



Common-EGSE

Installation Manual

Sara Regibo, Rik Huygen

Version 1.3, 09/02/2023

Table of Contents

Changelog	iv
Colophon	1
Conventions used in this Book	2
1. Introduction	3
2. Installing the Operating System	4
2.1. Server Installation	4
2.1.1. Server Hardware	4
2.1.2. Disk and Storage	5
2.1.3. Installation of Python 3.8	5
2.1.4. Open Ports on the Firewall	6
2.1.5. Setup Services for Core Control Servers with Systemd	7
2.1.6. Disable SELinux	9
2.1.7. Check your services	9
2.1.8. Export the /data folder	10
2.1.9. Knowing your hosts	10
2.2. Client Installation	11
2.2.1. NFS mount the /data folder from the egse-server	11
2.3. User Administration	12
2.3.1. Provide sudo permission for plato-data	13
3. Install the Prometheus server	14
4. Install the Grafana server	15
5. Install the Linux node-exporter	16
5.1. Install the exporter	16
5.2. Install the dashboard	17
5.3. Testing the node exporter metrics	17
6. Install Python	18
6.1. Python Download Pages	18
6.2. Anaconda	18
7. Installation of PyCharm	20
8. Installation of the Common-EGSE	21
9. Installing the Test Scripts	23
10. Installation of Setups	24
11. Installation of Desktop Entries	25
12. Installation of a VNC Server	27
12.1. Install packages	27
12.2. Configure the GNOME Display Manager	27
12.3. Install and Activate the VNC service	27

13. Setting up the environment	29
13.1. Environment Variables.....	29
13.2. The environment for Systemd services	30
14. Setting up ssh access and GitHub deploy keys	32
14.1. Create a deploy key for the plato-common-egse.....	32
14.2. Create a deploy key for the plato-test-scripts	33
14.3. Create a deploy key for the plato-cgse-conf.....	35
15. Update the Common-EGSE to the latest release	38
16. Update the Test Scripts to the latest release	39
17. Update Python packages	40
18. Data Propagation	41
19. Shared Libraries.....	43
20. Installing External Tools.....	44
20.1. Cutelog GUI	44
20.2. Textualog TUI	44

Changelog

09/02/2023 — v1.3

- Added description for the `PLATO_CAMERA_IS_EM` environment variable, see [Chapter 13](#)
- Added section on providing *sudo* permissions to plato-data in the user administration, see [Section 2.3](#).
- Added section about install and set up a node exporter dashboard, see [Chapter 5](#).
- Added info on the 'hosts' file, see [Section 2.1.9](#).
- Small fixes and updates in several sections

20/01/2023 – v1.2

- We will now try to maintain a changelog.
- Documented the installation and activation of a VNC Server on Ubuntu, see [Chapter 12](#)
- Improved section on 'Setting up the environment', see [Chapter 13](#)
- Updated the version numbering used for releases in update commands, see [Chapter 15](#) and [Chapter 16](#)
- Added info on how to set the terminal title, see [\[set-terminal-title\]](#)
- Added explanation of `--ignore-installed` optional argument for `pip`, see [Chapter 17](#)
- Added a brief explanation on exporting a mount point on the egse-server [Section 2.1.8](#) and mounting the mount point [Section 2.2.1](#) on the egse-client.



Colophon

Copyright © 2022, 2023 by the KU Leuven PLATO CGSE Team

1st Edition — February 2023

This manual is written in PyCharm using the AsciiDoc plugin. The PDF Book version is processed with asciidoctor-pdf.

The manual is available as HTML from [ivs-kuleuven/github.io](https://ivs-kuleuven.github.io). The HTML pages are generated with Hugo which is an OSS static web-pages generator. From this site, you can also download the PDF books.

The source code is available in a GitHub repository at [ivs-kuleuven/plato-cgse-doc](https://github.com/ivs-kuleuven/plato-cgse-doc).

When you find an error or inconsistency or you have some improvements to the text, feel free to raise an issue or create a pull request. Any contribution is greatly appreciated and will be mentioned in the acknowledgement section.

Conventions used in this Book

We try to be consistent with the following typographical conventions:

Italic

Indicates a new term or ...

Constant width

Used for code listings, as well as within paragraphs to refer to program elements like variable and function names, data type, environment variables (**ALL_CAPS**), statements and keywords.

Constant width between angle brackets <text>

Indicates **text** that should be replaced with user-supplied values or by values determined by context. The brackets should thereby be omitted.

When you see a **\$...** in code listings, this is a command you need to execute in a terminal (omitting the dollar sign itself). When you see **>>> ...** in code listings, that is a Python expression that you need to execute in a Python REPL (here omitting the three brackets).



Chapter 1. Introduction

This guide explains the installation and configuration of the following components:

- The operating system (CentOS-8, Ubuntu) on the `egse-server` and `egse-client` machines
- The installation of basic tools like git, Python
- The installation of the Common-EGSE (CGSE) ← `plato-common-egse`
- The installation of the Test Scripts (TS) ← `plato-test-scripts`
- The installation of the Configuration files (Setups) ← `plato-cgse-conf`

Please note there is a difference between an installation on an operational machine with respect to your development environment. The operational machine has in principle a *read-only* installation. That means no files in the repositories shall be changed and it will not be possible to push any changes to the GitHub repositories from any operational machine. The development installation on your local laptop or desktop is an installation where you can have full control over your development. In this environment you make changes, test your code, document the code, update, merge, and push to your *origin* repository, to end with a pull request from *origin* to *upstream* which is the official GitHub repo.

We will also spend some time in this manual on how to update your system for security updates and how to update the Common-EGSE and test-scripts with new releases.

This manual was originally written for installation of and on CentOS-8, but in the mean time we moved to Ubuntu 20.04 LTS. When feasible, we will use TABs to allow you to switch between information and code for either OS.

Chapter 2. Installing the Operating System

2.1. Server Installation

2.1.1. Server Hardware

Proposed hardware for the egse-server:

- 1x Supermicro SYS-6019P-MT
- 2x Intel Xeon Gold 5120 s3647 Skylake-SP 14 Cores 28 Threads 2.2GHz 2.6GHz Turbo
- 4x Samsung 32GB DDR4-2666 2Rx4 (128GB main memory)
- 1x Intel SSD 240GB → system disk
- 1x Intel SSD 2TB → life data disk
- 1x Seagate SATA 12TB → archive disk
- 2x Ethernet RJ45 1Gb/s

The **egse-server** will do quite some work in communicating with all the test equipment, receive commands from the GUIs and Python REPL running on the **egse-client**, commanding the system under test (SUT), retrieve CCD data from the Camera, and store all data from all communication to disk. Especially, the storage of all the data is demanding and needs to be as performant as possible in order to keep up with the data rate from the N-FEE or F-FEE during testing. That is the reason we have chosen three separate disks in the **egse-server** machine. The small SSD is used to install the operating system and the home directories. The larger SSD is the **/data** disk where the life data is stored. This disk needs to be an SSD in order to keep up with the data rate. We have seen in tests that this is a crucial part in the chain to prevent the N-FEE from reporting internal buffer overflows because data is not read fast enough. It is important to keep the **/data** disk from running full, you need about 20% free for an SSD to be performant. The big SATA disk is the **/archive** disk which will contain all test data and is synchronised periodically from the **/data** life data.

The two Ethernet cards are foreseen to split network traffic from the SpaceWire connection if needed. Especially for the F-FEE it might be needed to allocate the full bandwidth of one Ethernet cable in order to cope with the multiplexed four SpaceWire connections.

In the next sections, we will briefly explain how to install the **egse-server** and **egse-client** machines. The frame below explains where to download and perform a basic installation of CentOS 8 or Ubuntu 20.04.

CentOS 8

Perform a normal/default installation of CentOS-8. Follow the default settings.

- Version: CentOS Linux release 8.1
- Download CentOS 8 from XXXX



- Boot mode: UEFI

Ubuntu 20.04

- Download the Ubuntu Server distribution from: <https://ubuntu.com/download/server>.
- Refer to the [step-by-step server installation instructions](#) if you need more insight.
- Perform a normal/default installation of Ubuntu 20.X.

2.1.2. Disk and Storage

The following directories will be created on the server side:

- **/home**: used for the software installations and daily work.
- **/data**: used to store life data, i.e. image data from the FEEs, housekeeping data, logging information, metrics, etc. This is the mounted SSD disk of 2TB.
- **/archive**: used to archive all data. This data is transferred to the data archive in Leuven on a daily basis. This is the mounted SATA disk of 12TB.

2.1.3. Installation of Python 3.8

We will install Python from the official Python website, as described in [Chapter 6](#). The procedure is as follows: We first install the development tools for CentOS and a number of **devel** packages that are needed for header files during compilation. We then get the Python source distribution from www.python.org, unpack, configure and compile. Use **altinstall** instead of **install** if you don't want previous installations of Python to be overwritten. This will install **python3.8** in **/usr/local/bin**. All the commands need to be executed as root.

CentOS 8

```
yum -y groupinstall "Development Tools"
yum -y install openssl-devel bzip2-devel libffi-devel
yum -y install wget
curl https://www.python.org/ftp/python/3.8.13/Python-3.8.13.tgz --output
Python-3.8.13.tgz
tar xvf Python-3.8.13.tgz
cd Python-3.8.13/
./configure --enable-optimizations
make altinstall
```

Ubuntu 20.04

Python 3.8 comes installed with Ubuntu 20.04, no need to re-install it.

2.1.4. Open Ports on the Firewall

By default CentOS-8 has the Firewall enabled. When your system is installed in a save environment without external connectivity, you could consider to disable the Firewall altogether.

```
systemctl status firewalld
systemctl stop firewalld
systemctl disable firewalld
systemctl mask firewalld
```

When you do need the Firewall to be enabled, open up all the ports that are used by the Common-EGSE core services. This might be a lot of work, but fortunately, you can define ranges when making ports available.

The following type of ports are used by control servers and other processes:

Name	Description
SSH	Normal secure shell communication port
COMMANDING_PORT	Used by the control servers for commanding the devices
MONITORING_PORT	Used by the control servers to periodically send out monitoring info
SERVICE_PORT	Used by the control servers for services not related to commanding
METRICS_PORT	Used by the control servers, data acquisition, and GUIs to serve the metrics to Prometheus through an internal HTTP service.
LOGGING_PORT	Used by the CGSE Logger
DATA_DISTRIBUTION_PORT	Used by the DPU Processor to distribute the N-FEE data
DEVICE	Device specific port that are used by the controllers or device interfaces to connect to.
OTHER	Grafana [3000], Cutelog [19996]

All the port numbers for the different processes are defined in the `settings.yaml` file in the CGSE distribution and can be overwritten in the local settings file at `$PLATO_LOCAL_SETTINGS`.

Opening ports and port ranges can be done by introducing a new service on the server. The example below opens up the ports for the Hexapod PUNA Control Server. The commands to set up the service on the `firewalld` are:

```
sudo firewall-cmd --permanent --new-service=puna-control
sudo firewall-cmd --permanent --service=puna-control --set-description="Hexapod PUNA
Control Services"
sudo firewall-cmd --permanent --service=puna-control --add-port=6700-6703/tcp
sudo firewall-cmd --permanent --zone=public --add-service=puna-control
```

```
sudo firewall-cmd --reload
```

Repeat the same sequence for the other control services and processes.

2.1.5. Setup Services for Core Control Servers with Systemd



You might want to do these steps only after you have installed Prometheus [Chapter 3], Grafana [Chapter 4] and the Common-EGSE [Chapter 8]

The control servers for this project that run on the **egse-server** are all managed by the **systemd** service manager. For information on **systemd** check out the documentation on the Redhat System Administration Site at [RHEL7](#).

The service files for each of the core control servers are located in the **server** directory at the root of the **plato-common-egse** project. You will have to adapt the services —especially the absolute paths— to your needs and setup. Then copy the service files into the **/etc/systemd/system** directory:

```
sudo cp sm_cs.service /etc/systemd/system
sudo cp cm_cs.service /etc/systemd/system
sudo cp pm_cs.service /etc/systemd/system
sudo cp log_cs.service /etc/systemd/system
sudo cp syn_cs.service /etc/systemd/system
```

The following code lists the entire service for the Storage Manager Control Server. The text **EnvironmentFile** and **WorkingDirectory** need special attention for your specific setup.

```
[Unit]
Description=Storage Manager Control Server
After=network-online.target

[Service]
Type=simple
Restart=always
RestartSec=3
User=plato-data
Group=plato-data
EnvironmentFile=/cgse/env.txt
WorkingDirectory=/home/plato-data/workdir
ExecStart=/cgse/bin/sm_cs

[Install]
Alias=sm_cs.service
WantedBy=multi-user.target
```

The service starts the specific control server from a script that was created during the **setuptools** installation, in our example in the **/cgse/bin** folder. Check the services files for the Configuration Manager and Process Manager also, they contain a specific delay time of 3s to ensure the Storage manager had enough time to start up and process registrations.

```
[Service]
ExecStartPre=/bin/sleep 3
```



You will also need to create the **/home/plato-data/workdir** folder for the user **plato-data**. Without this folder, the service will not start and you will get a **(code=exited, status=200/CHDIR)** when you run a **systemctl status** command for the service.

Once the services file is correct, start the service as follows:

```
sudo systemctl start sm_cs
```

and to automatically start the service on boot:

```
sudo systemctl enable sm_cs
```

The counter parts of the above commands are **stop** and **disable** where the former just stops the service and the latter prevents the service to start at boot time.

Whenever you have made a change to the services file and copied it back into the **/etc/systemd/system** directory, reload the daemons as follows:

```
sudo systemctl daemon-reload
```

If you need to know the status of one of the control services, use the following command, e.g. for the Process manager:

```
sudo systemctl status pm_cs.service
```

This prints out the status info on the service plus the last few messages that were sent to stdout or stderr.

When you want to check and follow the output in **/var/log/messages** for the specific service, you can use the **journalctl** command. An example for the process manager **pm_cs**:

```
sudo journalctl -f -u pm_cs
```



2.1.6. Disable SELinux



This section is only relevant if you have installed a CentOS-8 system, Ubuntu does not install SELinux by default and it is therefore probably not activated..

When you run into a authentication error while starting the control servers, you will need to disable SELinux (Security-Enhanced Linux). The error will look something like this (excerpt from `/var/log/messages`):

```
Sep 11 17:59:46 localhost systemd[1]: sm_cs.service: Service RestartSec=3s expired,
scheduling restart.
Sep 11 17:59:46 localhost systemd[1]: sm_cs.service: Scheduled restart job, restart
counter is at 369.
Sep 11 17:59:46 localhost systemd[1]: Stopped Storage Manager Control Server.
Sep 11 17:59:46 localhost systemd[1]: Started Storage Manager Control Server.
Sep 11 17:59:46 localhost systemd[22013]: sm_cs.service: Failed to execute command:
Permission denied
Sep 11 17:59:46 localhost systemd[22013]: sm_cs.service: Failed at step EXEC
spawning /cgse/bin/sm_cs: Permission denied
Sep 11 17:59:46 localhost systemd[1]: sm_cs.service: Main process exited,
code=exited, status=203/EXEC
Sep 11 17:59:46 localhost systemd[1]: sm_cs.service: Failed with result 'exit-code'.
Sep 11 17:59:47 localhost setroubleshoot[19162]: failed to retrieve rpm info for
/cgse/bin/sm_cs
Sep 11 17:59:47 localhost setroubleshoot[19162]: SELinux is preventing
/usr/lib/systemd/systemd from 'read, open' accesses on the file /cgse/bin/sm_cs. For
complete SELinux messages run: sealert -l a77af8c2-c91a-43cd-9b64-e7c0a5b24311
Sep 11 17:59:47 localhost platform-python[19162]: SELinux is preventing
/usr/lib/systemd/systemd from 'read, open' accesses on the file
/cgse/bin/sm_cs.#012#012***** Plugin catchall (100. confidence) suggests
*****#012#012If you believe that systemd should be allowed read
open access on the sm_cs file by default.#012Then you should report this as a
bug.#012You can generate a local policy module to allow this access.#012Do#012allow
this access for now by executing:#012# ausearch -c '(sm_cs)' --raw | audit2allow -M
my-smcs#012# semodule -X 300 -i my-smcs.pp#012
```

To disable SELinux, edit the `/etc/selinux/config` file and set `SELINUX=disabled`. Then reboot your system (this is a kernel setting, therefore we need to reboot).

2.1.7. Check your services

A simple and quick way to check if the core services are still running together with Prometheus^[1] and Grafana^[2] is to check the running processes:

```
[plato-data@egse-server]$ ps -ef|egrep "prometheus|grafana|_cs"
plato-d+  64839      1  5 Jun24 ?          08:17:43 /home/plato-
data/software/prometheus/prometheus --config.file /home/plato-
```

```
data/software/prometheus/prometheus-egse-server.yml --storage.tsdb.path
/data/metrics/data/
plato-d+ 808513      1  0 Apr19 ?      06:33:25 /home/plato-
data/software/grafana/bin/grafana-server
plato-d+ 2519545     1  4 Jun21 ?      09:12:10 /usr/bin/python3 /cgse/bin/sm_cs
start
plato-d+ 2519684     1  3 Jun21 ?      06:57:04 /usr/bin/python3
/cgse/bin/syn_cs start
plato-d+ 2519771     1  2 Jun21 ?      04:36:55 /usr/bin/python3 /cgse/bin/cm_cs
start
plato-d+ 2543093     1  0 Jun21 ?      00:28:03 /usr/bin/python3
/cgse/bin/log_cs start
plato-d+ 2633916     1  2 Jun21 ?      04:28:20 /usr/bin/python3 /cgse/bin/pm_cs
start
[plato-data@egse-server]$
```

2.1.8. Export the /data folder

We will need access to the `/data` folder on the egse-client machine and will add the following to the `/etc/exports` file. Adding this info will need root permissions (you can use the `sudo` command too). Make sure you use the correct IP address of the egse-server from which this mount is accessed.

```
/data    192.168.0.74/26(rw,sync)
```

Some of our servers have two ethernet interfaces and you might want to add an additional IP address to the exports, e.g.:

```
/data/IAS    egse-server.ivs.kuleuven.be(rw,sync) 192.168.80.14(rw,sync)
```

The only thing to do now is to restart the NFS server on the egse-server machine:

```
service nfs-server status
sudo service nfs-server restart
```

How you need to NFS mount this `/data` folder on the egse-client machine is explained in the section on Client Installation, see [Section 2.2](#).

2.1.9. Knowing your hosts

For several reasons you will need to access other computers or devices in your network, e.g. as the URL for the Grafana monitoring web pages. It will therefore be very useful to access them with their usual name instead of using an IP address. The easiest way to accomplish this is adding entries in the `/etc/hosts` file. This needs to be done as root, and the following lines illustrates an



example (IPv6 entries are not shown):

```
$ sudo vi /etc/hosts
127.0.0.1 localhost
137.145.177.84 sour server-csl2
137.145.177.85 pisco client-csl2
```

2.2. Client Installation

- Perform a normal/default desktop installation of CentOS-8 or Ubuntu 20.
- Create a user plato-user
- Log in as user plato-user
- install the plato-common-egse in ~/git (see [Chapter 8](#)) *

2.2.1. NFS mount the /data folder from the egse-server

The first thing to do is to check which mounts are exported by the egse-server. That can easily be done with the `showmounts` command.

```
$ showmount -e 139.xxx.yyy.74
Export list for 139.xxx.yyy.74:
/data 139.xxx.yyy.74/26
```

With this knowledge you can already mount the directory on the egse-client:

```
$ sudo mount 139.xxx.yyy.74:/data /data
```

We will need to make this mount permanently available, also after rebooting the system. This can be done as a static mount or by using the autmounter daemon.

Static mount

With `fstab` we will create a static mount for the `/data` location. We add the mount point to `/etc/fstab` which will automatically mount the folder at startup and it will remain mounted until system shutdown or explicitly unmounted.

```
$ sudo vi /etc/fstab
...
139.xxx.yyy.74:/data /data      nfs      rw, sync 0 0
```

This now also provides us a simpler way to mount the directory:

```
$ sudo mount /data
```

Automounter

The automounter daemon is controlled by the **autofs** service. In the master configuration file for **autofs**, add a new line to specify a direct map for the file **/etc/auto.cgse**:

```
$ sudo vi /etc/auto.master
...
/- /etc/auto.cgse
```

The **/etc/auto.cgse** file shall then contain the following:

```
/data -fstype=nfs 139.xxx.yyy.74:/data
```

Finally you should reload the **autofs** configuration:

```
$ sudo service autofs reload
```

2.3. User Administration

There are 4 specific PLATO users defined:

- **plato-admin**: has basically the same rights as root and is used for system installation and administration tasks. All root commands must be executed with **sudo**, no password will be asked. Do not usually login to this account.
- **plato-ops**: is used to administer and control the Common-EGSE core services. This user can start and stop the Common-EGSE control servers as a service with the **systemd** command **systemctl**. Log in to this account to monitor the **systemd** services with **systemctl** and **journalctl**. This user can also control and monitor the Prometheus and Grafana servers using **systemctl** and **journalctl**.
- **plato-data**: all services should run under **plato-data**, data locations will be writable by **plato-data** and readable by **plato-user**. You do not normally log into this account, but the services are started under this account. All the software (Common-EGSE and Test Scripts), and all the data created by the different processes and control servers can best be run and created by the **plato-data** user for consistency. If device control servers need to be started manually, use this account.
- **plato-user**: the generic user account for running the test scripts, start GUIs and analysing the data. This user has no **sudo** rights, but has read access to the **/data** directory. This is the account used to execute the test scripts. Don't use this account on the **egse-server** unless you know what you are doing.



2.3.1. Provide sudo permission for plato-data

In a terminal session, log in as root, then add the following line in the file `/etc/sudoers`. You will need to edit this file using the `visudo` command^[3].

```
# visudo
```

Now add the line. This is best done in the section 'User privilege specification' to keep the file clean:

```
plato-data    ALL=(ALL:ALL) ALL
```

This will give 'root' privileges to the plato-data user. You might not want to do that on your system! It should be plato-admin with these rights and plato-data with some execution rights for systemctl and journalctl,

TODO: Describe the other sudo way....

[1] The installation of Prometheus is explained in [Chapter 3](#)

[2] The installation of Grafana is explained in [Chapter 4](#)

[3] The `visudo` command uses the EDITOR environment variable to start your favorite editor. I have set `export EDITOR=vi`

Chapter 3. Install the Prometheus server

Please note that in the developer documentation under the section [Monitoring](#) there is a description on *Installing Prometheus*. I will here only describe the setup for the `egse-server`. The best is to create a dedicated directory for the software installations, e.g. `~/software`. Then install Prometheus into that folder:

```
$ mkdir ~/software
$ cd ~/software
$ curl -L -o prometheus-2.36.2.linux-amd64.tar.gz
https://github.com/prometheus/prometheus/releases/download/v2.36.2/prometheus-
2.36.2.linux-amd64.tar.gz ①
$ tar xzvf prometheus-2.36.2.linux-amd64.tar.gz
$ ln -s prometheus-2.36.2.linux-amd64 prometheus
```

① the `-L` option is needed because the link will redirect and with this option `curl` follows the redirect.

We want to automatically start the Prometheus server from the systemd services as we did with the core-egse services. The service file, i.e. `prometheus.service`, can be copied from the `server` directory in the distribution to the `/etc/systemd/system` folder, same as for the core-egse services. Make sure you update the locations if necessary. The configuration files for Prometheus, i.e. `prometheus.yml` and `prometheus.rules.yml`, can best be soft linked from the `metrics` folder into the installation folder of Prometheus. That will automatically keep these files update-to-date with a new release of the software.

```
$ cp ~/git/plato-common-egse/server/prometheus.service /etc/systemd/system
$ ln -s ~/git/plato-common-egse/metrics/prometheus.yml ~/software/prometheus
$ ln -s ~/git/plato-common-egse/metrics/prometheus.rules.yml ~/software/prometheus
```

Finally, create the `metrics/data` directory in the proper location, e.g. in `/data`. That is the location given with the `--storage.tsdb.path` option in the Prometheus service file.

```
$ mkdir -p /data/metrics/data
```

Then enable the service as user `plato-admin` and reload the systemd services daemon:

```
$ sudo systemctl enable prometheus
$ sudo systemctl daemon-reload
$ sudo systemctl start prometheus
```



Chapter 4. Install the Grafana server

Please note that in the developer documentation under the section [Monitoring](#) there is a description on *Installing Grafana*. I will here only describe the setup for the `egse-server`. The best is to create a dedicated directory for the software installations, e.g. `~/software`. Then install Grafana into that folder.^[1]

```
$ curl -L -o grafana-enterprise-8.5.6.linux-amd64.tar.gz
https://dl.grafana.com/enterprise/release/grafana-enterprise-8.5.6.linux-
amd64.tar.gz
$ tar xzvf grafana-enterprise-8.5.6.linux-amd64.tar.gz
$ ln -s grafana-8.5.6 grafana
```

Grafana doesn't need any further configuration. That is done in the dashboards that are loaded as explained in XXXXX [Monitoring/Dashboard Configuration](#).

We also want the Grafana server to automatically start from the systemd services as we did for Prometheus. We currently use Grafana with the default configuration and have the database located in the installation directory. The service file is located in the `server` folder of the Common-EGSE project and should be copied to `/etc/systemd/system`. After that, enable Grafana with `systemctl` and reload the services daemon.

[1] Don't try to install Grafana using `yum`, because that will bring you into trouble with configuration files etc.

Chapter 5. Install the Linux node-exporter

5.1. Install the exporter

One of the useful dashboards that we have in Grafana is the *Linux Host Dashboard*. This dashboard pulls data from Prometheus and the node exporter. Install the node exporter on the egse-server and egse-client with the following commands as explained on the [node-exporter guide page](#).

```
$ cd ~/software
$ wget
https://github.com/prometheus/node_exporter/releases/download/v1.5.0/node_exporter-
1.5.0.linux-amd64.tar.gz
$ tar xzvf node_exporter-1.5.0.linux-amd64.tar.gz
$ ln -s node_exporter-1.5.0.linux-amd64 node_exporter
```

Then create a service in the `/etc/systemd/system` directory by copying the file `node_exporter.service` from the repository.

```
$ sudo cp ~/git/plato-common-egse/server/node_exporter.service /etc/systemd/system
```

Now enable the service, reload the systemd daemon and start the service with^[1]:

```
$ sudo systemctl enable node_exporter
$ sudo systemctl daemon-reload
$ sudo systemctl start node_exporter
```

Finally, the Prometheus service shall be restarted. Since this service is running on the egse-server, the configuration file `prometheus.yml` shall contain a scraping job for both the server and the client. The example below shows the entries for the machines at CSL where the server is 'localhost' and the client is 'pisco'.

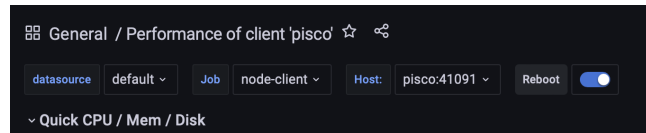
```
- job_name: node-server
  static_configs:
    - targets: ['localhost:41091']

- job_name: node-client
  static_configs:
    - targets: ['pisco:41091']
```



5.2. Install the dashboard

The node exporter dashboard can be imported directly from Grafana with ID 1860 (see [node exporter dashboard](#)). Configure the dashboard at the top of the dashboard setting the 'Job' and the 'Host'. From the example scrape jobs above use 'node-server', 'node-client' for the Job and 'localhost', 'pisco' for the Host respectively for the server and client dashboards.



5.3. Testing the node exporter metrics

On the server machine where the node exporter is running, you can test its proper functioning with the following command:

```
$ curl localhost:41091/metrics | egrep -v "^#"
```

That will print all metrics from the node exporter in the terminal.

[1] the daemon shall be reloaded because the service file was added

Chapter 6. Install Python



Use Python 3.8+

This code is written for **Python 3** and uses features of Python 3.8 (e.g. walrus operator), so make sure you have at least Python 3.8 installed and configured on your system before trying any of the guides. You can find installation instructions below.

6.1. Python Download Pages

Download the required version of Python from the [official Python download website](#). When you press the "Download" button, you should automatically be re-directed to the download page for your operating system.

After the download has completed, execute the package file and follow the instructions during the installation process.

For macOS 10.9 or higher, this will install Python in the dedicated system folder `/Library/Frameworks/Python.framework/Versions/<version number>`. For Linux a tarball with the latest release is available. For Windows you can download an executable installer or a ZIP file, depending on your preferences.



6.2. Anaconda



Python on operational system

Make sure you install the official release of Python on any operational machine and **not** the Anaconda distribution. There are too many dependency problem to solve for the Anaconda installation.

On you development machine, you can, alternatively, install Python with the [Anaconda distribution](#). This comes with the benefit of installing many additional packages for development, data analysis, and visualisation. Anaconda however nests itself into your system and makes it's difficult to set up environments without the interference of Anaconda. It also uses it's own package management and update script instead of the standard Python distribution with `pip`. Make sure you know what your doing before using this option.

Links to the download pages (follow the instruction listed there):

- [for macOS](#)
- [for Linux](#)



- [for Windows](#)

Upon installation, the following questions will pop up:

- accept license: yes
- where to install it
- initialise Anaconda in your `.bashrc`: no
- whether VS Code should be installed (source code editor): optional

This completes the Anaconda installation.





Chapter 7. Installation of PyCharm

PyCharm is the IDE that we will use to execute test scripts. For this purpose the PyCharm Community edition is sufficient.

TBW

Chapter 8. Installation of the Common-EGSE

The installation will be done as the `plato-data` user. We will *clone* the IvS-KULeuven `plato-common-egse` repository. There is no need to *fork* because we will not develop from this account and will therefore not do any pushes or pull requests.

Create a `~/git` folder in the home directory of the `plato-data` user, move into that directory and clone the GitHub repository. Then move into the `plato-common-egse` folder that was created by the *clone* command.

```
$ mkdir -p ~/git
$ cd git
$ git clone https://github.com/IvS-KULeuven/plato-common-egse.git
$ cd plato-common-egse/
```

The installation of the Common-EGSE will be done in a location that is accessible for `plato-data` and `plato-user`. Run the following commands as root. That will create the location where the Python installation will be done and give the proper permissions to the folders.

```
$ mkdir -p /cgse/lib/python
$ chown -R plato-data:plato-data /cgse
$ ls -ld /cgse
drwxr-xr-x. 4 plato-data plato-data 4096 Sep 11 16:55 /cgse
```

If not done already, you will need to install the `wheel` package, which is needed to install binary Python distributions.

```
$ python3.8 -m pip install wheel
```

Before actually installing the Common-EGSE, we have to set the `PATH` and `PYTHONPATH` environment variables, and checkout the branch of the release that we want to install^[1].

```
$ PATH=/cgse/bin:$PATH
$ export PYTHONPATH=/cgse/lib/python/
$ git fetch updates
$ git checkout tags/<release tag> -b <release tag>-branch ①
$ python3.8 setup.py install --home=/cgse/
$ git checkout develop ②
```

① The release tag takes the form `YYYY.MAJOR.MINOR-TH-CGSE`, e.g. `2022.2.17-IAS-CGSE`

② make sure to go back to the develop branch after the installation

The above commands install the full Common-EGSE and all its dependencies in the `/cgse/lib/python` folder. Note that we have not used any Python virtual environment for this installation.

After a successful installation, you can check which packages are known to Python and where they are located:

```
[plato-data@localhost ~]$ python3.8 -m site
sys.path = [
    '/cgse/lib/python',
    '/cgse/lib/python/ThorlabsPM100-1.2.2-py3.8.egg',
    '/cgse/lib/python/PyVISA-1.12.0-py3.8.egg',
    '/cgse/lib/python/PyVISA_py-0.5.3-py3.8.egg',
    '/cgse/lib/python/pyserial-3.5-py3.8.egg',
    '/cgse/lib/python/pylibftdi-0.20.0-py3.8.egg',
    '/cgse/lib/python/pyusb-1.2.1-py3.8.egg',
    '/cgse/lib/python/xlrd-2.0.1-py3.8.egg',
    '/cgse/lib/python/visidata-2.8-py3.8.egg',
    '/cgse/lib/python/typing_extensions-4.2.0-py3.8.egg',
    '/cgse/lib/python/transitions-0.8.11-py3.8.egg',
    '/cgse/lib/python/transforms3d-0.3.1-py3.8.egg',
    '/cgse/lib/python/textual-0.1.18-py3.8.egg',
    '/cgse/lib/python/sshtunnel-0.4.0-py3.8.egg',
    '/cgse/lib/python/rich-12.4.4-py3.8.egg',
    ...
    '/cgse/lib/python/pycparser-2.21-py3.8.egg',
    '/cgse/lib/python/pytz_deprecation_shim-0.1.0.post0-py3.8.egg',
    '/cgse/lib/python/tzdata-2022.1-py3.8.egg',
    '/cgse/lib/python/Common_EGSE-2022.2.16_IAS_CGSE-py3.8.egg',
    '/usr/local/lib/python38.zip',
    '/usr/local/lib/python3.8',
    '/usr/local/lib/python3.8/lib-dynload',
    '/usr/local/lib/python3.8/site-packages',
]
USER_BASE: '/home/plato-data/.local' (doesn't exist)
USER_SITE: '/home/plato-data/.local/lib/python3.8/site-packages' (doesn't exist)
ENABLE_USER_SITE: True
[plato-data@localhost ~]$
```

[1] The latest release tag can be found on the GitHub pages of the repository



Chapter 9. Installing the Test Scripts

The installation will be done as the `plato-data` user. We will *clone* the IvS-KULeuven `plato-test-scripts` repository. There is no need to *fork* because we will not develop from this account and will therefore not do any pushes or pull requests.

Create a `~/git` folder in the home directory of the `plato-data` user, move into that directory and clone the GitHub repository. Then move into the `plato-test-scripts` folder that was created by the *clone* command. If this is a first-time installation, you must first create a deploy key for the `plato-test-scripts`, see [Chapter 14](#).

```
$ mkdir -p ~/git
$ cd git
$ git clone git@repo-test-scripts:IvS-KULeuven/plato-test-scripts.git
$ cd plato-test-scripts/
```

Now, make a virtual environment for your `plato-test-scripts` installation:

```
$ python3 -m venv venv --prompt 'TS-venv'
$ source venv/bin/activate
```

Chapter 10. Installation of Setups

The Setups are the configuration files for your test house and define all the configuration settings for your equipment and for the camera (SUT). All the Setups for all the test houses live in the `plato-cgse-conf` repository. This section explains how to install this repo on the server and on the client.

Installation on the server

The `plato-cgse-conf` repository shall be cloned on the server in the folder `~/git` where also the `plato-common-egse` resides. To clone the repo, you need access to this repository first, follow the steps described in [Section 14.3](#). When you have executed that procedure, you should have the repository up-to-date on your system.



Check that the *upload* remote is indeed a tracking branch, see [upload-tracking](#). If not, your new submitted Setups will not be pushed to the GitHub repository.

Make a link in the `PLATO_DATA_STORAGE_LOCATION` to the `conf` directory containing the Setups for your test house.

```
$ ln -s ~/git/plato-cgse-conf/data/<TH>/conf /data/<TH>/conf ①
```

① replace `<TH>` in the above command with `CSL1`, `CSL2`, `IAS`, `INTA`, or `SRON`.

Installation on the client

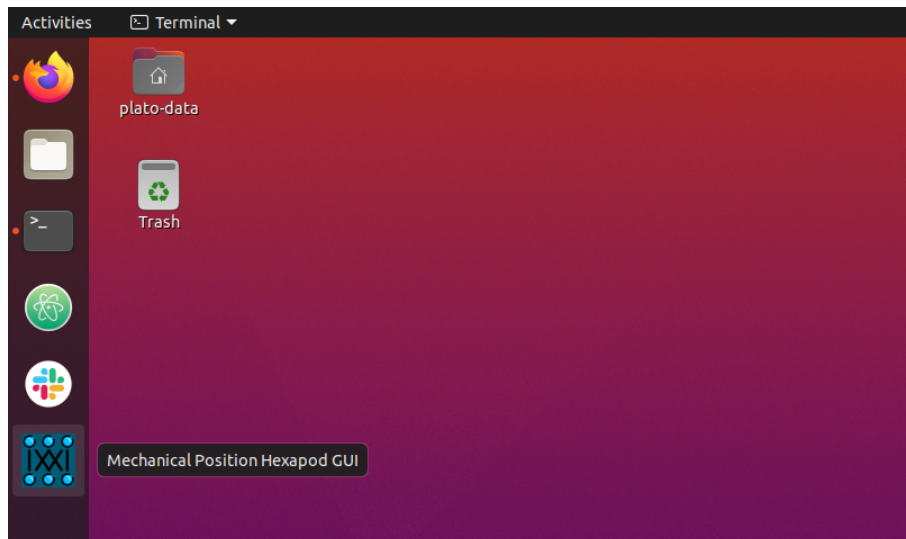
For the client follow the same procedure to create a deploy key and clone the repository (see [Section 14.3](#)), but **do not** allow write access on GitHub and only create the remote *updates* (the *upload* remote is not used on the client).

Alternatively, the folder `~/git/plato-cgse-conf` can be mounted read-only from the server. make sure the location is also `~/git/plato-cgse-conf`.

The `/data` folder is usually already mounted from the server, so the link from `/data` already exists and should be functional.

Chapter 11. Installation of Desktop Entries

This section explains how to install a Desktop icon for one of the GUI applications. In the screenshot below, you see the icon for the *Mechanical Positions Hexapod GUI* in the toolbar on the left side of the screen. Click that icon to start the Task GUI.



A desktop entry is defined in a file with the `.desktop` extension. The content of the desktop file for the Mechanical Position Hexapod is shown below. The desktop files for several Task GUIs are located in the repo folder `~/git/plato-common-egse/client/`.

The content of `mech_pos_ui.desktop`:

```
[Desktop Entry]
Name=Mechanical Position Hexapod GUI
Comment=Control the Hexapod on the Mechanical Position
GenericName=Task GUI
Exec=/home/plato-data/bin/start_mech_pos_ui.sh ①
Icon=/home/plato-data/git/plato-common-egse/src/egse/icons/logo-puna.svg ②
Type=Application
StartupNotify=true
Categories=Development;IDE;
Terminal=false
MimeType=text/plain;
```

- ① Commands are executed from a Bourne Shell and we therefore need to set the proper environment before we can start the GUI. The easiest way to do that is in a separate script that we will install in the `~/bin` directory.
- ② A nice looking icon is available for most of the GUIs at this location.

Install the `start_mech_pos_ui.sh` scripts in the `~/bin` directory and make this file executable. The content of this file is given below. Note that when you are starting a GUI that resides in the `plato-test-scripts`, you will also need to load the proper Python environment. An example is the

`start_pm_ui.sh` script, also shown below.

Content of the `start_mech_pos_ui.sh` script

```
#!/usr/bin/bash

source ~/.bash_profile

mech_pos_ui
```

Content of the `start_pm_ui.sh` script

```
#!/usr/bin/bash

source ~/.bash_profile
source ~/git/plato-test-scripts/venv/bin/activate

pm_ui
```

Now finally, use the following commands to verify and install the desktop entry:

```
$ desktop-file-validate mech_pos_ui.desktop
$ desktop-file-install --dir ~/.local/share/applications mech_pos_ui.desktop
$ update-desktop-database ~/.local/share/applications/
```

The desktop entry should now appear in the 'Show Applications' view on your desktop. If you right click the icon, you can *Add to Favourites* and the icon will appear in the toolbar.

After this you can delete the `mech_pos_ui.desktop` file that you created.

Chapter 12. Installation of a VNC Server

This section describes how to install a VNC Server on your client machine. The VNC server will run on `display=:0` and will allow remote users to connect to the main desktop that is used by the operator. This remote connection will then be able to provide support and in case of emergency take over the desktop from the operator.

We will install the packages `x11vnc` and `net-tools`, then we will configure the GNOME Display Manager (`gdm3`) and finally install and activate the X11 VNC service.

12.1. Install packages

Use the following command to install the necessary packages. You can do this as user 'plato-data' and when asked for a password, provide the password of that user^[1].

```
$ sudo apt-get install x11vnc net-tools
```

12.2. Configure the GNOME Display Manager

Edit the file `/etc/gdm3/custom.conf` to disable Wayland and enable automatic login by the 'plato-data' user. The changes are all in the daemon section of the configuration file. If some entries do not exist on your system, create them. You should edit this file as 'root' or with `sudo` under the 'plato-data' user.

```
[daemon]
# Uncomment the line below to force the login screen to use Xorg
WaylandEnable=false

# Enabling automatic login
AutomaticLoginEnable = true
AutomaticLogin = plato-data
```

12.3. Install and Activate the VNC service

To install the X11 VNC service under Systemd, create a file `/etc/systemd/system/x11vnc.service` with the following content.



In the text below, change the `####` into the `id` of the user that was set in the configuration file above. The `id` must be an integer number, e.g. 1003.

```
[Unit]
Description=x11vnc service
After=display-manager.service network.target syslog.target
```

```
[Service]
Type=simple
ExecStart=/usr/bin/x11vnc -forever -shared -display :0 -auth
/run/user/####/gdm/Xauthority
ExecStop=/usr/bin/killall x11vnc
Restart=on-failure

[Install]
WantedBy=multi-user.target
```

Then, reload the daemon configuration and start the service:

```
$ sudo systemctl daemon-reload
$ sudo systemctl enable x11vnc.service
$ sudo systemctl start x11vnc.service
```

[1] The 'plato-data' user is supposed to have sudo rights.

Chapter 13. Setting up the environment

13.1. Environment Variables

The environment variables that are used by the CGSE or test scripts are listed in the table below. It is best practice to define those variable in `~/.bash_profile` to have them loaded on login. With the below definitions, a standard `~/.bash_profile` would look like this (example taken from the CSL1 egse-client *daiquiri*)

```
# plato-data@daiquiri:~$ cat ~/.bash_profile
source ~/.bashrc

export PATH=/cgse/bin:~/git/plato-test-scripts/venv/bin:~/local/bin:$PATH

export PYTHONPATH=/cgse/lib/python:/cgse/lib/python3.8/site-packages:~/git/plato-test-scripts/src:~/git/plato-test-scripts/venv/lib/python3.8/site-packages
export PYTHONSTARTUP=~/git/plato-test-scripts/startup.py

export PLATO_LOCAL_SETTINGS=/cgse/local_settings.yaml
export PLATO_DATA_STORAGE_LOCATION=/data/CSL1/
export PLATO_CONF_DATA_LOCATION=/data/CSL1/conf/
export PLATO_LOG_FILE_LOCATION=/data/CSL1/log/
export PLATO_INSTALL_LOCATION=/cgse
export PLATO_COMMON_EGSE_PATH=/home/plato-data/git/plato-common-egse

export LD_LIBRARY_PATH=~/git/plato-common-egse/src/egse/lib/ximc/libximc.framework/

alias cd2cgse="cd ~/git/plato-common-egse"
alias cd2ts="cd ~/git/plato-test-scripts"
```

A Terminal Title

Something else that you can add to your `~/.bash_profile` on the client machine is the following function (not really an environment setting, but still useful):

```
function term_title(){
    if [ -z "$PS1_BACK" ]; # set backup if it is empty
    then
        PS1_BACK="$PS1"
    fi

    TITLE="\[ \e]0;$*\a\"
    PS1="${PS1_BACK}${TITLE}"
}
```

This function will allow you to set the title of a Terminal window or a TAB in a Terminal window. This was possible on CentOS-8 with a context menu entry when right-clicking on the terminal TAB, but unfortunately that menu entry doesn't exist on Ubuntu. Run the function from within the terminal session that needs a new title, e.g. **TS-venv**:

```
$ term_title TS-venv
```

Table 1. Overview of all the environment variables

Variable Name	Default value
PYTHONPATH – egse-server	/cgse/lib/python/:/cgse/lib/python3.8/site-packages
PYTHONPATH – egse-client ^[1]	/cgse/lib/python:/cgse/lib/python3.8/site-packages:/home/plato-data/git/plato-test-scripts/src:/home/plato-data/git/plato-test-scripts/venv/lib/python3.8/site-packages/
PLATO_COMMON_EGSE_PATH	/home/plato-data/git/plato-common-egse
PLATO_CONF_DATA_LOCATION	/data/<SITE_ID>/conf
PLATO_CONF_REPO_LOCATION ^[2]	/home/plato-data/git/plato-cgse-conf
PLATO_DATA_STORAGE_LOCATION	/data/<SITE_ID>
PLATO_INSTALL_LOCATION	/cgse
PLATO_LOCAL_SETTINGS	/cgse/local_settings.yaml
PLATO_LOG_FILE_LOCATION	/data/<SITE_ID>/log
LD_LIBRARY_PATH	/home/plato-data/git/plato-common-egse/src/egse/lib/ximc/libximc.framework
PLATO_CAMERA_IS_EM ^[3]	True False

13.2. The environment for Systemd services

For the CGSE core services, that are started by the Systemd on the egse-server, the environment variable are defined in the file **/cgse/env.txt**. The example below shows the content of this file from the CSL1 egse-server *strawberry*

```
# plato-data@strawberry:~$ cat /cgse/env.txt
PYTHONPATH=/cgse/lib/python/:/cgse/lib/python3.8/site-packages
PLATO_LOCAL_SETTINGS=/cgse/local_settings.yaml
PLATO_CONF_DATA_LOCATION=/data/CSL1/conf
PLATO_CONF_REPO_LOCATION=/home/plato-data/git/plato-cgse-conf
PLATO_DATA_STORAGE_LOCATION=/data/CSL1
PLATO_LOG_FILE_LOCATION=/data/CSL1/log
```

[1] The PYTHONPATH on the client is different because it also has the plato-test-scripts installed, so in addition to the CGSE



packages, it needs to have access to the test scripts Python code and packages that were installed as part of the plato-test-scripts virtual environment.

[2] This variable only needs to be defined in the `/cgse/env.txt` file on the egse-server.

[3] PLATO_CAMERA_IS_EM was introduced in release 2022.3.2+CGSE to handle the difference between EM and pFM camera with respect to the format of image and housekeeping data provided by the FPGA. The EM camera data was in twos-complement.

Chapter 14. Setting up ssh access and GitHub deploy keys

We have a semi-automatic update procedure in place for the Common-EGSE and the Test Scripts. This procedure will be described in [Chapter 15](#) and [Chapter 16](#), but before we can use that we need to have the proper permissions to fetch changes from the different GitHub repositories without the need to provide our credentials every time. The best way to do that is to set up deploy keys on the GitHub repos.

14.1. Create a deploy key for the plato-common-egse

You can create an ssh key-pair for the CGSE with the following command. Execute this code on the `egse-server` as user `plato-data` in the folder `~/.ssh` (create the folder if it doesn't exist).

```
$ cd ~/.ssh
$ ssh-keygen -t ed25519 -C plato-data@egse-server-inta ①
Generating public/private ed25519 key pair.
Enter file in which to save the key (/home/plato-data/.ssh/id_ed25519):
id_cgse_server_inta ②
Enter passphrase (empty for no passphrase): ③
Enter same passphrase again:
Your identification has been saved in id_cgse_server_inta
Your public key has been saved in id_cgse_server_inta.pub
The key fingerprint is:
SHA256:kOUBdc*****7NSgd1WX+Ds plato-data@egse-server-inta
The key's randomart image is:
+--[ED25519 256]--+
| ..000+*00=..+.+|
| . .+. =++0..00|
| . 0 0 00+0= .|
| . = .00= =.|
| . + S .0 B +|
| 0          = E |
| 0          . .|
| 0          |
| .          |
+-----[SHA256]-----+
```

① The email address can be `plato-data@egse-server-<th>` where `<th>` is your test house site id.

② name of the file: `id_cgse_server-<th>`, again `<th>` is the test house site id.

③ do not provide a passphrase, just hit return

Send the `id_cgse_server-<th>.pub` file to the maintainer of the GitHub repository. She will copy the content of this file into a new deploy key for the `plato-common-egse` @ GitHub.



Now we need to create a generic hostname for the repository such that this can be picked up by the ssh protocol when accessing the repository at GitHub. Add the following lines to the file `~/.ssh/config`:

```
Host repo-common-egse
  Hostname github.com
  IdentityFile ~/.ssh/id_cgse_server_<th> ①
```

① don't forget to use your test house site id

Since we have created some new files in the `~/.ssh` folder we have to make sure the permissions of these files are correct and also the `~/.ssh` folder itself is fully protected.

```
$ ls -ld .ssh
drwx-----. 2 plato-data plato-data 4096 May 25 12:24 .ssh

$ ls -l .ssh
-rw-r--r--  1 plato-data plato-data  412 May 23 14:19 config
-rw-----  1 plato-data plato-data  419 May 23 14:16 id_cgse_server_inta
-rw-r--r--  1 plato-data plato-data  109 May 23 14:16 id_cgse_server_inta.pub
```

The file permissions can be changed with the `chmod` command as follows:

```
$ chmod 700 ~/.ssh
$ chmod 644 ~/.ssh/config
```

We will now add a new remote to our git repository. This is needed to (1) use the generic hostname created above, and (2) use a standard name for the remote that is used by the update script. Add a remote for doing the updates as follows:

```
$ cd ~/git/plato-common-egse
$ git remote add updates git@repo-common-egse:IvS-KULeuven/plato-common-egse.git
```

That was the last step, we can now try to fetch new updates from the GitHub repo to confirm that this works as expected:

```
$ git fetch updates
```

14.2. Create a deploy key for the plato-test-scripts

You can create an ssh key-pair for the test scripts with the following command. Execute this code on the `egse-client` as user `plato-data` in the folder `~/.ssh` (create the folder if it doesn't exist).



```

$ cd ~/.ssh
$ ssh-keygen -t ed25519 -C plato-data@egse-client-<th> ①
Generating public/private ed25519 key pair.
Enter file in which to save the key (/home/plato-data/.ssh/id_ed25519):
id_test_scripts-<th> ②
Enter passphrase (empty for no passphrase): ③
Enter same passphrase again:
Your identification has been saved in id_test_scripts-<th>
Your public key has been saved in id_test_scripts-<th>.pub
The key fingerprint is:
SHA256:kOUBdc*****7NSgd1WX+Ds plato-data@egse-client-<th>
The key's randomart image is:
+--[ED25519 256]--+
|  ..000+*00=..+.+|
|  .  ..+ =++0..00|
|  . 0 0 00+0= .|
|  . = .00= =.|
|  . + S .0 B +|
|  o      = E |
|  o      . .|
|  o      |
|  .      |
+-----[SHA256]-----+

```

- ① The email address can be `plato-data@egse-client-<th>` where `<th>` is your test house site id.
- ② name of the file: `id_egse_egse_server-<th>`, again `<th>` is the test house site id.
- ③ do not provide a passphrase, just hit return

Send the `id_test_scripts-<th>.pub` file to the maintainer of the GitHub repository. She will copy the content of this file into a new deploy key for the `plato-test-scripts @ GitHub`.

Now we need to create a generic hostname for the repository such that this can be picked up by the ssh protocol when accessing the repository at GitHub. Add the following lines to the file `~/.ssh/config`:

```

Host repo-test-scripts
  Hostname github.com
  IdentityFile ~/.ssh/id_test_scripts-<th> ①

```

- ① don't forget to use your test house site id

If you have not yet cloned the `plato-test-scripts` repository, you can do that now with the following command:

```
$ cd ~/git/
```



```
$ git clone git@repo-test-scripts:IvS-KULeuven/plato-test-scripts.git
```

We will now add a new remote to our git repository. This is needed to (1) use the generic hostname created above, and (2) use a standard name for the remote that is used by the update script. Add a remote for doing the updates as follows:

```
$ cd ~/git/plato-test-scripts
$ git remote add updates git@repo-test-scripts:IvS-KULeuven/plato-test-scripts.git
```

That was the last step, we can now try to fetch new updates from the GitHub repo to confirm that this works as expected:

```
$ git fetch updates
```

On a first-time-installation, perform an update as follows:

```
$ git fetch updates
$ git rebase updates/develop
$ python3 -m pip install -e .
```

otherwise, use the `update_ts` command:

```
$ update_ts
```

14.3. Create a deploy key for the plato-cgse-conf

You can create an ssh key-pair for the plato-cgse-conf with the following command. Execute this code on the `egse-server` as user `plato-data` in the folder `~/.ssh` (create the folder if it doesn't exist).

```
$ cd ~/.ssh
$ ssh-keygen -t ed25519 -C plato-data@egse-server-<TH> ①
Generating public/private ed25519 key pair.
Enter file in which to save the key (/home/plato-data/.ssh/id_ed25519):
id_cgse_conf_<TH> ②
Enter passphrase (empty for no passphrase): ③
Enter same passphrase again:
Your identification has been saved in id_cgse_egse_server_inta
Your public key has been saved in id_cgse_egse_server_inta.pub
The key fingerprint is:
SHA256:k0UBdc*****7NSgd1WX+Ds plato-data@egse-server-inta
The key's randomart image is:
```

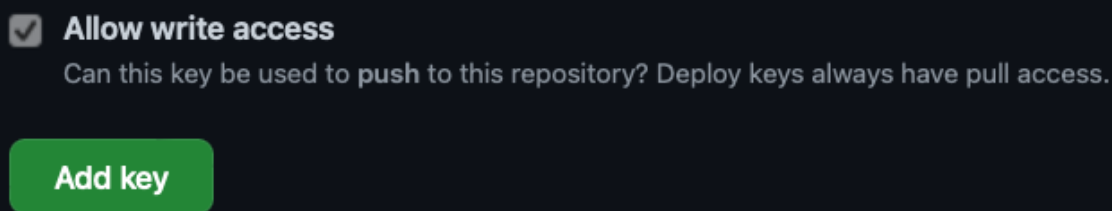
```

+--[ED25519 256]--+
| ..000+*00=..+.+|
| . .+. =++0..00|
| . 0 0 00+0= .|
| . = .00= =.|
| . + S .o B +|
| 0          = E |
| 0          . .|
| 0          |
| .          |
+-----[SHA256]-----+

```

- ① The email address can be `plato-data@egse-server-<th>` where `<th>` is your test house site id.
- ② name of the file: `id_cgse_egse_server-<th>`, again `<th>` is the test house site id.
- ③ do not provide a passphrase, just hit return

Send the `id_cgse_conf-<th>.pub` file to the maintainer of the GitHub repository. She will copy the content of this file into a new deploy key for the `plato-cgse-conf` @ GitHub. It is basically the same procedure as for the previous two repos, except for `plato-cgse-conf` using the configuration manager: when you add the deploy key to GitHub, you must check the *Allow write access* checkbox. That will allow the configuration manager to upload new Setups to the repo.



Now we need to create a generic hostname for the repository such that this can be picked up by the ssh protocol when accessing the repository at GitHub. Add the following lines to the file `~/.ssh/config`:

```

Host repo-cgse-conf
  Hostname github.com
  IdentityFile ~/.ssh/id_cgse_conf-<th> ①

```

- ① don't forget to use your test house site id

Check the permissions of the `~/.ssh` directory and the files in it (see [Section 14.1](#)).

If you have not yet cloned the `plato-cgse-conf` repository, you can do that now with the following command:

```
$ cd ~/git/
```




```
$ git clone git@repo-cgse-conf:IvS-KULeuven/plato-cgse-conf.git
```

Alternatively, you can add a new remote to your git repository. This is needed to (1) use the generic hostname created above, and (2) use a standard name for the remote that is used by the update/upload script. Add a remote for doing the updates as follows:

```
$ cd ~/git/plato-cgse-conf  
$ git remote add updates git@repo-cgse-conf:IvS-KULeuven/plato-cgse-conf.git
```

For the configuration manager, we will also add a remote **upload** which is needed by the configuration manager when submitting a new Setup.

```
$ git remote add upload git@repo-cgse-conf:IvS-KULeuven/plato-cgse-conf.git
```

Make sure that the following environment variable is defined in **/cgse/env.txt**:

```
export PLATO_CONF_REPO_LOCATION=/home/plato-data/git/plato-cgse-conf
```

Make sure that the branch has the upload/main as its tracking branch:

```
$ git branch -u upload/main  
$ git branch -vv  
* main 17fb23c [upload/main] change filter wheels parameters ①
```

① between square brackets is the remote/branch that is tracked.

Chapter 15. Update the Common-EGSE to the latest release

At some point you will be asked to update to a specific release. Make sure you are in the develop branch, then execute the following commands:

```
$ cd ~/plato-common-egse  
$ update_cgse ops --tag=2022.3.4+CGSE ①
```

① as a reminder, the release tag for the Common-EGSE takes the following form:
YYYY.MAJOR.MINOR+CGSE

You can check if the correct version is installed as follows:

```
$ python3 -m egse.version  
CGSE version in Settings: 2022.3.4+CGSE  
CGSE installed version = 2022.3.4+cgse
```



Chapter 16. Update the Test Scripts to the latest release

When you need to update the test scripts on your `egse-client` machine, use the following commands:

```
$ cd ~/git/plato-test-scripts  
$ update_ts
```

We do not have an up-to-date release strategy yet for the test scripts. The command above will install the latest version from the develop branch. Therefore, only update the test scripts when a new release is created on the GitHub repository. That will assure the updates have at least been verified and reviewed.

To know the version of the test scripts that is installed on your machine, use the following command:

```
$ python3 -m camtest.version  
CAMTEST version in Settings: 2022.3.6+TS  
CAMTEST git version = 2022.3.6+TS-0-g602f139 ①
```

① The version that is presented here is explained in the developer manual in [Version Numbers](#).

Chapter 17. Update Python packages

Ideally, the installed third party packages should be the versions that are given the requirements file of the project. If the requirements file is updated, you can use the following command to update your installation:

```
$ cd $PLATO_COMMON_EGSE_PATH  
$ python3 -m pip install --upgrade --prefix=/cgse -r requirements.txt ① ②
```

- ① note the `--prefix` to make sure the upgrade of the packages is done in the correct location and not in the system folders. Instead of `/cgse` you can also use `$PLATO_INSTALL_LOCATION` for the `--prefix` value.
- ② on Ubuntu, the system Python installation has packages installed in `/usr/lib/python3/dist-packages` and that location is not writable for the 'plato-data' user. When you get an error during installation, add the option `--ignore-installed` to the `pip` command.

To update the Python packages for the test scripts, make sure you are inside the virtual environment:

```
$ cd ~/git/plato-test-scripts  
$ source venv/bin/activate ①  
$ python3 -m pip install --upgrade -r requirements.txt
```

- ① it's important that you are in the virtual environment before performing the upgrade, then the packages will be installed—as intended—in your virtual environment.



Chapter 18. Data Propagation

- ☐ Shortly describe the storage strategy (refer to developer manual section for more detail)
- ☐ Refer to section about disk organisation
- ☒ describe rsync from /data to /archive
- ☒ describe rsync to Leuven
- ☒ crontab examples

The following line is a crontab entry for syncing the /data life data storage to the /archive permanent storage every 15 minutes.

```
# Synchronise /data to /archive
*/15 * * * * rsync -av /data/ /archive/
```

The next line is a crontab entry for syncing the /data folder from IAS to the KU Leuven archive. It will update destination files in place and exclude FITS image files because they are intermediate files before generating the FITS cubes and do not need to be archived. The -rloptD options represent the archiving mode and the -x option is to prevent rsync to cross filesystem boundaries.

```
# Synchronise /data to KU Leuven
*/15 * * * * rsync -rloptDv --chmod=Dg+s,Dug=rwx,Do=rx,Do-w,Fug=rw,Fo=rx,Fo-w
--inplace --exclude '_images.fits' -x /data/
ias@copernicus.ster.kuleuven.be:/STER/platodata/IAS/data/
```

The similar rsync command for syncing the /archive folder to KU Leuven is given below:

```
# Synchronise /archive to KU Leuven
*/15 * * * * rsync -rloptDv --chmod=Dg+s,Dug=rwx,Do=rx,Do-w,Fug=rw,Fo=rx,Fo-w
--inplace --exclude '_images.fits' -x /archive/
ias@copernicus.ster.kuleuven.be:/STER/platodata/IAS/archive/
```

The above examples are stripped from logging commands to focus on the relevant parts of the rsync command. A full crontab entry for the synchronisation of /data to KU Leuven is given below for completeness.

```
*/15 * * * * echo "-----" >>
/home/plato-data/logs-rsync-data-to-KU-Leuven ; date >> /home/plato-data/logs-rsync-
data-to-KU-Leuven ; rsync -rloptDv --chmod=Dg+s,Dug=rwx,Do=rx,Do-w,Fug=rw,Fo=rx,Fo-w
--inplace --exclude '_images.fits' -x /data/
ias@copernicus.ster.kuleuven.be:/STER/platodata/IAS/data/ 2>&1 >> /home/plato-
```



data/logs-rsync-data-to-KU-Leuven



Chapter 19. Shared Libraries

Some components use a shared library that is loaded by the Python interpreter. In order to load the library, the absolute path to the shared object file must be known. Different modules handle this in their own way.

The `egse.dsi` module searches for the RMAP and ESL libraries in the operating system specific folder in the `egse.lib` module. The libraries are then loaded using the `ctypes` module. If the CGSE is properly installed, this should work out-of-the-box.

The `egse.filterwheel.eksma` module needs a library `libximc` which is also provided in the `egse.lib` module, but the Python code needs the proper location in the environment variable `LD_LIBRARY_PATH`. The required files are included in the CGSE repo at `~/git/plato-common-egse/src/egse/lib/ximc/libximc.framework`. The library that is needed can also be downloaded from: <https://files.xisupport.com/libximc/libximc-2.13.3-all.tar.gz>.

The preferred solution is to add the location of the library files to the environment variable `$LD_LIBRARY_PATH`. In your terminal or better in your bash profile:

```
$ export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:~/git/plato-common-egse/src/egse/lib/ximc/libximc.framework/
```

Alternatively, you can install the '.deb' package from the link above on your system with `dpkg`. That will put the files under `/usr/lib`.

Chapter 20. Installing External Tools

20.1. Cutelog GUI

Cutelog is a GUI that can be installed in your virtual environment using **pip**.

```
$ python -m pip install cutelog
```

You can use this application to inspect the log messages on the **egse-client** machine. Start the GUI from the process manager GUI (**pm_ui**) or from a terminal.

cutelog

FileTabServerRecordsHelp

Log namespaces

egse

procman

procman_ui

process

resource

ds1

esl

rmap

config

settings

protocol

tcs

tcs_devif

tcs_protocol

sockets

storage

Time	Name	Level	Message
2022-04-04 17:02:25.157064	egse.procman.procman_ui	INFO	Start the TCS Control Server
2022-04-04 17:02:25.157133	egse.procman	DEBUG	Starting TCS
2022-04-04 17:02:26.145537	egse.procman	INFO	Starting Control Server for TCS in operational mode
2022-04-04 17:02:26.145652	egse.process	DEBUG	SubProcess command: ["/usr/bin/python3", "-m", "egse.tcs.tcs_cs", "start"]
2022-04-04 17:02:26.157772	egse.process	DEBUG	SubProcess started: ["/usr/bin/python3", "-m", "egse.tcs.tcs_cs", "start"], pid=1559840, sub_process=psutil.Process(pid=1559840, name="sh", status="running", started="17:02:25")
2022-04-04 17:02:26.768375	egse.resource	DEBUG	Resources have been defined: DEFAULT_RESOURCES={"icons": "/icons", "images": "/images", "lib": "/lib", "styles": "/styles", "data": "/data", "auidata": "/aui/arbdata"}
2022-04-04 17:02:27.124382	egse.ds1.esl	DEBUG	Locating shared library EtherSpaceLink_v34_86.dylib in dir "/lib/CentOS-8"
2022-04-04 17:02:27.124458	egse.config	INFO	no git repository found, assuming installation from distribution.
2022-04-04 17:02:27.124931	egse.config	DEBUG	Common-EGSE location is automatically determined: /cgse/lib/python/Common_EGSE-2022.1.10_SRON_CGSE-py3.8.egg/egse
2022-04-04 17:02:27.126719	egse.ds1.esl	DEBUG	Loading shared library: /cgse/lib/python/Common_EGSE-2022.1.10_SRON_CGSE-py3.8.egg/egse/lib/CentOS-8/EtherSpaceLink_v34_86.dylib
2022-04-04 17:02:27.131302	egse.ds1.rmap	DEBUG	Locating shared library ESL-RMAP_v34_86.dylib in dir "/lib/CentOS-8"
2022-04-04 17:02:27.132681	egse.ds1.rmap	DEBUG	Loading shared library: /cgse/lib/python/Common_EGSE-2022.1.10_SRON_CGSE-py3.8.egg/egse/lib/CentOS-8/ESL-RMAP_v34_86.dylib
2022-04-04 17:02:27.153946	egse.settings	DEBUG	Parsing YAML configuration file /cgse/lib/python/Common_EGSE-2022.1.10_SRON_CGSE-py3.8.egg/egse/storage/storage.yaml
2022-04-04 17:02:27.160077	egse.procman	WARNI...	Could not start Control Server for TCS
2022-04-04 17:02:27.161119	egse.settings	DEBUG	Parsing YAML configuration file /cgse/lib/python/Common_EGSE-2022.1.10_SRON_CGSE-py3.8.egg/egse/synoptics/syn.yaml
2022-04-04 17:02:27.164352	egse.settings	DEBUG	Parsing YAML configuration file /cgse/lib/python/Common_EGSE-2022.1.10_SRON_CGSE-py3.8.egg/egse/tcs/tcs.yaml
2022-04-04 17:02:27.193535	egse.settings	DEBUG	Parsing YAML configuration file /cgse/lib/python/Common_EGSE-2022.1.10_SRON_CGSE-py3.8.egg/egse/services.yaml
2022-04-04 17:02:27.210245	egse.protocol	DEBUG	Creating ServiceCommand command with name="set_monitoring_frequency", cmd="[{delay}]", device_method=None
2022-04-04 17:02:27.210604	egse.protocol	DEBUG	Creating ServiceCommand command with name="set_hk_frequency", cmd="[{delay}]", device_method=None
2022-04-04 17:02:27.210903	egse.protocol	DEBUG	Creating ServiceCommand command with name="set_logging_level", cmd="[{name}]", device_method=None
2022-04-04 17:02:27.210906	egse.protocol	DEBUG	Creating ServiceCommand command with name="quit_server", cmd="", device_method=None
2022-04-04 17:02:27.211096	egse.protocol	DEBUG	Creating ServiceCommand command with name="get_process_status", cmd="", device_method=None
2022-04-04 17:02:27.211257	egse.protocol	DEBUG	Creating ServiceCommand command with name="get_cs_module", cmd="", device_method=None
2022-04-04 17:02:27.211461	egse.protocol	INFO	Binding to tcp://-6607
2022-04-04 17:02:27.211652	egse.protocol	INFO	Binding to tcp://-6606
2022-04-04 17:02:27.223111	egse.tcs.tcs	DEBUG	Initializing TCSController with hostname=192.168.80.4 on port=6666
2022-04-04 17:02:27.223335	egse.tcs.tcs_devif	DEBUG	Connecting a socket to host "192.168.80.4" using port 6666
2022-04-04 17:02:27.256030	egse.sockets	DEBUG	Connecting a socket to host "192.168.80.4" using port 6667
2022-04-04 17:02:27.261465	egse.storage	ERROR	Could not register TCS-HK: An item with name 'TCS-HK' is already registered, please unregister first.
2022-04-04 17:02:27.261974	egse.storage	INFO	TCS-HK is already registered.
2022-04-04 17:02:27.262285	egse.tcs.tcs	WARNI...	Format error: no new housekeeping values received.
2022-04-04 17:02:27.265014	egse.tcs.tcs	INFO	No response received from the get_Conf command to TCS EGSE.
2022-04-04 17:02:27.265763	egse.storage	ERROR	Could not register TCS: An item with name 'TCS' is already registered, please unregister first.
2022-04-04 17:02:29.265547	egse.sockets	DEBUG	Disconnecting from 192.168.80.4
2022-04-04 17:02:37.308508	egse.tcs.tcs_protocol	ERROR	Exception caught: exc=DeviceConnectionError('TCS EGSE', 'Socket communication error:')
2022-04-04 17:02:37.309080	egse.tcs.tcs_protocol	DEBUG	errors=None
2022-04-04 17:02:37.309129	egse.tcs.tcs_protocol	DEBUG	new_errors=None

Levels

Show

Name

✓

DEBUG

✓

INFO

✓

WARNING

✓

ERROR

✓

CRITICAL

Name	Value
------	-------

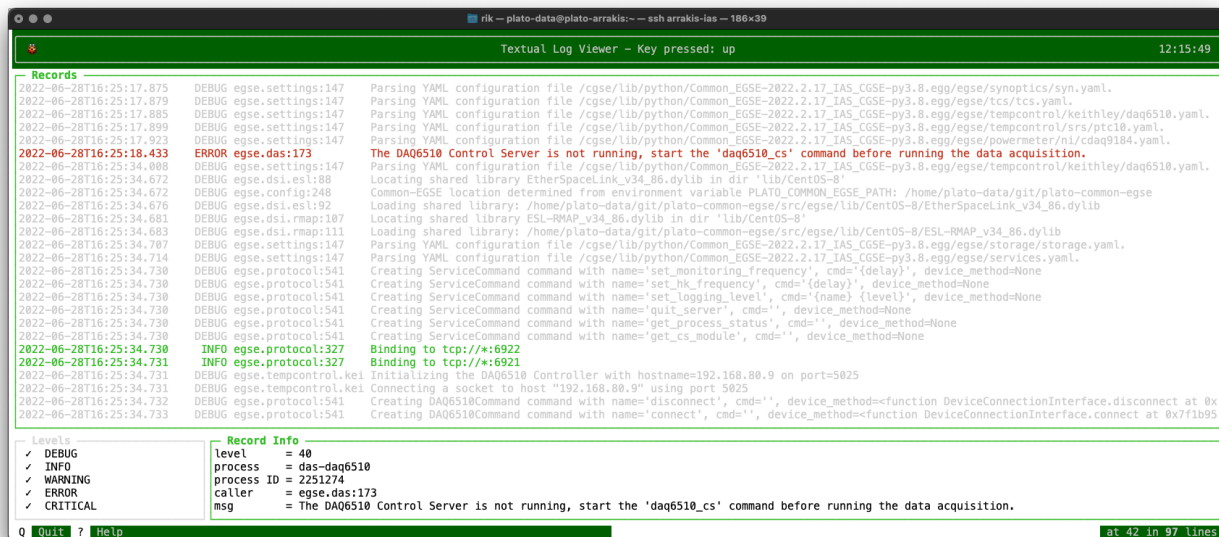
20.2. Textualog TUI

Textualog is a terminal application (TUI) that allows you to inspect the log files in **\$PLATO_LOG_FILE_LOCATION**. The package is open-source and can be installed in your virtual environment from PyPI using **pip**:

```
$ python -m pip install textualog
```


Textuallog is extremely useful to inspect and follow logging messages in a remote terminal. It is inspired on the **cutelog** app and developed specifically for the remote users. After installation, the current log file can be inspected with the following command^[1]:

```
$ textuallog --log $PLATO_LOG_FILE_LOCATION/general.log
```



[1] the textuallog package might already installed on the egse-server at the test houses.