP 1/2/19) Faire évoluer le script pyros2.py pour :
  - pyros2 **--configfile** <config-file-name> **start** <agent-name> => on peut avoir plusieurs fichiers de config différents (chaque fichier utilise un simulateur ou unit différent)
  - Parser XML AK:
    - Cd config/
    - ./test_config_xml1.py

- (EP 1/2/19) Exceptions au lieu de return (errno, return_value)

- (EP 1/2/19) Move TODOLIST dans Redmine

- (AK 1/2/19) Requirements Windows installés manuellement => à éviter
  - ~~Package lxml~~
  - Package voeventparse : décommenter code d'install manuel de voeventparse dans install.py et décommenter ligne voeventparse dans REQUIREMENTS_WINDOWS pour que voeventparse soit installé automatiquement

- (EP 29/1/19) Utiliser **pyenv** pour l'installation de python et **pipenv** pour la gestion des packages (pip) et requirements (Pipenv will automatically convert your old requirements.txt into a Pipfile)
  - ⇒ https://hackernoon.com/reaching-python-development-nirvana-bb5692adf30c
  - Gérer aussi les dev-req et stable-req

- (EP 28/1/19) Résoudre le pb avec la BD test_pyros lors de l'exécution des tests (pyros.py unittest) :
  - "access denied for user pyros@localhost"
  - Faut-il la créer avec le script d'installation ou bien laisser django le faire au moment de l'exécution des tests ? est-ce que ça marche bien avec un vieux sql ? sur Windows ?

- (EP 28/1/19) Faire un agent générique AgentX qui hérite de Agent

- (EP 27/1/19) : Divers

    - INFO:
        - Samuel Ronayette : nouveau sur AIT/AIV
        - UNIT = 1 telescope, mais en fait 1 coupole = 1 monture (mount)
        - 1 fichier de config XML = 1 unit
        - CHANNEL = une voie d'instrumentation
        - 1 UNIT est attaché à un toit roulant (mais il peut y avoir plusieurs unit sous un même toit roulant)
        - NODE = plusieurs units
        - Telescopes :
            - Valencia 50cm
            - Chinois Tango (Tibet) 50cm

    - Isoler les paramètres d'un UNIT (telescope) en dehors du projet pyros :
        - pyros_unit_taca/ => une unit
        - Depuis pyros.settings, aller chercher la conf xml de la unit

    - return (errno, result, […])

    - Faire un script pour update (git pull + DB sync)
        - Faire un pyros « update »:
        - Devra inclure$ git -c http.sslVerify false pull

    - Fichier config pour install
        - WIN: Trouver un moyen de trouver le chemin de mysql automatiquement (ou à la façon de « ros install » en mode graphique ou en mode console, et écrire le fichier de config correspondant aux réponses données pour ne pas avoir à les redonner lors d'une 2ème install (car les propositions par défaut seront ok)

    - Update django (on utilise 2.0.5, mais la version stable courante est 2.1.5)
        - (12/19) **Django 3.0**: https://docs.djangoproject.com/en/dev/releases/3.0/
            - Python 3.6+

        - (04/19) **Django 2.2** (**LTS**) release notes : https://www.djangoproject.com/weblog/2019/jan/17/django-22-alpha-1/
            - Python 3.5+

        - Constraints: The new CheckConstraint and UniqueConstraint classes enable adding custom database constraints. Constraints are added to models using the Meta.constraints option.
        -

    - (08/18) **Django 2.1** release notes : https://docs.djangoproject.com/en/2.1/releases/2.1/
        - Python 3.5+
        - Model “view” permission: A “view” permission is added to the model Meta.default_permissions. The new permissions will be created automatically when running migrate. This allows giving users read-only access to models in the admin. ModelAdmin.has_view_permission() is new. The implementation is backwards compatible in that there isn’t a need to assign the “view” permission to allow users who have the “change” permission to edit objects.
        - jQuery is upgraded from version 2.2.3 to 3.3.1