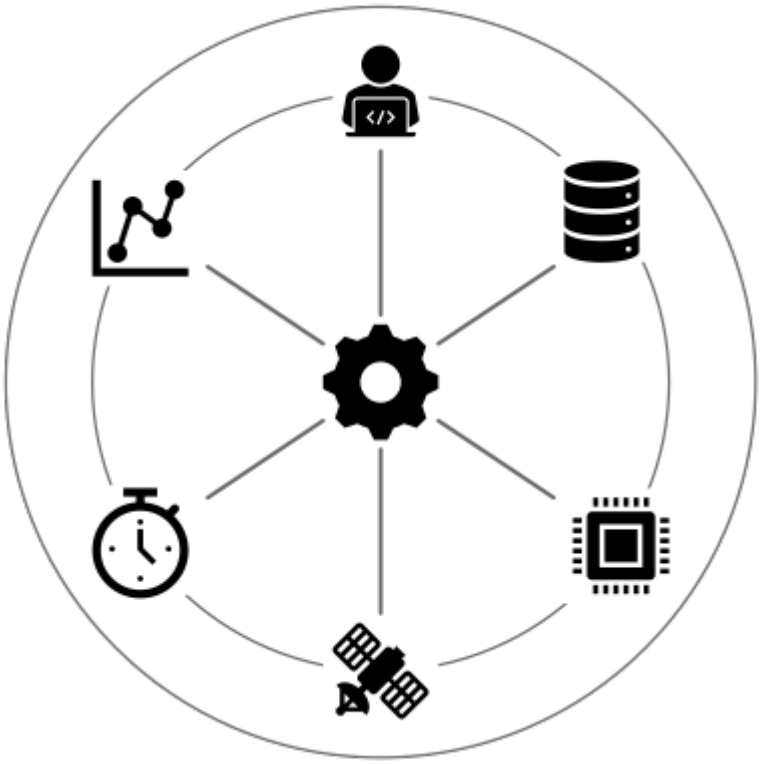


Common–EGSE & Test Scripts

User Manual

Rik Huygen, Sara Regibo

Version 1.1, 03/11/2022

Table of Contents

Chapter 1. Abstract

This document is the user manual for the PLATO Common-EGSE and Test Script. The software is used at CSL and the test houses at IAS, INTA and SRON. The document describes all user interactions with the different components of the Common-EGSE system and how to run the different test scripts to perform camera testing.

This manual assumes the system has been installed and is properly configured and will therefore not explain configuration settings nor installation procedures. Please refer to the installation manual for this information.

The Common-EGSE framework and the Test Scripts are part of the PLATO Camera Alignment and Testing program. The software is developed in-house in the Python 3 programming language and runs on the Linux OS. The Common-EGSE is a distributed system with (micro-)services and controllers for hardware devices that make up the ground test equipment. Camera testing is performed by executing test scripts in a Python 3.8+ environment following the *as-run* test procedures.

Chapter 2. TODO

What can be a good structure for this manual? I was thinking about make chapters per topics and not so much per component. Something like:

```
Starting up
  Starting the core egse services
  The Process Manager
Monitoring the system
  Using device GUIs
  Using Grafana
Running test scripts
  The as-run procedure
  Using PyCharm or the test house GUIs
Inspecting the data
  Location of housekeeping and image data
```

- Try to go through it with a user hat on — what is a user? operator/developer/sysadmin?
  - installation procedure: be clear about it, which installation is for the normal users/developers/sysadmins?
    - using modules or .bash_profile to set up the environment?
  - Inspection of the Settings & local settings
    - python -m egse.settings
    - python -m egse.setup --use-cm
    - GUI ?
  - starting the different core services:
    - core services: invoke start-core-egse for users & developers? Systemd for sysadmins
    - how to check if these core services have been started and are working properly?
      - invoke status-core-egse or status for individual core services, e.g. log_cs status
      - cm_cs status
- What about the GUIs of the core services?
  - log_cs has no GUI yet→ yes it does, cutelog and Textualog
  - sm_cs has no GUI yet
  - cm_cs has a GUI cm_ui can not start when cm_cs is not running
- dpu_cs
  - should be able to run as a daemon server, be able to connect/reconnect to N-FEE when needed/requested
  - start it in the background?

- What shall go into the UM and what in the DM?
- Installation for the user:
  - git clone from the IvS-KULeuven repo
  - `python setup.py develop`
  - `python -m pip install -e .`
- What is the focus of the user with respect to CGSE?
  - GUIs
- Status Overview of the System
  - How can we see what is running/active/ready…
    - commandline
    - GUIs

Chapter 3. Documents and Acronyms

3.1. Applicable documents

[AD-01] PLATO Common-EGSE Requirements Specification, PLATO-KUL-PL-RS-0001, issue 1.4, 18/03/2020

[AD-02] PLATO Common-EGSE Design Description, PLATO-KUL-PL-DD-0001, issue 1.2, 13/03/2020

[AD-03] PLATO Common-EGSE Interface Control Document, PLATO-KUL-PL-ICD-0002, issue 0.1, 19/03/2020

3.2. Reference Documents

[RD-01] PLATO Common-EGSE Installation Guide, PLATO-KUL-PL-MAN-0002

[RD-02] PLATO Common-EGSE Developer Manual, PLATO-KUL-PL-MAN-0003

[RD-03] Common-EGSE on-line Documentation https://ivs-kuleuven.github.io/plato-cgse-docs/

3.3. Acronyms

AEU	Ancillary Electronics Unit
API	Application Programming Interface
CAM	Camera
CGSE	Common-EGSE
COT	Commercial off-the-shelf
DPU	Data Processing Unit
DSI	Diagnostic SpaceWire Interface
EGSE	Electrical Ground Support Equipment
OS	Operating System
PID	Process Identifier
PPID	Parent Process Identifier
PLM	Payload Module
REPL	Read-Evaluate-Print Loop, e.g. the Python interpreter prompt
RMAP	Remote Memory Access Protocol
SpW	SpaceWire

SVM	Service Module
TBC	To Be Confirmed
TBD	To Be Decided or To Be Defined
TBW	To Be Written
TS	Test Scripts
TV	Thermal Vacuum
USB	Universal Serial Bus

Chapter 4. Introduction

The Common-EGSE consists of several components, each with specific responsibilities, that communicate over a ZeroMQ messaging network protocol. The core functionality of the CGSE are what we call *core services*:

- Configuration Manager: manages and keeps track of the configuration of the test Setup and test equipment.
- Storage Manager: manages and stores all camera test data, i.e. CCD images, housekeeping data, monitoring data.
- Process Manager: manages and keeps track of all running processes that are used by the test site.
- Synoptics Manager: manages the synoptic housekeeping
- Logging Manager: stores and distributes all logging messages from all components in the system

The core services are independent of the test house or test equipment that is used and are background processes that are started by the Linux Systemd service during startup of the CGSE server. No user interaction is need with the core services as they are managed by Systemd to restart when they crash or are stopped. You should be confident that the core services are always running.

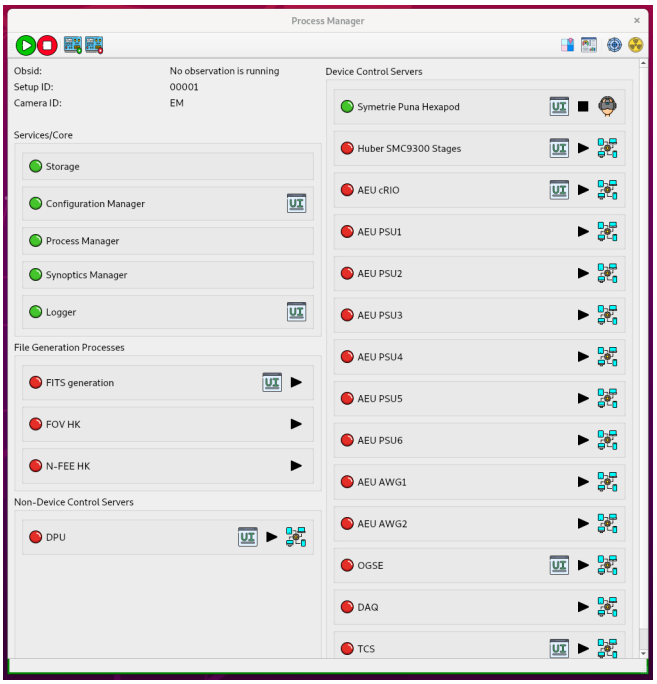
4.1. The User Perspectives

Although this manual is intended for operators, i.e. those users that operate the system, perform the tests, and execute the procedures, some sections will be dedicated to administrators or developers when there is a need in the context of the section. Please note that this user manual is complementary to the PLATO Camera Ground Test Commanding Manual [PLATO-KUL-PL-MAN-0004] with only minimal overlap.

4.2. Where do I start?

Since we assume the Common-EGSE and Test Scripts have been installed and configured, the starting point for you is the Process Manager. The Process Manager (PM) provides a GUI which is started on the egse-client and looks much like the screenshot below. The PM GUI can be started either by clicking its icon on the desktop or from a terminal with the `pm_ui` command.

```
$ pm_ui
```



The Process Manager GUI represents the state of the system. It provides information about all the processes that are needed to perform the camera tests. These processes are divided in four categories: Core services, File Generation Processes, Non-Device Control Servers, and Device Control Servers. The screenshot below is taken from CSL and the GUI only shows processes that are applicable to CSL. The left column of services and processes are common for CSL and all the test houses. The right column with Device Control Servers is test house specific and is defined by the content of the Setup. [XXXXX explain what the Setup is and where more info can be found].

The red and green bullets in the PM GUI describe the state of the service or process. When the bullet is green the process is running as expected, when the bullet is red the process is not running. For some devices the bullet can become orange to indicate that the control server process is running, but the device is not connected and can therefore not be commanded. The icons at the right of the processes are for starting/stopping the process. Some processes can be started in simulator mode (toggle the device icon) and some of the processes have a GUI associated that can be started from the PM GUI. Please note that none of the core services can be started from the Process Manager GUI. Core services are managed by the Systemd services of your Linux system and when stopped or crashed they will be automatically restarted.

The Process Manager is fully explained in its own section *The Process Manager GUI* in Section 7.2.

The Process Manager presents you with a view of the state of the system and allows you to start and stop device processes and start some GUIs to manipulate certain devices and services. The main task for an operator to perform camera tests or alignment procedures is however to execute the test procedure as described in the TP-0011 Google Sheet for each of the test houses and in the ambient alignment procedure PR-0011 for CSL. These documents contain step-by-step as-run procedures that describe all manual interactions, tasks and code execution needed to successfully perform a camera test. Each of the rows in these procedures describe one action that needs to be successfully finished before proceeding to the next action. As said, this can be a manual interaction with a device, a task to be executed from a specific GUI, or a code snippet that needs to be executed in the dedicated environment. The Python source code from the as-run

procedure will be executed in a Python Console that is used throughout the completion of the as-run procedure.

We recommend to use the Python Console provided by the PyCharm IDE. [XXXXX What about the Qt Python Console that is available from the e.g. CSL Commanding GUI?]

4.3. Client vs. Server

Your server is a powerful machine with one or two CPUs and a few tens of cores. This was chosen because we will have a lot of applications/services running, and they can take up different cores without being interrupted. The server also has several disks that are used for active data storage and data archive. The active data storage needs to be a fast SSD disk with enough space to store the data of a few test days. The disk needs to be fast because of the image data that is received from the camera. When the image data is not written fast enough the camera front-end electronics might generate a buffer over-full error hereby corrupting the image data for the current exposure. The archive disk is a normal SATA disk that can hold the test data for the whole campaign.

The client machine is a small desktop computer with preferably two large screens to place all the GUIs and the browser for commanding and monitoring the tests.

4.3.1. What should run on the Server?

The following processes should run on the server:

- The core services (Storage Manager, Configuration Manager, Process Manager, Synoptics Manager, and Logger), which are (re-)started automatically via `systemd`;
- All control servers (incl. device control servers, FDIR, Alert Manager);
- ALl file generation processes (FITS generation, FOV HK, and N-FEE HK).

These processes can be started in two different ways:

- From the PM UI (see the Sect. below), which is running on the client;
- Directly on the command line on the server.

4.3.2. What should run on the Client?

The following things should be started on the client:

- The test scripts, executed in a Python interpreter (PyCharm or the Operator GUI)
- All GUIs. These can be started from the command line or from the PM UI.

NOTE	All processes (not the GUIs) that are started from the PM UI are actually started on the server by the Process Manager. When you press a start button on the PM UI, you send a request to the Process Manager to start the service. Remember the Process Manager itself is running on the server.

Chapter 5. Update the Common-EGSE Software

```
update_cgse develop
```

To update the Common-EGSE on the operational machine, use the `ops` argument instead. An operational installation is different from a developer installation. There is no virtual environment and the all required Python packages, including the Common-EGSE are installed at a specific location. The installation process makes use of the following environment variables:

- `PLATO_COMMON_EGSE_PATH`: the location of the plato-common-egse repository on your machine, e.g., `~/git/plato-common-egse`
- `PLATO_INSTALL_LOCATION`: the location where the packages shall be installed, usually `/cgse`

```
update_cgse ops --tag=2022.2.2-<TH>-CGSE
```

The `<TH>` must be replaced with the test house name, CSL, IAS, INTA, or SRON.

Chapter 6. Starting the Core Services

The *core services* of the Common-EGSE are those services that should be running all the time and preferably on the dedicated EGSE server. In the production environment, i.e. when we are actually running the tests in the lab, these services will be started automatically when the system boots, see the section on the core services in systemd. During our development and when using the Common-EGSE outside of the Camera Test environment, we can start the core services on our local machine.

When the system is properly installed, you should have an `invoke` command available from the terminal. When you run it with the `--list` option, it will show which commands are available. Make sure you execute this command in the CGSE project folder.

```
$ cd ~/git/plato-common-egse
$ invoke --list

Available tasks:

  pytests           Run the tests for this project.
  start-core-egse   Start the CGSE core services.
  stop-core-egse    Stop the CEGSE core services.
  status-core-egse  Print the status information of the CGSE core services
```

Some of these commands are for development purposes, but the `start-core-egse` and the `stop-core-egse` commands can be used to start/stop the core services.

```
$ invoke start-core-egse
Starting log manager..
Starting storage manager..
Starting configuration manager..
Starting process manager..
Starting synoptics manager..
```

The servers are started in a background job.[1] To see if they are indeed started, you can run the following in your terminal:

```
$ ps -ef|grep _cs
459800007 64147     1   0  2:25PM ttys001    0:00.42
/Library/Frameworks/Python.framework/Versions/3.8/Resources/Python.app/Contents/MacO
S/Python /Users/rik/Git/plato-common-egse/venv38/bin/log_cs start
459800007 64148     1   0  2:25PM ttys001    0:02.73
/Library/Frameworks/Python.framework/Versions/3.8/Resources/Python.app/Contents/MacO
S/Python /Users/rik/Git/plato-common-egse/venv38/bin/sm_cs start
459800007 64162     1   0  2:25PM ttys001    0:07.90
/Library/Frameworks/Python.framework/Versions/3.8/Resources/Python.app/Contents/MacO
```

```
S/Python /Users/rik/Git/plato-common-egse/venv38/bin/cm_cs start
459800007 64171     1   0  2:25PM ttys001    0:03.14
/Library/Frameworks/Python.framework/Versions/3.8/Resources/Python.app/Contents/MacO
S/Python /Users/rik/Git/plato-common-egse/venv38/bin/pm_cs start
459800007 64180     1   0  2:25PM ttys001    0:02.94
/Library/Frameworks/Python.framework/Versions/3.8/Resources/Python.app/Contents/MacO
S/Python /Users/rik/Git/plato-common-egse/venv38/bin/syn_cs start
```

Keep the core services running, they do not harm and other components of the Common-EGSE need these services. But, if you want to stop them anyway at the end of a long working day, just use:

```
$ invoke stop-core-egse
```

[1] After the storage manager started, there will be a short delay of 2 seconds, this allows the service to fully start and initialise before the other services (confman & procman) try to register to the storage.

Chapter 7. The Graphical User Interfaces (GUI)

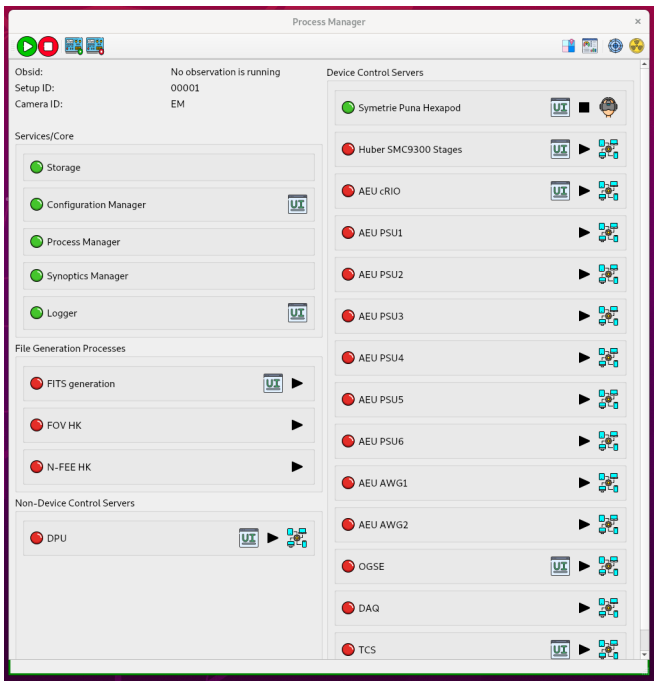
7.1. Icons used for different GUIs

Component	icon	Command	Comment/description
Process Manager GUI		`pm_ui`	
Configuration Manager GUI		`cm_ui`, `setup_ui`	this logo is also usedfor the Setup GUI
CSL Commanding GUI		`csl_ui`	
Contingency GUI		`contingency_ui`	
PUNA Hexapod GUI		`puna_ui`, `mech_pos_ui`	this logo is also used for the Mechanical Positions GUI, used only on position 'M' in the CSL cleanrooms
ZONDA Hexapod GUI		`zonda_ui`	
Gimbal GUI		`gimbal_ui`	
HUBER Stages GUI		`smc9300_ui`	
OGSE GUI		`ogse_ui`	only used at CSL
TCS GUI		`tcs_ui`	only used in the test houses
AEU GUI		`aeu_ui`	
DPU GUI		`dpu_ui`	
FITS Generation GUI		`fitsgen_ui`	
FOV GUI		`fov_ui`	
Visited Positions GUI		`visited_positions_ui` | `vis_pos_ui`	

Component	icon	Command	Comment/description
Power Meter GUI		`pm100a_ui`, `cdaq9184_ui`	

7.2. The Process Manager GUI

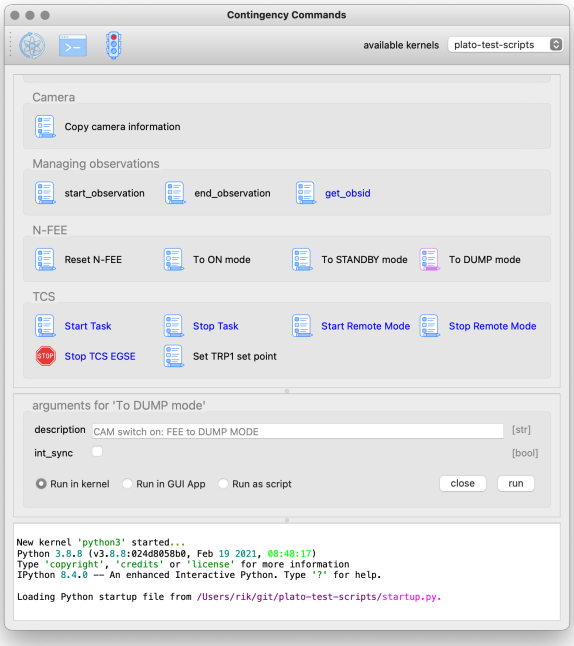
This section describes the Process Manager GUI.



7.3. The Contingency GUI

This section describes the Contingency Tasks GUI. The GUI is a Task GUI based on the `gui-executor` Python package. The GUI has the following components (aka panels).

- A toolbar for managing the Python kernel and opening a Python Console window.
- A panel containing all the tasks. The tasks are grouped by type or component, e.g. tasks for the N-FEE or the TCS or tasks that manage observations. The panel is scrollable when more groups or tasks are available than can fit in the available space.
- An arguments panel that is shown when a task is selected (in the figure below that is `set_trp1`) and which allows you to enter parameters that will be passed into the task. You can also specify here how you want the task to be run.
- An output panel where the output of the task will appear.



The idea of the Contingency Task GUI is to provide you with a clear overview of the contingency tasks, a simple way to customize the task by specifying parameters and run the task with immediate feedback all in the same GUI app. A contingency task is an action that is needed to recover from a failure state or to put the system into a defined state.

Let's explain what happens when you execute the tasks.

As an example, if you press the `set_trp1` task button, it will change color to indicate this task has been selected (see screenshot above) and an arguments panel will appear in the middle of the GUI. The `set_trp1` task expects one parameter (`temperature`) for which no default was provided. The expected input is a float. When you press the `Run` button, the task will be executed *in the kernel*. All tasks are by default executed in the kernel. You will sometimes see that a task will execute in the GUI App or as a script, don't use those options yourself unless you know what you are doing.

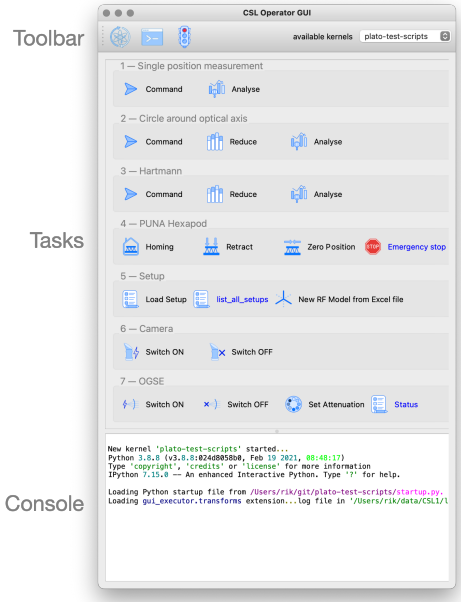
For more information on the working of the Task GUIs, please go and read the section on the Tasks GUI in Chapter 8.

7.4. The CSL Operator GUI

This section briefly describes the CSL specific task GUI, the *CSL Operator GUI*.

The intent of the *CSL Operator GUI* is to simplify the work of the test operator at CSL when executing routine tasks to perform the alignment of the PLATO Cameras. At the top of the GUI you will find the tasks for taking a single position measurement, a circle around the optical axis, and a Hartmann measurement. Each of these task groups has a task to do the actual commanding and a task to reduce and analyse the data. Then, there are tasks that are grouped by components like the PUNA Hexapod, the Camera, OGSE, etc. Each of these groups contain tasks to perform a specific action on that component. When you hover over the tasks with your mouse, the

documentation for that task will show up in a tooltip. Alternatively, you can right click on a task and select *View source...* to inspect the source for for the task.



For more information on the working of the Task GUIs, please go and read the section on the Tasks GUI in Chapter 8.

7.5. The Hexapod Puna GUI

7.5.1. Monitoring and Commanding the Hexapod

Monitoring and commanding the hexapod can be done via a designated GUI. This is described in the sections below.

7.5.2. Synopsis

To start the Hexapod Puna GUI, type the following command:

```
puna_ui --type [simulator/proxy/direct]
```

Arguments

The Hexapod GUI should be started with the following arguments:

`--type`

The hexapod implementation you want to connect to. The options are `simulator`, `proxy`, and `direct`. The `simulator` option starts the GUI in simulation mode, which means that instead of connecting to the real hardware, the GUI communicates with a simulator. This option is mainly used for testing or demonstration when no hardware is available. The `direct` option connects the GUI directly to the hardware controller without a control server in between. The

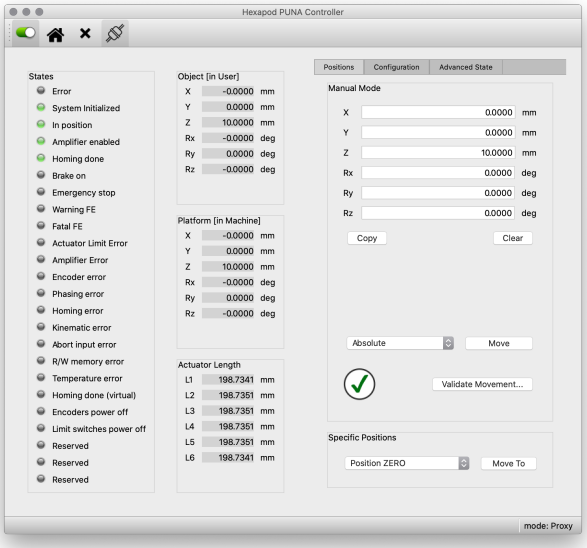
`proxy` option is the default and connects the GUI to the Puna Control Server allowing other processes to connection simultaneously for monitoring.

`--profile`

Optional. Profile for logging.

7.5.3. Description

A screenshot of the Hexapod Puna GUI is shown below.



We discern the following components in the GUI:

- the [toolbar](#toolbar),
- the left panel, displaying the [status](#states) of the hexapod,
- the middle panel, with the [user and machine positions, and actuator lengths](#positions),
- and the right panel with the [tabs that allow settings, movements, maintenance, etc.](#tabs)

7.5.4. Toolbar

Enable/Disable Amplifier

The first button, the switch, is used to enable/disable the amplifier, i.e. to activate/de-activate the control loop of the motors.

Homing

The second button in the toolbar, with the little house, currently does not work yet. It will be used in the future to move the hexapod back to its homing position.

Clear Errors

The third butting in the toolbar, with the cross, currently does not work yet. It will be used in the future to clear the errors.

Connectivity

The fourth button in the toolbar, with the plug icon, indicates whether or not a connection has been established to the Hexapod Control Server, and can be used to re-connect to or disconnect from it. The connection is handled with the [ZeroMQ](http://zeromq.org/) *request-reply* protocol.

7.5.5. States

The left panel reports on the status of the hexapod. This is done by means of a series of LEDS, where a green LED indicates information, an orange LED indicated a warning, and a red LED indicates an error has occurred.

The meaning of the individual reported states can be found in the Application programming interface (API) of the Hexapod controller (MAN_SOFT_API).

7.5.6. Positions

The middle panel displays the user and machine positions, and actuator lengths.

Object [in User]

At the top the position of the Object Coordinate System is given in the User Coordinate System. The fields `X`, `Y`, and `Z` denote the translation (in mm) along the x-, y-, and z-axis (of the User Coordinate System) resp. The fields `Rx`, `Ry`, and `Rz` denote the rotation (in degrees) around these axes.

Platform [in Machine]

In the middle the position of the Platform Coordinate System is given in the Machine Coordinate System. The fields `X`, `Y`, and `Z` denote the translation (in mm) along the x-, y-, and z-axis (of the Machine Coordinate System) resp. The fields `Rx`, `Ry`, and `Rz` denote the rotation (in degrees) around these axes.

Actuator Lengths

At the bottom, the fields `L1` to `L6` display the lengths (in mm) of the corresponding actuators. Currently the underlying method polling these lengths has not been implemented yet and all actuator lengths are set to NaN.

7.5.7. Tabs

The tabs in the right panel allow settings, movements, maintenance, etc. The different tabs are discussed in the subsequent sections.

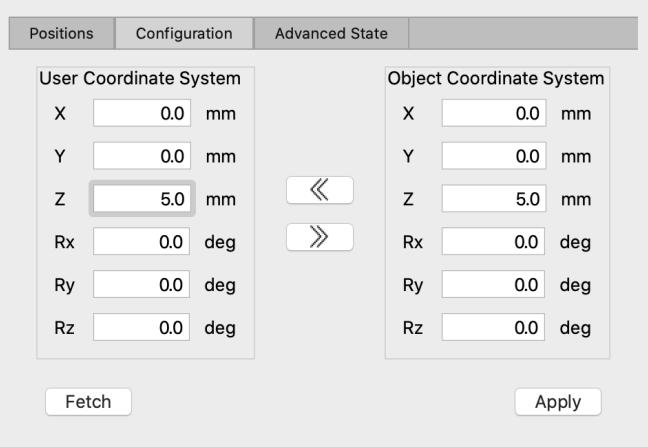
Positions

The first tab, `Positions`, allows to command the hexapod to move to a given position in manual mode. The type of movement, absolute or relative (user \& object), can be selected by the combo box. Before you perform a movement, it is always a good idea to validate. Press the *Validate Movement..* button to send a check command to the Hexapod Controller and return a valid/invalid condition.

There are two specific positions that can be moved to with the combo box at the bottom of this tab, ZERO and RETRACTED.

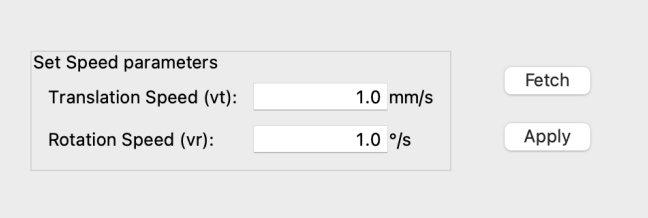
Configuration

The second tab, `Configuration`, allows to manually change the definition of the User coordinate system and the Object coordinate system. The User coordinate system is defined relative to the Machine coordinate system, and the Object coordinate system is defined relative to the Platform coordinate system. This configuration is not saved automatically and will be reset after power-on of the controller.



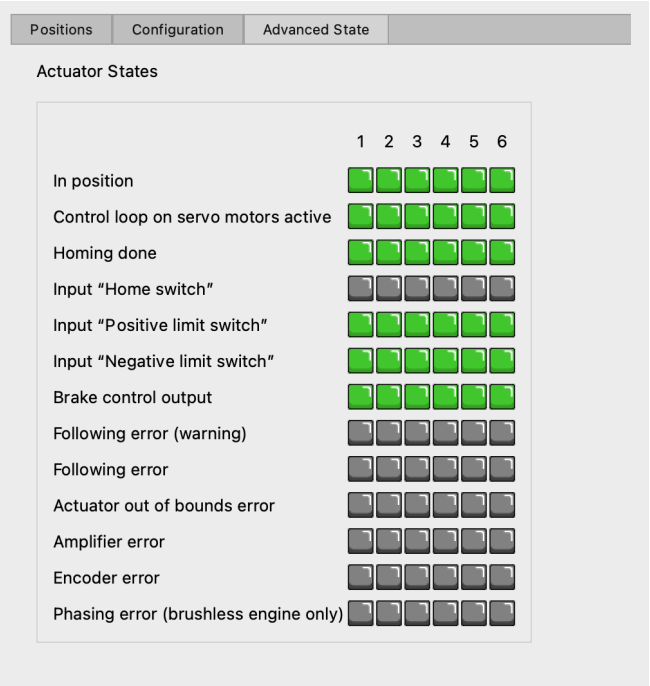
The double arrow buttons in the middle are used to copy the settings from one coordinate system to the other. Use the `Fetch` button to load the settings from the Hexapod controller and when you want to apply your changes, click the `Apply` button.

In the lower part of this tab you will find speed settings. Rotation and translation speed of the hexapod can be set independently. Use the `Fetch` button to retrieve the current speed settings from the hexapod and click the `Apply` button to save your changes to the controller.



Advanced State

The third tab, `Advanced State`, shows the state of each of the actuators of the hexapod.



7.6. The HUBER Stages GUI

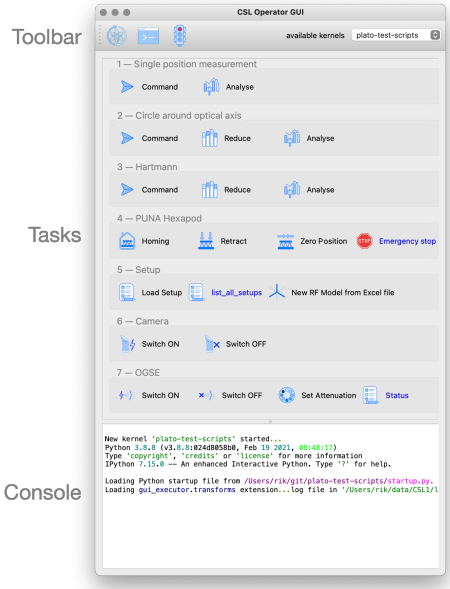
Chapter 8. The Tasks GUI

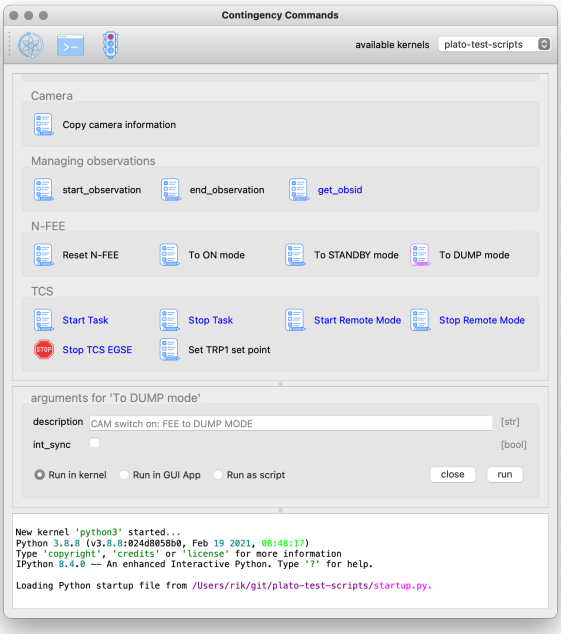
The Tasks GUI is a collective noun for all the task GUIs that we use in our CGSE and TS environment. All these GUIs have the same principle user interface since they are all based on the same Python package that generates the Graphical interface and executes the code. That package is `gui-executor` which is under development at the institute of astronomy at KU Leuven. The `gui-executor` package is open-source and can be installed from PyPI with `pip`:

```
python3 -m pip install gui-executor
```

This chapter explains how the `gui-executor` and therefore the Task GUIs can be used to ease your work in executing scripts and procedures. This chapter is focussed on the user of the GUI, if you are a develop and need coding information on the `gui-executor`, please refer to the *Developer Manual* tasks-gui

Table 1. A few examples of the Task GUI





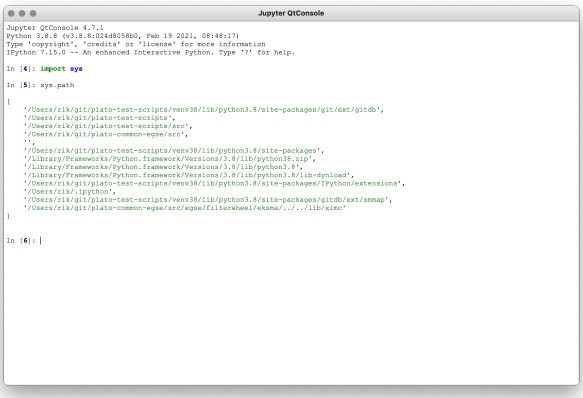
There are four distinct parts in the Task GUI: (1) the toolbar with actions for the Python kernel, (2) the button panel with all the tasks arranged by groups in columns of four tasks., (3) an arguments panel, and (4) the output console.

We will explain all of these panels in more detail next.

8.1. The Toolbar

The toolbar is dedicated to the Python kernel that is used to execute the tasks. The left-most button can be used to restart the kernel. Do this when you want to start a fresh new Python Interpreter or when you need to change the kernel. The second button on the toolbar is used to

open a Python Console that is connected to the currently running kernel. Pressing this button will open a window with a prompt where you can enter or paste Python code to be executed. Here you can also find back the code that was executed by pressing one of the buttons.



In the screenshot on the left, I have entered two lines of Python code, but you see already that the line number starts with `[4:]`. That means I have already executed three blocks or lines of code. Some of that code was executed by the application right after starting the kernel, other code was generated and executed by pressing a button.

You might ask why you would need to change the kernel? Normally, you don't need to do this and you can simply execute code from the Python prompt or by pressing buttons and running tasks. But it might happen that the kernel crashed or hangs due to a bug in the executed code. At that point you would need to restart the kernel. A second reason is when you want to use another kernel from the drop-down menu at the right end of the toolbar. By default, the *plato-test-scripts* kernel will be started if it is available, otherwise the fall-back kernel is *python3*. Please note that only one kernel can be managed from this application, and you can also open only one Python Console window.

8.2. The Button Panel

All tasks are available in the *Button Panel*. The tasks are arranged in groups and in each group in columns of four tasks.

8.3. The Arguments Panel

When you press a task button an associated arguments panel will appear below the button panel. Before pressing the *Run* button you can provide input for all the parameters of the task. Most of the arguments will have a simple builtin type like int, float or bool, but more complex argument types are possible and for some of those a special icon will appear on the right side to

8.4. The Output Console

Chapter 9. Frequently Asked Questions (FAQ)

9.1. How can I check the installed version of the CGSE and Test Scripts

Use the following commands in a terminal:

```
$ python -m egse.version
version: 2022.2.15-IAS-CGSE
release: May 2022, 2022.2.15-IAS-CGSE
```

```
$ python -m camtest.version
CAMTEST Test Scripts version = 2022.1.12-SRON-TS-37-ga4f04cd
```

NOTE

The Common-EGSE is installed on both the egse-server and the egse-client and shall have the same version.

See the section Chapter 5 on how to get the latest version installed in your environment.

9.2. How do I check if all the devices have been connected properly and are active?

TBW

9.3. How do I check if the Storage Manager is running?

You can check this in a terminal with the following command:

```
$ sm_cs status
Storage Manager:
  Status: active
  Hostname: 172.20.10.3
  Monitoring port: 6101
  Commanding port: 6100
  Service port: 6102
  Storage location: /Users/rik/data/CSL/
  Registrations: ['obsid', 'CM', 'PM', 'SYN-HK', 'SYN']
```

If you need more information e.g. to debug which files are being used to save the observation data, add the `--full` option to the above command:

```
$ sm_cs status --full
Storage Manager:
  Status: active
  Hostname: 172.20.10.3
  Monitoring port: 6101
  Commanding port: 6100
  Service port: 6102
  Storage location: /Users/rik/data/CSL/
  Registrations: ['obsid', 'CM', 'PM', 'SYN-HK', 'SYN']
Filenames for all registered items:
  obsid      ->  [PosixPath('/Users/rik/data/CSL/obsid-table.txt')]
  CM         ->
[PosixPath('/Users/rik/data/CSL/daily/20220530/20220530_CSL_CM.csv')]
  PM         ->
[PosixPath('/Users/rik/data/CSL/daily/20220530/20220530_CSL_PM.csv')]
  SYN-HK     ->  [PosixPath('/Users/rik/data/CSL/daily/20220530/20220530_CSL_SYN-
HK.csv')]
  SYN        ->
[PosixPath('/Users/rik/data/CSL/daily/20220530/20220530_CSL_SYN.csv')]
No observation is registered.
Total disk space: 931.547 GiB
Used disk space: 887.819 GiB (95.31%)
Free disk space: 43.728 GiB (4.69%)
```

More information can be found in the

9.4. How do I check if the Configuration Manager is running?

You can check this in a terminal with the following command:

```
$ cm_cs status
Configuration manager:
  Status: active
  Site ID: IAS
  No observation running
  Setup loaded: 00077
  Hostname: 192.168.0.251
  Monitoring port: 6001
  Commanding port: 6000
  Service port: 6002
```

9.5. How do I check if the Process Manager is running?

You can check this in a terminal with the following command:

```
$ pm_cs status
Process Manager:
  Status: active
  Hostname: 192.168.0.251
  Monitoring port: 6201
  Commanding port: 6200
  Service port: 6202
```

9.6. How do I check if the Synoptics Manager is running?

You can check this in a terminal with the following command:

```
$ syn_cs status
Synoptics Manager:
  Status: active
  Hostname: 192.168.0.251
  Monitoring port: 6205
  Commanding port: 6204
  Service port: 6206
```

9.7. How do I check if the Logger is running?

You can check this in a terminal with the following command:

```
$ log_cs status
```

9.8. Where can I find my test data?

TBW