European Organisation for Astronomical Research in the Southern Hemisphere

Programme: ELT

Project/WP: Control System Project Management

ELT HLCC User Manual

Document Number: ESO-515958

Document Version: 6.0

Document Type: User Manual (UMA)

Released On:

Document Classification: ESO Internal [Confidential for Non-ESO Staff]

Owner:

Name

Authors

Name	Affiliation
N.Benes	ESO
G.Chiozzi	ESO
A.Hoffstadt	ESO
E.Pozna	ESO
M.Schilling	ESO
H.Sommer	ESO
P.Szubiakowski	ESO
L.C.Goncalves	Critical SW
R.Leao	Critical SW
M.Przybylski	ITTI

Change Record from previous Version

Affected Section(s)	Changes / Reason / Remarks
1.0	First version
1.5	Updated for HLCC v1.2.0-alpha4
1.5	Updated for HLCC v1.2.0-alpha5
1.5	Updated for HLCC v1.2.0
All	Updated for HLCC v2.0.0, ported to DevEnv 5
	Updated for HLCC v2.1.1
	Updated for HLCC v2.2.0-pre1
	Updated for HLCC v2.2.0-pre2
	Updated for HLCC v2.2.0 release

Contents

1. Introduction

The ELT HLCC (High Level Coordination and Control) is the component of the control software responsible for coordination of all telescope subsystems to properly perform the activities required by scientific and technical operations.

The HLCC offers a single interface of the whole telescope toward operators and the instrument control software, except for deterministic interfaces that are provided by TREx. Its main task is for supervisory applications to coordinate the various telescope subsystems.

1.1 Scope

This document is the user manual for the ELT HLCC .

The intended audience are ELT users, consortia developers or software quality assurance engineers.

At the moment the document covers primarily the Telescope Simulator Software to be used by consortia and in the ECM as well as for development of HLCC itself.

In future issues of this document, alongside with the development of HLCC, it is foreseen to split this document putting the user manual for the actual system separated from the telescope simulator.

Some hyperlinks require access to ESO pages that require authentication (Jira, PDM, Gitlab, OneNotes). If you do not have acces to a page you would like to read, ask the HLCC team or you ESO contact person.

1.2 Related documents

1.2.1 Applicable Documents

The following documents, of the exact issue shown, form part of this specification to the extent specified herein. In the event of conflict between the documents referenced herein and the content of this document specification, the content of this document specification shall be considered as a superseding requirement.

AD1 Not for the time being

1.2.2 Reference documents

The following documents, of the exact version shown herein, are listed as background references only. They are not to be construed as a binding complement to the present document.

RD1 ICD between the Instruments and the Central Control System
ESO-311982 version 3.8

RD2 ELT stellar positions calculations and structure
ESO-394280

RD3 Telescope Wavefront Perturbations Data Package for the E-ELT Instrument Consortia
ESO-271292 Version 2 - https://pdm.eso.org/kronodoc/HQ/ESO-271292/1

1.3 Acronyms

OLDB	CII Online Database
CCS	Central Control System
ECM	ELT Control Model
ELT	Extremely Large Telescope
HLCC	High Level Coordination and control
MS	Main Structure
PFS	Pre-Focal Station
TREx	Telescope Real-Time Executor

1.4 Overview

1.4.1 HLCC Deployment

The HLCC is C++, Python, and Qt software for Linux which are executed on a mix of physical and virtual machines in the Armazones Computer Room. The software has only low requirements on determinism and synchronization, allowing hosting virtual machine instances on ESX servers with NTP time synchronization. Where greater determinism is required (for example for pointing kernels transmitting UDP data at 20Hz), physical machines with network connections to the Deterministic Network, and Time Reference Network (PTP) will be used. UIs will be displayed on terminals in the

Armazones Local Control Room, remotely in Paranal Control Room, or on other terminals in the telescope area if required.

Most of the HLCC interfaces to Subsystem Control Systems are present in the Computer Room over 10GbE. Where data is exchanged with subsystems in the telescope area (e.g. PLCs of an LCS), an infrastructure of single and multimode fibers is available.

1.4.2 HLCC Architecture

The HLCC is architecturally composed of several functional building blocks, as can be seen in the Figure 1.1 below.

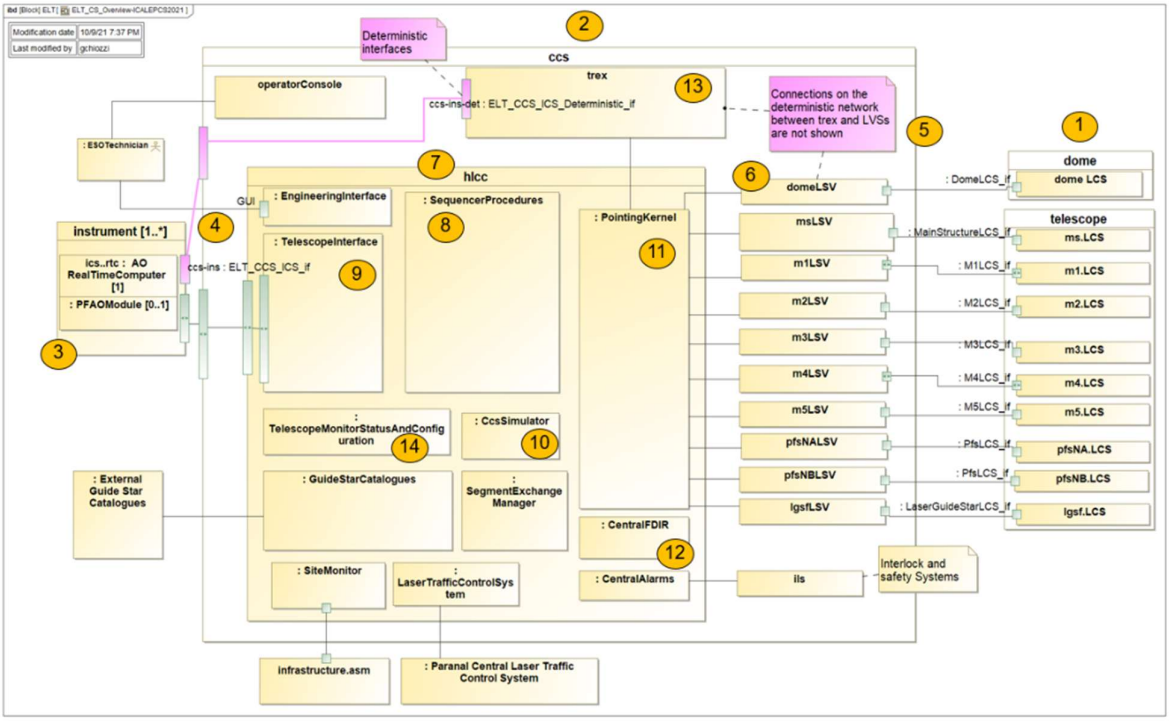


Figure 1.1

Among these:

- The interface toward the instruments, defined in the CCS-INS ICD [RD1], is allocated to the single component in Figure 1.1-(9), whose responsibility is to filter, validate and re-dispatch all commands received. This gives the flexibility to modify the internal structure without impacting the clients.
- In order to provide maximum flexibility during AIV and Commissioning, *features* of the system are implemented by independent supervisory applications designed to perform a complete operational function/use case of value to the users of the system. These procedures are

managed by the SequencerProcedures module (Figure 1.1-(8)) and are written and executed using the Sequencer tool and are decoupled from the rest of the system. In this way also members of the AIV and commissioning teams can directly modify them and add new ones.

- The Telescope Monitor Status and Configuration component (Figure 1.1-(14)) is responsible for monitoring all components of CCS and consolidating this information to build global status indicators. It manages requests for high level changes in the configuration, propagating information to individual subsystems. It makes high level status information available to human users as well to other systems, instruments in particular.
- The PointingKernel (Figure 1.1-(11)) component coordinates and interacts with the pointing kernel components in dome, main structure, PFS and laser systems. It interacts with TREx for most of the pointing logic, but also commands the LSVs directly.
- The CentralFDIR (Figure 1.1-(12)) application monitors those aspects of quality and failures that involve more than a single subsystem.
- Other dedicated applications exist for star catalogues, monitoring and configuration, alarms, or specific tasks such as segment exchanges.

As a general architecture concept at ELT CCS level,

- Instructions are given to components using a command/reply pattern, but the reply is only used to acknowledge proper reception of a command.
- A client shall infer the proper execution and completion of a request from status information available through a publish/subscribe pattern.

For example, an instrument will send a Preset command to the HLCC Telescope Interface to request pointing and tracking to a new target in the sky and receive an OK if the target is valid. It will know that the telescope is on target and ready for the instrument to start exposures when the "*ready for handover*" status information is published.

2. Pointing Kernel functional design

This section is about design aspects related to domain concepts such as astrometric calculations and data products . Software design is described on the separate document Pointing Kernel software design.

Pointing kernel astrometric calculations are implemented in C++ in the ecos/ptk module.

The astrometric calculations are based on the ERFA library (https://github.com/liberfa/erfa), a variant of the Sofa reference implementation, (https://www.iausofa.org/).

The purpose of the ecos/ptk module is to

- Make a selection of ERFA functionality needed for the ELT, so that the same goals are reached in exactly the same way in all ELT CS applications. The SOFA/ERFA library itself is so flexible that different paths can be chosen for a task, yielding slightly different results.

- Enforce coordinate systems, time scales, and angle units by using specific types.
- Connect nomenclature and conventions traditionally used at ESO with the corresponding names and concepts from SOFA/ERFA. Both an expert of VLT instrument or tracking control, as well as a software developer newly joining the ELT project without having a background in astrometry, should find detailed explanations about the concepts used and their names in ESO and SOFA speak, to be able to explore further in those directions.

ERFA supports only transformations from star catalog data given in ICRS coordinates, where also the other intermediate coordinate systems that are part of the IAU 2000 definitions (e.g, CIRS), are different from the transformation chain previously used with Slalib (http://star-www.rl.ac.uk/docs/sun67.htx/sun67.html) on the VLT and most other older telescopes. The pointing kernel uses the modern concepts, with two legacy exceptions:

- Intermediate results are still published in the old way: Local Apparent Sidereal Time instead of Earth Rotation Angle, and Apparent Position instead of CIP-based intermediate position. Changing this will require CCS-INS ICD changes. Likewise, the API still uses terms such as "mean position" that are associated with the old equinox-based coordinate systems.
- As currently also required by the ICD [RD1], the ptk supports not only ICRS/J2000.0 catalog positions, but also equinox-based coordinate systems with a Julian epoch other than J2000.0. For those we use code copied from PAL (http://www.starlink.ac.uk/star/docs/sun267.htx/sun267.html) as a temporary adaptation layer, to make the underlying ERFA look like the old Slalib API.

The calculations largely follow [RD2]. The main transformation chain is:

- From ICRS catalog position to apparent position.
  - Both are given in RA/DEC (or [α, δ]] coordinates, referring to the ICRS/J2000 frame, and the frame of date respectively.
  - Impl in ptk class HorizonEquatorialConverter, method MeanToApparent.
- From apparent position to observed position (HA/DEC).
  - The HA/DEC (or [h, δ]) is given in a frame that is aligned with Earth's axis of date, co-rotates with Earth, and counts HA westward from the site's meridian.
  - Impl in ptk class HorizonEquatorialConverter, method ApparentToObserved.
- From observed position (HA/DEC, local equatorial) to observed position (ALT/AZ, horizon).
  - This is merely a mathematical coordinate transformation, without considering physical effects.
  - Impl in ptk class HorizonEquatorialConverter, method LocalEquatorialToHorizontal.

Below we describe the steps in more detail. Features not yet implemented are mentioned *in italics*.

2.1 Preset/track with target given in catalog (mean) coordinates

- Read the RA/DEC target (incl. cumulative sky offset).
  - *We do not yet use "Xins" / "Yins" PFS straight through focal plane pointing origin coordinates.*
- Get applicable current time: We wait for the next global time event (every 50 ms tick in TAI/UTC) and calculate the time 100 ms in the future, see Jira tickets ETCS-904, ETCS-1105.
- An optional velocity offset (typically used for non-sidereal tracking on mean coordinates) is applied, using the given velocities in RA and DEC, and the time between the given epoch and the applicable current time.
  - *Note that to be actually useful, the epoch will need to be defined more precisely (ongoing ICD discussion).*
  - *We do not publish the resulting corrected mean position, but this requirement could be added.*
  - *We do not protect against unphysical velocities, since the velocity vector does not necessarily describe a physical motion.*
- Radial velocity: If zero (there is no "undefined" option in the interface) then derive it from a given redshift:
  - Uses the formula for Doppler redshift taken from https://en.wikipedia.org/wiki/Redshift#Doppler_effect, solved for v:

```
Int    c =  299792458; // in m/s
double v = c * (2*z + z*z) / (2 + 2*z + z*z);
```

  - *The feature of deriving radial velocity from redshift is ill-defined and will probably be removed, see ongoing ICD discussion*.
- Proper motion correction from target position epoch to coordinate system epoch:
  - This is necessary because the subsequent eraAtci13 (or palMap) call accepts the star position and magnitude of proper motion only for the epoch of the coordinate system (ICRS or equinox).
  - Uses ERFA function eraPmsafe (rather than eraStarpm) to
    - Correct the RA/DEC catalog position.
    - 3D-scale the PM vector (mainly perspective change due to radial velocity).
    - Correct a possible unphysical mismatch between PM velocity and parallax value.
    - (We do not precess the PM vector.)
- Transformation to a geocentric equatorial coordinate system of current epoch:
  - This corrects space motion, annual parallax, light deflection, annual aberration, precession-nutation.

  - We convert applicable current time from TAI to TT (Terrestrial Time).
  - In case of ICRS/J2000.0 target: Call eraAtci13, which includes the transformation from barycentric to geocentric. Then use the returned "equation of the origins" value to convert the returned CIO-based intermediate place to a legacy apparent position.
  - In case of a target given in legacy equinox-based mean coordinates other than J2000.0 (deprecated): Call palMap (drop-in replacement for slaMap) to get the apparent position.
  - We publish the apparent position (for HLCC-internal use only), along with other derived data.
- Transformation from apparent position to observed ALT/AZ coordinates:
  - This handles earth rotation, polar motion, transformation to local horizontal coordinates, diurnal aberration and parallax, atmospheric aberration/refraction.
  - We call eraApio13 and eraAtioq.
  - *We currently use some hardcoded values that should later be provided by the site monitor ubsystem:*
    - temperature = 273.15 K, pressure = 70108 Pa, humidity = 0.1 (should come from ASM).
    - Zero values for DUT1 correction and polar motion (should come from ELT TRS HKS).
    - Observing wavelength = 0.65*1e-6 m (should come from INS).
- Enforce telescope operational limits:
  - If tracking has taken the telescope outside the configured operational range (usually ALT from 20 to 88.5 deg, AZ from -180 to 360 deg), then we mark the preset target as invalid and set the current telescope position as the target position. This stops any presetting/tracking and leaves the telescope ready to receive a new preset command.
  - It is the user's responsibility to detect this condition from the published state and other measurement data.
  - For telescope testing, a "functional range" with wider limits can be enabled, but this is not possible for the instruments.
- Resolve ambiguous AZ target:
  - Note that the ELT AZ range is larger than a full circle: From -180 deg to +360 deg. Many sky AZ coordinates can be realized with 2 telescope AZ coordinates.
  - Normally we choose the telescope AZ target that is closest to the current telescope position, to make presets faster.
  - Circumpolar stars, when the telescope AZ is close to 360 deg, have the risk that later we track into the 360 deg limit. This applies to targets inside the small circle around the south pole, which always stays above the minimum ALT of 20 deg. (Normally "circumpolar" refers to the larger circle of targets that always stay above the horizon.) We play it safe in such cases and always preset the telescope by a full turn to the AZ = 0 deg range, where it can then track continuously for as long as it needs.

We do not optimize this yet by considering the length of the observation or the time until end of night.

- This strategy guarantees that the telescope never hits an AZ limit during tracking. This avoids the problems of either making the telescope "jump" during an observation, or ending the observation prematurely.

- In the telescope simulator, move the simulated telescope
  - This gets done using the RA/DEC target position that applies to current time, not the one computed 100 ms into the future.
  - The telescope's current position gets moved toward the target position. We use the constant configured speeds in ALT and AZ, ignoring acceleration or noise issues.

2.2 Published data

	Det. netw. (MUDPI)	Nondet. netw. (DDS)	OLDB public	OLDB HLCC-private	Det.netw. Llnetio – rtms Pointing kernel positions data
Target RA/DEC	x	x	x		x
Target ALT/AZ	x	x	x		x
Current ALT/AZ	x	x	x		x
RA/DEC apparent back-calculated from current ALT/AZ				x	
Hour angle back-calculated from current ALT/AZ				x	
Local sidereal time	x	x	x		x
Time TAI, UTC	x				x
Parallactic angle, north angle etc.	x	x	x		x

Control state		x	x		
Remaining tracking time					
Platescale					

All of the above steps together typically take less than 1 ms to execute, so that new data could then be published quickly for every iteration. However, in the simulation we introduce an artificial 10 ms delay in the independent software process that receives and publishes current and target positions, to be closer to the future situation with a real telescope, where measurements have to travel from the local control system in various steps through the network.

The above table shows the data that gets published in every 20 Hz loop iteration. Not shown is other data that we publish at lower frequencies. Data in green rows applies to current time, while orange means 100 ms in the future.

The full set of data published by CCS for instrumentation is defined in [RD1].

The data published by the current implementation of the telescope simulator are listed in section 6.4.3.

2.3 Preset/track with target given in ephemeris

TODO: We do not yet support ephemerides.

2.4 Handling of time

The pointing kernel interface receives time values as TAI timepoints. For some parameters, it also accepts time as Julian epoch strings. The DUT1 offset gets passed separately. We never pass UT1 time through the interface.

Internally we convert as needed for the ERFA interface, e.g. to time in UTC in Julian Date representation.

The ptk is designed to work smoothly across leap seconds (positive or negative ones). This is achieved by using the C++ utc_clock::time_point and safe conversion routines. We only use system_clock when needed for calendar conversions, but then handling a possible current leap second as a special case.

With ERFA we consistently use their convention of "quasi-JD" leap second smearing and do not compute time durations as differences of such quasi-(M)JD values.

Another feature related to leap seconds is the adjustment of DUT1 values provided by ptk class Dut1. A DUT1 value that was retrieved before a leap second gets adjusted by Dut1, eventually shifted by 1 s, so that UTC+DUT1 stays continuous and correct. At some convenient time later, a new DUT1 value can be retrieved, which already contains the shift that was introduced by the leap second.

2.5 Backward calculation of mean (ra,dec)

The pointing kernel supports back calculations, using eraAtoi13 (to apparent) and eraAtic13 (to astrometric position in ICRS).

The result will in general be different from the original catalog position, because in the back-calculation we don't consider star-specific data such as proper motion or parallax.

HLCC only publishes back-calculated position in its private OLDB branch. There is no such data in the public interface. The back-calculation only considers the nominal mount position, without the effect of the various mirrors.

In the telescope simulation, when tracking, the published value "radec_at_xy_from_guide_stars" is equal to the commanded RA/DEC target position. In the actual system the value will be actually calculated based on the position of the guide stars in the PFS, when available.

3. Install and Build HLCC

Last Updated: 20250724 – Git tag: v2.2.0-pre5 · Tags · ccs / hlcc · GitLab

3.1 Prerequisite and general notes

These instructions are targeted at building and installing HLCC on a development Virtual Machine

These instructions can be used also to update a machine that is already installed.

These instructions are tested with an up to date Platform-CCS configuration:

```
                CPU : AMD EPYC 7702P 64-Core Processor
          CPU Cores : 4
         Hypervisor : vmware
             Memory : 5.9Gi used, 25Gi avail, 31Gi total
       ELT Platform : 25.15
             Kernel : 6.14.5-100.fc40.x86_64
         Clock Sync : NTP
```

```
CII MAL : 5.0.0
CII SRV : 5.0.0
    RAD : 6.2.0
```

Instructions for special cases, like manual installation of packages for debugging purposes are in light grey, so you know that you can normally skip them.

- This document describes how to install hlcc and the hlcc telescope simulator
  - From RPM
  - From sources extracted from the git repository

  on a machine already prepared with an ELT Development Environment.

  Execute first the common steps here below.

- See the DevEnv release notes here:
  Releases · Wiki · ecos / ELT SW Docs · GitLab (eso.org)

- 16 GB of RAM are required for compilation.
- 32 GB of RAM and at least 16GB of disk space in /opt/nomad/data are required to run HLCC in "ECM on DEV mode", i.e. with the MS, PFS, DOME and TREx full simulation packages on the same VM
- For more information on GIT/gitlab and our repository, see here:
  - GIT (internal OneNote link)
  - Remember to set your `user.name` and `user.email` configuration
  - **Note:**
    There are two options for cloning a repository:
    - `git clone git@gitlab.eso.org …`

    or
    - *`git clone https://gitlab.eso.org/ ….`*

    The first format is convenient on your own development machine, while the second format, asking for uid and password every time you access the remote repository, is better and safer on shared machines.

- Editorconfig - making editing rules consistent (not required, but convenient)
  - EditorConfig helps maintain consistent coding styles for multiple developers working on the same project across various editors and IDEs.

- The project's home page is: https://editorconfig.org/
- Here you find configuration instructions and plugins for many editors/IDEs
- Eclipse configuration was already described above
- For Emacs, install the editorconfig plugin following instructions here: https://github.com/editorconfig/editorconfig-emacs

3.1.1 HLCC installation with ansible playbook

- The ansible playbook takes care of configuring all aspects of a development machine, including login scripts, RPM repositories, CII, Nomad and Consul.
- Login as eltdev
- Check if ansible is already installed on the machine,

```
eltdev:~ 1 > which ansible
```

otherwise install it:

```
eltdev:~ 2 > su
[root@elthlccd175 hlcc]# dnf install ansible
[root@elthlccd175 hlcc]# exit
```

- Get the hlcc git repository with the ansible playbook

  (ToDo: we plan to replace this step with the installation of an RPM containing only the ansible files)

```
login on the machine as eltdev
eltdev:~ 3 > mkdir MODULES
eltdev:~ 4 > cd MODULES/
eltdev:~/MODULES 5 > git clone https://gitlab.eso.org/ccs/hlcc.git
eltdev:~/MODULES 6 > cd hlcc
```

- Run the ansible playbook as eltdev,

  it will show a BECOME prompt where you should provide the root password.

  There are two options:

1) To install a machine outside eso, **without** access to the ESO local DNF repository (this is the default and can be used a instrumentation consortia)

```
eltdev:~/MODULES/hlcc 8 > ansible-playbook ansible/hlcc.yml --ask-become-pass
```

2) To install a machine outside eso, **with** access to the ESO local DNF repository (that includes proprietary software not available outside and in particular the implementation of the LSVs and Trex software)

```
eltdev:~/MODULES/hlcc 8 > ansible-playbook ansible/hlcc.yml --ask-become-pass -e eso_local=true
```

The procedure might require a reboot, for example if it replaces the kernel.

This is the output in case a reboot is required:

```
TASK [Check if restart is needed]
**********************************************************************
**************

fatal: [localhost]: FAILED! => {"changed": true, "cmd": "needs-restarting -r", "delta": "0:00:00.574741", "end": "2025-05-08 09:14:15.665014", "msg": "non-zero return code", "rc": 1, "start": "2025-05-08 09:14:15.090273", "stderr": "", "stderr_lines": [], "stdout": "Core libraries or services have been updated since boot-up:\n  * kernel\n  * kernel-core\n  * linux-firmware\n  * systemd\n\nReboot is required to fully utilize these updates.\nMore information: https://access.redhat.com/solutions/27943", "stdout_lines": ["Core libraries or services have been updated since boot-up:", "  * kernel", "  * kernel-core", "  * linux-firmware", "  * systemd", "", "Reboot is required to fully utilize these updates.", "More information: https://access.redhat.com/solutions/27943"]}

...ignoring


TASK [After upgrade reboot is required]
**********************************************************************
********

Pausing for 5 seconds

(ctrl+C then 'C' = continue early, ctrl+C then 'A' = abort)

[After upgrade reboot is required]

Host will reboot in 5s after upgrade, please re-run this playbook:

ok: [localhost]
```

Wait for the reboot, login again as eltdev and run again the playbook:

```
elthlccd175 eltdev:~ 9 > cd MODULES/hlcc/
```

```
elthlccd175 eltdev:~/MODULES/hlcc 10 > ansible-playbook ansible/hlcc.yml --
ask-become-pass
```

- You can check if CII services are now up and running using

```
eltdev@elthlccd175 ~ $ cii-services info function
# function ..........
Log      |functional:yes
OLDB DP  |functional:yes
OLDB CE  |functional:yes
IntCfg   |functional:yes
```

To see all details (e.g. if something is not functional), use

```
eltdev@elthlccd175 ~ $ cii-services info
```

- You can also install the build dependencies if you need to build hlcc from sources.

```
elthlccd175 eltdev:~/MODULES 13 > su
dnf builddep elt-hlcc-sw-devel
[root@elthlccd175 MODULES]# exit
```

- logout and login again to ensure the environment and lua files are properly loaded
- at this point hlcc is installed from RPMs and can be used in that configuration on DEV and INS deployments

3.1.2 Special dependencies

At the moment of releasing this document the following special dependencies (or newer compatible versions, in most cases the master head should work fine) must be installed if you need to build from sources and run the deployments that include the actual LSVs:

- None now

You might want to install from sources the following dependencies, instead of using the RPMs:

- https://gitlab.eso.org/lsv/pfs.git
- https://gitlab.eso.org/lsv/dome.git
- https://gitlab.eso.org/lsv/ms.git
- https://gitlab.eso.org/trex/trex.git

For this purpose, you have to checkout the correct tag/branch/commit that works with your version of HLCC.

3.2 Build HLCC from sources

These steps describe how to install/upgrade an hlcc installation starting from sources extracted from the git repository, from the latest tag or the master head.

- If you are upgrading a machine to a new DevEnv release, do not forget to:
  - Remove the INTROOT
  - cleanup hlcc and all dependencies running in each waf project with the commands

```
> cd hlcc/interface
> waf distclean
> cd ../software
> waf distclean
> cd ../test
> waf distclean
```

- Get from GIT the HLCC software, in the most convenient place.
  - A typical place used by several developers is ~/MODULES:

```
git clone git@gitlab.eso.org:ccs/hlcc.git
Or
Git clone https://gitlab.eso.org/ccs/hlcc.git
```

- Checkout the branch/tag/commit you want to work with, for example the hlcc release tag mentioned on top of this chapter.
- **As root**,verify if all required develop RPM packages needed to build from sources are installed, in the proper versions, using the commands:

```
> dnf builddep elt-hlcc-sw
```

  If packages are missing or in the wrong version, the commands will interactively ask to install the required ones.

  If the packages are in a specific repository, not configured by default, like the Beta repository, this can be specified on the command line. For example:

```
> dnf builddep --enablerepo=elt-platform-obs elt-hlcc-sw
```

- The `dnf builddep` command will only report and eventually install the packages needed to BUILD hlcc. Depending on the deployment (see 5.1.1) and what the user wants to do, more packages might be needed at run-time as RPM or as sources.
- If you need to build from sources pfs and/or ms LSVs or other packages, as updates in INTROOT for the available RPMs, you can ensure you have all the necessary dependencies by running

  ```
  > dnf builddep <package>sw--devel
  ```
  For example,
  ```
  > dnf builddep elt-lsv-pfs-sw-devel
  ```

- After having installed all dependencies, logout from all shells and login again, to refresh the environment variables and run the new/changed lua files.
- Build the HLCC code:

  ```
  > cd hlcc/interface
  > waf configure build install
  > cd ../software
  > waf configure build install
  > cd ../test
  > waf configure build install
  ```

- The test project contains utilities that are used by integration tests only and that need to be built if one wants to run integration tests (or to check the code of these utilities as examples).
- *it is also very often necessary to cleanup and reload the OLDB as described in section 4.3 (after having installed the latest HLCC)*.

3.3 Hlcc configuration module

ToDo Note: this section is still being developed. You can for the time being ignore it.

To install hlcc configuration module first clone

ccs / hlcc-config · GitLab

into `~eltdev/` (parallel to `INTROOT`). Then `CFGPATH` has to be updated with the location of the hlcc-config path project, for example:

```
> export CFGPATH=/home_local/eltdev/hlcc-config:$CFGPATH
```

3.4 Dependencies

- During development it might be necessary to install manually different versions of an RPM package or to install dependency packages from sources.
- To run hlcc in the ECM deployment on an individual VM, all packages for the non-hlcc code to be executed on the hlcc machine need to be installed:

```
> dnf install elt-lsv-ms-sw-devel
> dnf install elt-lsv-pfs-sw-devel
> dnf install elt-lsv-dome-sw-devel
> dnf install elt-trex-sw-devel
```

- To update to the latest packages (all elt-* packages) in the repository and follow the rolling deployment of packages:

```
> dnf --repo ccs list 'elt-*'
> dnf install elt-*
```

- Here some tips / common procedures taking the rad project as an example:
  - Check first if the correct version and all packages are already installed with the command:

```
eltdev@elthlccd60 ~ $ rpm -qa | grep elt-rad
elt-rad-doc-5.2.0-5.fc34.noarch
elt-rad-5.2.0-5.fc34.x86_64
elt-rad-devel-5.2.0-5.fc34.x86_64
```

  - To install the latest version for all packages:

```
> dnf install elt-rad*
```

  - To upgrade to the current version:

```
> dnf update elt-rad
```

  - You can check out from Git, build and install it into your INTROOT, if you need a version different from the one distributed with the DevEnv.
    For example if you need version 5.2.0:

```
cd somewhere  # e.g. MODULES/
```

```
git clone git@gitlab.eso.org:ifw/rad
cd rad
git checkout v5.2.0
waf configure
waf build install
```

4. Configure the system to run the HLCC code standalone

4.1 Additional system configurations

- Manually update the astropy cache (only needed if running outside the ESO network):
  - HLCC Python scripts use astropy with a configuration that tries to avoid direct downloads from public internet servers for IERS data and leap second data. Instead, we download these cache updates from an ESO-internal server.
  - If you run HLCC outside the ESO network, then this download of astropy data updates will fail.
  - It is in that case recommended to update your local astropy cache.
    - This can be done by running (in a python shell outside of HLCC scripts):
      ```
      from astropy.time import Time
      Time.now().ut1
      ```
    - Or alternatively follow https://docs.astropy.org/en/stable/utils/data.html to more explicitly manipulate the astropy cache, leading to up-to-date tables
      https://datacenter.iers.org/data/9/finals2000A.all
      https://hpiers.obspm.fr/iers/bul/bulc/Leap_Second.dat
  - Note that even having older IERS_A data in the local cache is much better than not having this data cached at all. It avoids that astropy falls back to using the less accurate IERS_B table that was installed with astropy.

4.2 CII services

In order to run the telescope simulator or an application prototype, the CII services for Configuration and OLDB have to be running.

The installation procedure should have already performed all the necessary steps, but here some information in case you want/need to check or to configure manually.

Details are described here:

https://gitlab.eso.org/ecs/eltsw-docs/-/wikis/KnowledgeBase/CII#how-to-startstop-cii-services

You can check the status of the CII services as described in 3.1.1 and you can start the services with the command:

```
$ cii-services start all
```

- To start specific applications and the telescope simulator, see the individual pages in this section.
- The first thing to do is in any case to load the OLDB; see: Loading the OLDB (section (4.3))
- HLCC is available in different deployment configurations.
  See the specific pages below in this document for details on how to configure and start/stop each configuration.

4.3 Loading the OLDB

The OLDB of the HLCC is loaded using the oldbloader tool.

The HLCC startup sequences automatically check the status of the OLDB and eventually load the OLDB corresponding to the selected deployment, but it might be useful to load/check the OLDB manually.

You can skip this section is you are running HLCC only through the standard startup sequences and if your machine does not show problems with the OLDB.

Notes:

If upgrading from a previous DevEnv, it is most probably necessary and in any case advisable to clean-up the old oldb content and reload it, following Note2 here below.

The command

```
hlccOldbloader
```

iterates through all the Oldb description files and check/load them into the Oldb. This is also used in the startup sequence scripts to check the Oldb before start the applications.

The command line options are similar to `oldbloader` with a few additions; check below the most important:

- --check/--cleanup/none
  - --check - will check the *Oldb* for errors or inconsistencies using the *Oldb* description files defined by the --deployment option.

  - --cleanup - will clear all the datapoints described in the *Oldb* description files defined by the --deployment option.
  - If none of the above it will load all the datapoints described in the *Oldb* description files defined by the --deployment option.
- --override(-o) - when this command line option is used in conjunction with --check means that all the datapoints identified with errors/inconsistencies will be reloaded (cleared from the *Oldb* and loaded again) using the specification in the *Oldb* description files.
- --reload(-r) - when this command line option is used, all the datapoints are reloaded with default values when checking Oldb.
- --deployment(-d) - defines which deployment will be executed in the current operation. In this case deployment means a different set of *Oldb* description files that will be used to load/check/clear the *Oldb*.

  There are currently three deployments available:

    - dev - development
    - ecm - ELT control model. Will start all subsystems, assuming that the Nomad configuration provides all the necessary resources, properly configured.
    - ecmondev – ELT control model configuration, but able to run on an individual VM, with all processes for some basic subsystems running in the same machine.
    - ins - instruments
- --abort - Aborts on error (by default would continue with the next item instead)

Therefore…..

To load the Oldb the best is to issue:

```
hlccOldbloader -d dev -o
```

To check the Oldb:

```
hlccOldbloader --check -d dev
```

Look inside the code of the command for more details on the individual steps executed.

Note 1:

If you are updating the HLCC application, you might get an updated OLDB that is not compatible with the one from the previous version that you have already installed and therefore you might get errors when loading the new database.

In this case, delete the current oldb using the oldb-gui application:

- start oldbGui
- enable write mode: Check options -> write enabled
- select the `elt` node

- press the delete button (or delete just the nodes that give problems) and confirm.

Note 2:

If you are updating the DevEnv or in any case if you find errors, you might need to clean-up the oldb content.

It should be sufficient to call the command:

```
> oldbReset
```

The option `-nuke` would execute, if necessary, a harder reset

Note 3:

If you want to load/check individual configuration files, for example for debugging purposes, use the oldbloader command:

- Change the current directory to the place where the oldb files are located.

  source tree: cd <hlcc_root>/<module path>/resource

  INTROOT: cd $INTROOT/resource

  RPM: cd /elt/hlcc/resource

- Run the oldbloader command:

  oldbloader -p <prefix> oldb/<file to load>.yaml

The <prefix> parameter has the following format:

"elt/telif"

where the are no leading or training "/" characters.

The list of files handled by the hlccOldbloader is in

- folder: hlcc/software/common/hlccOldbloader/resource/config/hlccOldbloader
- files: config_xxx.yaml
  one file per each deployment containing names of files and prefix for the OLDB nodes in yaml format

4.4 Startup/shutdown of HLCC services using Nomad jobs

Once Nomad/Consul and the OLDB are ready, it is possible to start HLCC services using Nomad jobs.

This is actually done only for development and debugging, since normally HLCC services are started up and shutdown using sequences/OBs, as described below in section (5) for the telescope simulator.

Therefore, this section can be normally safely skipped.

For example, to start the main HLCC services on the command line:

```
$ nomad job run /elt/hlcc/share/nomad/hlcc.nomad.hcl
```

or

```
$ nomad job run -var=prefix=$INTROOT
                    $INTROOT/resource/nomad/hlcc.nomad.hcl
```

depending on the installation for hlcc (RPM or from sources in $INTROOT).
There are specific startup jobs per each deployment.

Stopping HLCC services on the command line:

```
$ nomad job stop hlcc
```

Commands to start/stop nomad jobs can be issued with the nomad command line tool from any ma chine on any cluster, assuming NOMAD_ADDR is properly set.

Both Nomad and Consul have a web ui:

Nomad web UI: http://elthlccd63.hq.eso.org:4646/ui/clients

Consul web UI: http://elthlccd63.hq.eso.org:8500/ui/dc1/services

Nomad log files are easily accessible from the UI, but they are also available through journalclt and the filesystem:

Logs by the nomad jobs are under `/opt/nomad/data/alloc`

Logs by nomad itself: `journalctl -u nomad | less`
(explicit piping to less prevents line truncation)

5. Telescope System Deployment

The HLCC can be deployed in different ways, depending on the usage foreseen.

The components of the system are as much as possible the same, to allow maximizing reusage and testing, and the deployment is whenever possible managed through configuration options.

5.1.1 Available deployments

Currently 4 different deployments are available:

- INS: Integration with instrumentation software for HLCC simulation

  This deployment is meant to be used by instrumentation consortia to test their instrumentation software against a simulation of the telescope.
  The deployment is on a single VM and consists only of what is necessary to implement the INS/CCS interfaces.

- DEV: Development environment on one single virtual machine

This is the normal deployment used by the members of the HLCC development team.

It consists of all components of HLCC, used in standalone mode, without the need of other subsystems. Interfaces to other subsystems (in particular LSVs) are implemented with simulation processes.

It is normally run on a single development Virtual Machine but it is also possible to run this configuration on the ELT Control Model for testing.

- ECM: ELT Control Model

  This is the configuration deployed on the ELT Control Model and it assumes that all LSVs/subsystems are installed and available there.

  At the moment startup/shutdown sequences and other files from packages HLCC depends from, are available from the elt-xxx-devel RPMs. Therefore to use HLCC in ECM mode it is necessary to install on the same machine where HLCC is running on, the following RPMs (or the corresponding build from source) (ToDo - TO BE CHECKED if devel packages needed as well):

  - elt-lsv-ms-sw
  - elt-lsv-pfs-sw
  - elt-lsv-dome-sw
  - elt-trex-sw

  The lsv software does not create the oldb with an oldbLoader configuration, but at the first startup of the processes. Therefore, a lot of error occur the first time this configuration is started from the hlcctelsimui.

- ECM on Dev: ELT Control Model configuration on an individual development machine, typically for testing.

  This is the subset of the configuration deployed on the ELT Control Model and it assumes that all LSVs/subsystems that are part of this deployment are installed and available there.

  - When running the ECM deployment on an individual VM, the following additional RPMs (or equivalent build from sources) are necessary, in the same version as the interface packages that are anyway installed:
    - elt-lsv-ms-sw
    - elt-lsv-pfs-sw
    - elt-lsv-dome
    - elt-trex-sw

In the future more deployments will be added, with different configuration options.

5.1.2 Deployment files and structure

If you are going to use the HLCC software integrated with the INS software or in the ECM, check the specific procedures for integrated deployment in the instrumentation documentation.

What is provided here are for the time being essentially examples that will be used to create the fully integrated deployment procedures.

For a better understanding, check the software deployment strategy document OneNote, that explains how deployment was planned and implemented.

Deployments are started and shutdown using sequences (actually sequencer Observation Blocks – OBs – that invoke sequences with proper parameters). Every deployment has a startup OB and a shutdown OB.

All OBs are located in the hlcc repository path: `"hlcc/software/seq/resource/config/seq"`

The OBs can be configured using OB parameters to handle specific situations (the default values for the parameters are for the most common situation).

The OBs call/use sequences that are stored in "hlcc/software/seq/src/hlccseq"

Also in the same path we find other helper files like:

- `common/hlccdeployments.py` - This file contains the specification of all deployments. The startup / shutdown sequences will rely on the information in this file to properly execute.
- `common/hlccseq_lib.py` – Contains methods that are common to all deployments and which will be used by sequences.

Deployment sequences also start / stop nomad jobs, whose files are located in hlcc path: `"hlcc/software/nomad/src/"`

We assume here that Nomad and Consul have been properly configured by the ansible playbook as described in section 3.1.1.

OBs and sequences can be launched from the `seqtool gui` or from the `hlcctelsimui` or from the command line `seqtool run`.

In the two sections below we show how to run startup/shutdown OBs from the command line.

5.1.3 Deployment startup

Deployment startup OBs/sequences are responsible for:

- Load/check the Oldb according to the deployment needs
- Start all applications by running the jobs associated with the deployment.
- Init and Enable all applications and leave them in the ready state.

Select the proper OB and launch it manually (with the proper path). For example:

```
$ seqtool run hlccStartupIns.json
```

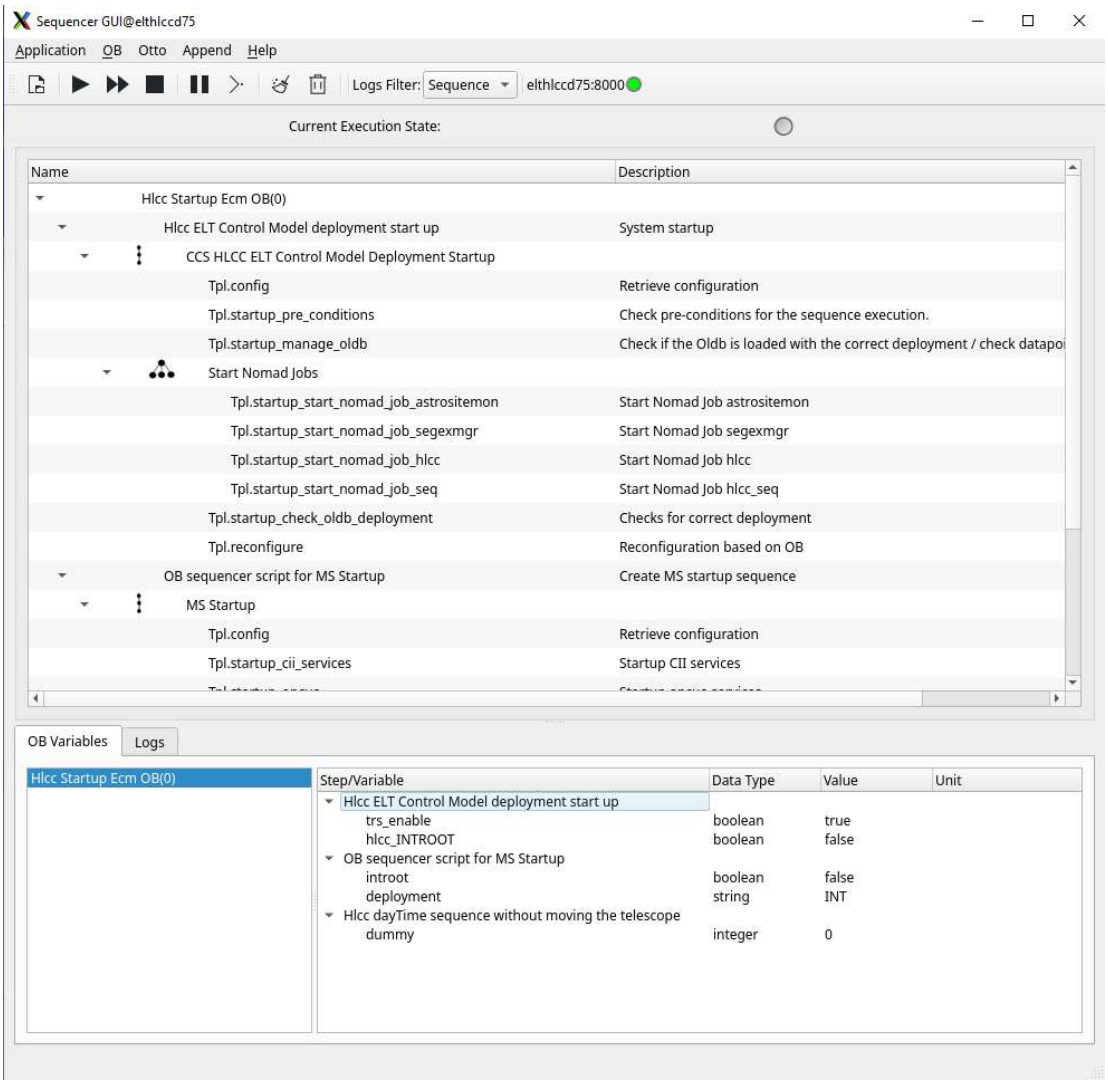
The name of the startup OB depends on the deployment, for example:

- `hlccStartupDev.json` - Development VM deployment
  Startup of the standalone HLCC configuration (no LSVs), assuming installation in INTROOT.
- `hlccStartupDevOnEcm.json` - Development configuration on ECM system
  Startup of the standalone HLCC configuration (no LSVs), assuming installation in INTROOT and assuming it is running on the ECM HW and SW configuration.
- `hlccStartupEcm.json` - ECM deployment
  Assume a deployment on the ECM HW and SW configuration, with the actual LSVs. Assuming installation as RPM for both HLCC and MS.
- `hlccStartupEcmOnDev.json` - ECM configuration on development VM
  Startup of HLCC including a subset of LSVs, but assuming a single VM development HW configuration. Assuming by default installation in INTROOT of HLCC and as RPM for MS.
- `hlccStartupEcmOnDevFull.json` - ECM configuration on development VM
  Startup of the complete HLCC including all LSVs, but assuming a single VM development HW configuration. Assuming installation in INTROOT of HLCC and as RPM for MS. This deployment requires an host with sufficient resources to run all processes. HLCC development team has one such machine available for testing.
- `hlccStartupIns.json` - Instruments deployment

Alternatively, the OB can be launched from the `seqtool gui` or from the `hlcctelsimui`.

The Startup OBs are actually calling the same startup sequences setting different values for the parameters passed to them.

The OBs listed above are setting the parameters for the most common and convenient cases, but it is possible to manage more alternatives (for example ECM on DEV machine with MS installed in INTROOT) by changing the parameters on the seqtool gui:



trs is normally enabled on ECM, but not on development VMs

5.1.4 Deployment shutdown

Deployment Shutdown OBs are responsible for:

- Safely shutdown all deployment applications.
- Stop all the deployment *Nomad* jobs .

Select the proper sequence and launch it manually:

```
$ seqtool run hlccseq.hlccEcmShutdown
```

The name of the Shutdown sequence depends on the deployment:

- `hlccShutdownDev.` json - Development VM deployment
- `hlccShutdownEcm.` json - ECM deployment
- `hlccShutdownEcmOnDevFull.` json - ECM full deployment on Dev machine
- `hlccShutdownEcmOnDev.` json - ECM subset deployment on DevMachine
- `hlccShutdownIns.` json – Instruments deployment

Alternatively the OS can be launched from the `seqtool gui` or from the `hlcctelsimui`.

6. Telescope Simulator

6.1 Overview

The Telescope Simulator consists in a subset of the HLCC tailored at providing a simulation of the telescope behaviour, focusing in particular in providing an implementation of the CCS-INS telescope interface.

The first few sections in this chapter describe the basic functionality, useful when running in simulation mode from the INS perspective.

The following sections go into deeper details and are mostly useful for the users of the DEV and ECM deployments, to test functionality or to configure the system.

6.2 Telescope Simulator deliverables

The ELT Telescope Simulator includes:

- HLCC code (as RPM or source code from Git)
- Documentation

This allows to install it on the corresponding DevEnv release.

The delivered software components are:

- Run-time processes:
  - telif
  - telmon

- eltpk
- trksim (Ms tracking simulator)
- lsvsim (lsv simulator configurable/generic process)

- User interfaces
  - hlcctelsimui
  - hlcctelsimui_mon
- Utilities
  - See: (7.11)

6.3 Telescope Simulator startup

The most natural way to startup the telescope simulator (i.e. the INS deployment on a Virtual Machine), once the system is fully configured, is the following:

- Load the OLDB:

```
> hlccOldbloader -d ins -o
```

- Start the HLCC UI:

```
> hlcctelsimui
```

- The Sequences tab will present a list of the OB/Sequences available for the INS deployment:

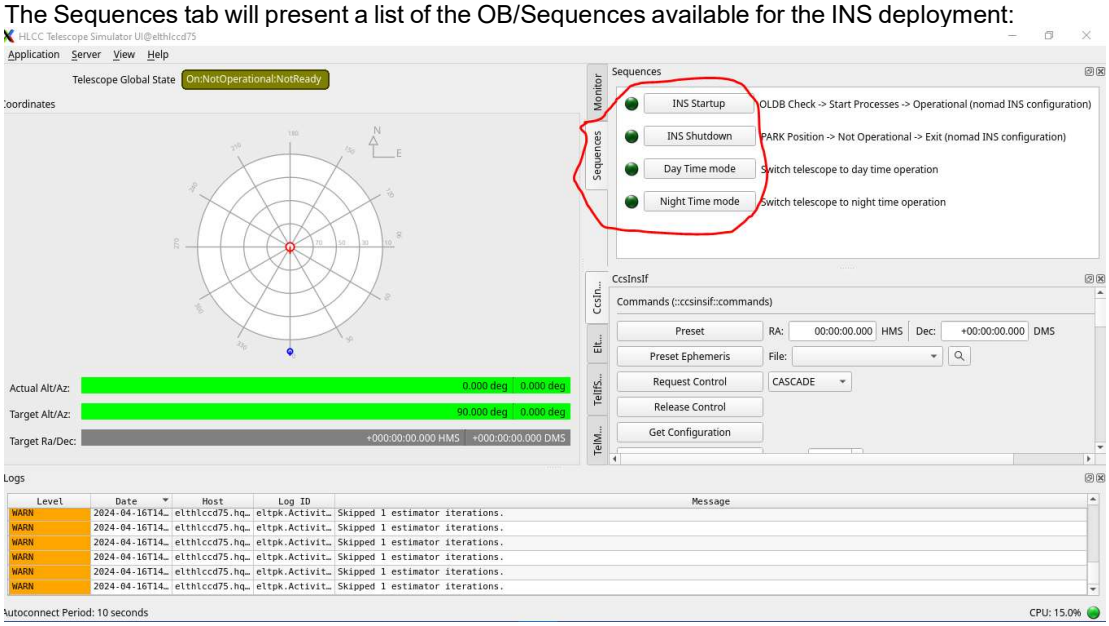


Figure 6.1

- Select the INS Startup button
- A sequencer GUI will open and the OB will be loaded
- Run the OB and the telescope will be started in simulation
- At the end of the startup, the Telescope will be in Day Time mode, On:NotOperational:NotReady
- To bring it in Night Time mode, ready to preset, run in the same way the Night Time mode sequence/OB
- In a similar way you can shutdown the telescope.

More information on the hlcctelsimui is available in section 7.

6.4 Implemented features

The Telescope Simulator provides a (partial) implementation of the following features.

6.4.1 Standard Commands

The standard commands supported are described in this ICD:

- std/if/src/stdif.xml · master · ecs / ecs-interfaces · GitLab (eso.org)

The server URI is:

- `zpb.rr://[ip]:[port]/StdCmds`

Example:

```
$> msgsend --uri zpb.rr://telif/StdCmds ::stdif::StdCmds::GetState
```

Note: The standard commands have only a partial implementation. In particular State and Status are reporting currently only the State and Status of the server process and not of the whole simulator.

Note: As in the example above, msgsend is capable of querying consul and resolve the uri, but tThe `consulGetUri` *utility can also be used to query Consul for the uri dynamically assigned to any of the HLCC processes when they are started up by Nomad:*

```
$> msgsend --uri `consulGetUri -r ifuri -s telif -i StdCmds`
::stdif::StdCmds::GetState
```

6.4.2 CCS-INS ICS Commands on control network

The specific ccs-ins interface commands supported are described in this ICD:

- ccsinsif/icd/src/ccsinsif.xml · master · ccs / hlcc · GitLab (eso.org)

The following commands are currently (partially) implemented:

- Preset
- RequestControl
- ReleaseControl
- SkyOffset
- SetObservingWavelength
- SetVelocityOffset
- GetConfig
- RousConfig, RousExecute and corresponding monitor points

The server URI is:

- `zpb.rr://[ip]:[port]/Commands`

Example:

```
$> msgsend --uri=zpb.rr://telif/Commands ::ccsinsif::Commands::Preset
'{"command": "FAST_PRESET", "preset_data": {"ra":5.5361, "dec":-1.55258,
"system":"J2000.0", "proper_motion_ra":0.0, "proper_motion_dec":0.0,
"epoch":"J2000.0", "parallax":0.0, "radvel":0.0, "rshift":0.0,
"velocity_offset_ra":0.0, "velocity_offset_dec":0.0, "object_name":"-",
"guide_stars": [{"ra":0.1, "dec":0.2, "system":"J2000.0",
"proper_motion_ra":0.3, "proper_motion_dec":0.4, "epoch":"J2000.0",
"parallax":0.5, "radvel":0.6, "rshift":0.7, "magnitude":1.0,
"band":"M"}]}}'

$> msgsend --uri=zpb.rr://telif/Commands
::ccsinsif::Commands::RequestControl '"CASCADE"'

$> msgsend --uri=zpb.rr://telif/Commands
::ccsinsif::Commands::ReleaseControl

$> msgsend --uri=zpb.rr://telif/Commands ::ccsinsif::Commands::SkyOffset
'{"ra": 0.1, "dec": 0.1, "field_stabilization":
"FIELD_STABILIZATION_CLOSED", "guide_star_param": ""}'

$> msgsend --uri=zpb.rr://telif/Commands
::ccsinsif::Commands::SetObservingWavelength 300

$> msgsend --uri=zpb.rr://telif/Commands
::ccsinsif::Commands::SetVelocityOffset '{"ra": 0.1, "dec": 0.1}'

$> msgsend --uri=zpb.rr://telif/Commands ::ccsinsif::Commands::GetConfig

$> msgsend --uri=zpb.rr://telif/Commands ::ccsinsif::Commands::RousConfig
'"ENABLE"'

$> msgsend --uri=zpb.rr://telif/Commands
::ccsinsif::Commands::RousExecute
```

(Notice the single quotes ' and double quotes " usage to delimit the parameter value according to the syntax accepted by msgsend)

6.4.3 Monitor data

The report below lists the data points defined in the CCS-INS ICD that are currently published in the oldb, in the non-deterministic pub/sub (DDS) and in the deterministic pub/sub (MUDPI), with characteristics and limitations.

HLCC publishes data on different DDS Domain IDs.

According to the OneNote page ECM DDS domain IDs hlcc shall publish:

- **Domain 1:** for general info over several subsystems. In particular subsystems shall publish on domain 1 status information for supervisory applications, or data published by HLCC to be used by other subsystems.
- **Domain 2:** CCS/INS interface
- **Domain 3:** HLCC internal/private supervisory communication

The report itself is difficult to read on a printed version of this document but is in the online documentation **[ToDo: Link to be added]** and can be generated dynamically from the source tree.

```
Report: topic_listing_full

 telif:ccs:target_observed_altaz
    Uri  : dds.ps://2/TELIF_CCS_TRG_OBS_ALTAZ
    CPP  : Ccsinsif.hpp - ccsinsif::AltAz
    PY   : ModCcsinsif.Ccsinsif.AltAz
    File : interface/ccsinsif/icd/src/ccsinsif.xml

    Spec/Impl : 1.0 Hz / 20 Hz
    Veri/Ref  : Real / activityDoControl

    OLDB:
       CCS_TARGET_OBSERVED_ALTAZ

 telif:ccs:current_observed_altaz
    Uri  : dds.ps://2/TELIF_CCS_CUR_OBS_ALTAZ
    CPP  : Ccsinsif.hpp - ccsinsif::AltAz
    PY   : ModCcsinsif.Ccsinsif.AltAz
    File : interface/ccsinsif/icd/src/ccsinsif.xml

    Spec/Impl : 1.0 Hz / 20 Hz
    Veri/Ref  : Simulated / activityDoEstimation

    OLDB:
       CCS_CURRENT_OBSERVED_ALTAZ

 telif:ccs:deterministic
    Uri  : mudpi.ps://239.128.7.1:10000/TELIF_CCS_PK_POS
```

```
    CPP  : Ccsinsdetif.hpp - ccsinsif::PointingKernelPositions
    PY   : ModCcsinsdetif.Ccsinsif.PointingKernelPositions
    File : interface/ccsinsif/icd-determ/src/ccsinsdetif.xml

    Spec/Impl : 20.0 Hz / 20 Hz
    Veri/Ref  : Simulated / activityDoEstimation

telif:current_time
    Uri  : dds.ps://2/TELIF_CUR_TIME
    CPP  : Ccsinsif.hpp - ccsinsif::CurrentTime
    PY   : ModCcsinsif.Ccsinsif.CurrentTime
    File : interface/ccsinsif/icd/src/ccsinsif.xml

    Spec/Impl : 1.0 Hz / 20 Hz
    Veri/Ref  : Real / activityDoEstimation

    OLDB:
       TIME_LST

telif:tracking_data
    Uri  : dds.ps://2/TELIF_TRK_DATA
    CPP  : Ccsinsif.hpp - ccsinsif::TrackingData
    PY   : ModCcsinsif.Ccsinsif.TrackingData
    File : interface/ccsinsif/icd/src/ccsinsif.xml

    Spec/Impl : 1.0 Hz / 20 Hz
    Veri/Ref  : Simulated / activityDoEstimation

    OLDB:
       CCS_PARALLACTIC_ANGLE
       CCS_NORTH_ANGLE
       CCS_PUPIL_ANGLE
       CCS_ELEVANTION_DIRECTION_ANGLE
       CCS_RADEC_AT_XY_FROM_GUIDE_STARS
       CCS_OBSERVED_ALTAZ_AT_REQUESTED_XY

telif:tracking_data_extended
    Uri  : dds.ps://2/TELIF_TRK_DATA_EXT
    CPP  : Ccsinsif.hpp - ccsinsif::TrackingDataExtended
    PY   : ModCcsinsif.Ccsinsif.TrackingDataExtended
    File : interface/ccsinsif/icd/src/ccsinsif.xml

    Spec/Impl : 1.0 Hz / <none> Hz
    Veri/Ref  : <none> / <none>

    OLDB:
       CCS_REMAINING_TRACKING_TIME_VALUE
       CCS_PLATESCALE
```

```
telif:application_status
   Uri  : dds.ps://2/TELIF_STATUS
   CPP  : Stdif.hpp - stdif::Status
   PY   : ModStdif.Stdif.Status
   File : /elt/stdif/interface/icd/stdif.xml

   Spec/Impl : 1.0 Hz / on change Hz
   Veri/Ref  : Real / main

   OLDB:
      KEY_MON_STATE

telif:ready_for_handover
   Uri  : dds.ps://2/TELIF_RDY_FOR_HANDOVER
   CPP  : Ccsinsif.hpp - ccsinsif::ReadyForHandover
   PY   : ModCcsinsif.Ccsinsif.ReadyForHandover
   File : interface/ccsinsif/icd/src/ccsinsif.xml

   Spec/Impl : 1.0 Hz / 1 Hz
   Veri/Ref  : Simulated / ReadyForHandoverEstimation

   OLDB:
      CCS_READY_FOR_HANDOVER

telif:command_source
   Uri  : dds.ps://2/TELIF_CMD_SOURCE
   CPP  : Ccsinsif.hpp - ccsinsif::CommandSource
   PY   : ModCcsinsif.Ccsinsif.CommandSource
   File : interface/ccsinsif/icd/src/ccsinsif.xml

   Spec/Impl : 1.0 Hz / 1 Hz
   Veri/Ref  : Simulated / activityDoEstimation

   OLDB:
      CCS_COMMAND_SOURCE

telif:site_data
   Uri  : dds.ps://2/TELIF_SITE_DATA
   CPP  : Ccsinsif.hpp - ccsinsif::SiteData
   PY   : ModCcsinsif.Ccsinsif.SiteData
   File : interface/ccsinsif/icd/src/ccsinsif.xml

   Spec/Impl : 1.0 Hz / <none> Hz
   Veri/Ref  : <none> / <none>

   OLDB:
      SITE_AIR_TEMPERATURE
      SITE_AIR_TEMPERATURE_LAPSE_RATE
      SITE_RELATIVE_HUMIDITY
```

```
      SITE_AMBIENT_PRESSURE
      SITE_WIND_SPEED
      SITE_WIND_DIRECTION
      SITE_SEEING
      SITE_MOON_POS_ALTAZ
      SITE_MOON_PHASE
      SITE_MOON_POS_RADEC
      SITE_MOON_TARGET_DISTANCE

telif:site_info
   Uri  : dds.ps://2/TELIF_SITE_INFO
   CPP  : Ccsinsif.hpp - ccsinsif::SiteInfo
   PY   : ModCcsinsif.Ccsinsif.SiteInfo
   File : interface/ccsinsif/icd/src/ccsinsif.xml

   Spec/Impl : 0.05 Hz / 0.05 Hz
   Veri/Ref  : Real / activityDoEstimation

   OLDB:
      SITE_INFO_ELEVATION
      SITE_INFO_ID
      SITE_INFO_LATITUDE
      SITE_INFO_LONGITUDE

telif:ccs_info
   Uri  : dds.ps://2/TELIF_CCS_INFO
   CPP  : Ccsinsif.hpp - ccsinsif::CcsInfo
   PY   : ModCcsinsif.Ccsinsif.CcsInfo
   File : interface/ccsinsif/icd/src/ccsinsif.xml

   Spec/Impl : 0.05 Hz / on change Hz
   Veri/Ref  : Real / datacontext

   OLDB:
      CCS_INFO_SW_VERSION
      CCS_INFO_OPERATOR

telif:insref:offsets
   Uri  : dds.ps://2/TELIF_INSREF_OFFSETS
   CPP  : Ccsinsifextref.hpp - ccsinsif::insref::Offsets
   PY   : ModCcsinsifextref.Ccsinsif.Insref.Offsets
   File : interface/ccsinsif/icd-33-extref/src/ccsinsifextref.xml

   Spec/Impl : 1.0 Hz / 1 Hz
   Veri/Ref  : Defaults / activityDoEstimation

   OLDB:
      INSREF_ACT_FOCUS
      INSREF_ACT_ABER
```

```
telif:rous:maneuver
   Uri  : dds.ps://2/TELIF_ROUS_MANEUVER
   CPP  : Ccsinsifstroke.hpp - ccsinsif::rous::Maneuver
   PY   : ModCcsinsifstroke.Ccsinsif.Rous.Maneuver
   File : interface/ccsinsif/icd-34-stroke/src/ccsinsifstroke.xml

   Spec/Impl : 5.0 Hz / 1 Hz
   Veri/Ref  : Simulated / activitySimulateRousHandling

   OLDB:
      ROUS_COUNTER_ACT
      ROUS_STATUS
      ROUS_MANEUVER_IN_PROGRESS

telif:ao:health
   Uri  : dds.ps://2/TELIF_AO_HEALTH
   CPP  : Ccsinsifao.hpp - ccsinsif::ao::Health
   PY   : ModCcsinsifao.Ccsinsif.Ao.Health
   File : interface/ccsinsif/icd-35-ao/src/ccsinsifao.xml

   Spec/Impl : 1.0 Hz / 1 Hz
   Veri/Ref  : Defaults / activityDoEstimation

   OLDB:
      M4_HEALTH_STATUS
      M5_HEALTH_STATUS
```

The command to create the above report from the code is:

```
software/tools/icd-info.py  interface/ccsinsif/topics/src/CcsInsTopics.xml
interface/ccsinsif/icd/src/ccsinsif.xml  interface/ccsinsif/icd-
determ/src/ccsinsdetif.xml     interface/ccsinsif/icd-33-
extref/src/ccsinsifextref.xml  interface/ccsinsif/icd-34-
stroke/src/ccsinsifstroke.xml  interface/ccsinsif/icd-35-ao/src/ccsinsifao.xml
software/telif/eltpk/eltpkif/icd/src/eltpkif.xml  interface/ccsinsif/icd-info-
topic-impl.yaml /elt/stdif/interface/icd/stdif.xml -R topic_listing_full
```

Note: The report is printed on `stdout` and the format is chosen with the `-R` option, for the available report options, see `software/tools/icd-info.py -h`

To subscribe to monitor data, see for example, the utility (7.11.2):

```
$ msgsendSub –help
```

and the example Jupyter notebooks.

6.4.4 METADAQ Data Acquisition commands on control network

The application supports the metadaq commands described in this ICD:

- metadaq/if/src/metadaqif.xml · master · ecs / ecs-interfaces · GitLab (eso.org)

The following commands are currently (partially) implemented:

- StartDaq
- StopDaq
- AbortDaq
- GetDaqStatus

The server URI is:

zpb.rr://[ip]:[port]/MetaDaq

Example:

```
$> msgsend --uri=zpb.rr://telif/MetaDaq ::metadaqif::MetaDaq::GetDaqStatus
'"1"'
```

6.4.5 RAD Application commands

The application supports the standard RAD Application commands described in this ICD:

rad/cpp/appif/src/appif.xml · master · ifw / rad · GitLab (eso.org)

The server URI is:

zpb.rr://[ip]:[port]/AppCmds

Example:

$> msgsend --uri=zpb.rr://telif/AppCmds ::appif::AppCmds::GetConfig """"

6.4.6 Simulation specific commands

Additional commands to provide control of the simulation are described in this ICD:

- telif/telifsimif/icd/src/telifsimif.xml · master · ccs / hlcc · GitLab (eso.org)

The following commands are currently (partially) implemented:

- UpdateRousTimer

The server URI is:

- `zpb.rr://[ip]:[port]/SimCmds`

Example:

```
$> msgsend --uri=zpb.rr://telif/SimCmds
::telifif::SimCmds::UpdateRousTimer "1"
```

6.4.7 TelifCommands specific commands

Additional commands to provide control of the simulation are described in this ICD:

- telif/telifsimif/icd/src/telifsimif.xml · master · ccs / hlcc · GitLab (eso.org)

The following commands are currently (partially) implemented:

- MoveToNamedPos
- MoveToAltAzPos
- SetPresetSequenceMode
  Set the mode to be used when handling a preset command:
  - HEADLESS – executes the preset sequence without user interaction, started using the `seqtool run` command
  - GUI – executes the preset sequence expecting a sequencer GUI for the interaction with the operator.
    The sequence runs in a sequencer server instantiated at startup with Nomad.
    As such, the sequencer GUI can be therefore started with the following command:

    ```
    seqtool gui --address `consulGetUri -s seqserver -r
    ipport`
    ```
- SetPresetSequence
  Allows to define the specific preset sequence to be used when a preset command is received and mode is GUI or HEADLESS.

The server URI is:

- `zpb.rr://[ip]:[port]/TelifCmds`

Example:

```
$> msgsend --uri =zpb.rr://telif/TelifCmds
::telifif::TelifCmds::MoveToNamedPos '"PARK"'
```

6.4.8 Supported commands and replies/error replies

Command		Condition	Message
TelifCommands	Preset	Command accepted	"OK"
		Failed to convert RA/DEC coordinates from mean to apparent position. hlcc::ptk::MeanApparentConverter throws exception during conversion.	"PRESET DATA ERROR!" (ccsinsif::GeneralException)
		Failed to get site position from Oldb due to error reading Longitude	"FAILED TO GET SITE POSITION: Error reading

			Longitude from Oldb" (ccsinsif::GeneralException)
		Failed to get site position from Oldb due to error reading Elevation	"FAILED TO GET SITE POSITION: Error reading Elevation from Oldb" (ccsinsif::GeneralException)
		Failed to get site position from Oldb due to error reading Latitude	"FAILED TO GET SITE POSITION: Error reading Latitude from Oldb" (ccsinsif::GeneralException)
		Failed due to position out of operational range	"POSITION OUT OF TELESCOPE OPERATIONAL RANGE!" (ccsinsif::GeneralException)
		Failed to publish preset args to Oldb	"ERROR PUBLISHING PRESET ARGUMENTS IN OLDB: " + reason (ccsinsif::GeneralException)
		Failed to connect to eltpk	"CONNECTION TO ELTPK NOT STABLISHED!" (ccsinsif::GeneralException)
		Command rejected.	"COMMAND REJECTED FROM ELTPK: " + reason (ccsinsif::GeneralException)
	SetObservingWavelength	Command accepted	"OK"
		Command failed	"FAILED TO SET OBSERVING WAVELENGTH: " + reason (ccsinsif::GeneralException)
	SetVelocityOffset	Command accepted	"OK"
		Ready for handover = false	"FAILED TO SET VELOCITY OFFSET: NOT READY!" (ccsinsif::GeneralException)
		Command failed	"FAILED TO SET VELOCITY OFFSET: " reason (ccsinsif::GeneralException)
	SkyOffset	Command accepted	"OK"
		Ready for handover = false or not successfully accepted be eltpk	"FAILED TO SET OFFSET SKY: NOT READY" (ccsinsif::GeneralException)

		Command failed	"FAILED TO SET OFFSET SKY: " + reason (ccsinsif::GeneralException)
		Command accepted	"OK"
	SetReferenceFocus		
		Command accepted	"OK"
	SetReferenceAberration		
		Command accepted	"OK"
	RequestControl	Command accepted but the requested control mode is already applied	"OK, NO CHANGE"
		If parameter not CASCADE nor SEQUENTIAL.	"FAILED TO REQUEST CONTROL: PARAMETER NOT VALID!" (ccsinsif::GeneralException)
		Ready for handover = false	"FAILED TO REQUEST CONTROL: NOT READY!" (ccsinsif::GeneralException)
		Failed to publish in oldb	"FAILED TO REQUEST CONTROL: " + reason (ccsinsif::GeneralException)
	ReleaseControl	Command accepted	"OK"
		Ready for handover = false	"FAILED TO RELEASE CONTROL: NOT READY!" (ccsinsif::GeneralException)
		Command source not CASCADE nor SEQUENTIAL	"FAILED TO RELEASE CONTROL: NOT IN CONTROL!" (ccsinsif::GeneralException)
		Failed to publish in oldb	"FAILED TO RELEASE CONTROL: " + reason (ccsinsif::GeneralException)
	GetConfig	Command accepted	String containing configuration as a json map. Any datapoint not successfully retrieved from Oldb will not be included in the map.
	RousConfig	Command accepted	"OK"
	RousExecute	Command accepted	"OK"

MetadaqCommands	StartDaq	Command accepted	The ID of created DAQ
		Ready for handover = false	"FAILED TO START DAQ: NOT READY" (ccsinsif::GeneralException)
		Daq id just created state is different from 'NotStarted'	"NOT POSSIBLE TO START DAQ ID" (ccsinsif::GeneralException)
		In the case the command was issued with empty ID and some exception happened when trying to determine the ID of next DAQ.	"FAILED TO DETERMINE ID FOR NEW ACQUISITION" (ccsinsif::GeneralException)
		If the provided DAQ ID already exists.	"DAQ ID ALREADY EXISTS" (ccsinsif::GeneralException)
		If for some unknow reason the DAQ just created return nullptr.	"FAILED TO CREATE NEW DAQ: UKNOWN REASON" (ccsinsif::GeneralException)
	StopDaq	Command accepted	The 'metadaqif::DaqStopReply' data
		Daq Id provided does not exist	"FAILED TO STOP DAQ: ID DOESN'T EXIST" (ccsinsif::GeneralException)
		Daq state is different from 'Acquiring'	"FAILED TO STOP DAQ ID. CURRENT DAQ STATE: " + daq current_state (ccsinsif::GeneralException)
	AbortDaq	Command accepted	The ID of aborted DAQ
		Daq Id provided does not exist	"FAILED To ABORT DAQ: ID DOES'T EXIST" (ccsinsif::GeneralException)
		Daq state is different from 'Acquiring' and 'NotStarted'	"FAILED To ABORT DAQ ID. CURRENT DAQ STATE: " + daq current_state (ccsinsif::GeneralException)
	GetDaqStatus	Command accepted	The 'metadaqif::DaqStatus' data
		Daq Id provided does not exist	"FAILED TO GET DAQ STATUS: ID DOES'T EXIST" (ccsinsif::GeneralException)
SimCommands	MoveToNamedPos	Command accepted	"OK"

		If given position is not in configuration	"POSITION NOT FOUND!" (stdif::ExceptionErr)
		If given position is out of telescope functional range	"POSITION COORDINATES OUT OF TELESCOPE FUNCTIONAL RANGE!" (stdif::ExceptionErr)
		Failed to connect to eltpk	"FAILED SET NEW COORDINATES:" + reason (stdif::ExceptionErr)
	MoveToAltAzPos	Command accepted	"OK"
		If given position is out of telescope functional range	"POSITION COORDINATES OUT OF TELESCOPE FUNCTIONAL RANGE!" (stdif::ExceptionErr)
		Failed to connect to eltpk	"FAILED SET NEW COORDINATES:" + reason (stdif::ExceptionErr)
	UpdateRousTimer	Command accepted	"OK"
		Rous state is different from ROUS_WAITING	"ROUS NOT IN CORRECT STATE!" (stdif::ExceptionErr)
AppCmds			
	GetConfig	Command accepted	String containing the configuration contents requested
	LoadConfig	Command accepted	"OK"
	SaveConfig	Command accepted	"OK"
	GetTrsHealth	Command accepted	"OK", "BAD"
	… some more commands…		

7. HLCC Applications

HLCC consists of several interconnected applications.

Depending on the deployment, these applications are started with different configurations and talk with simulations of subsystems or with the real counterpart.

This is done with the purpose of minimizing the differences between the telescope simulation and the real telescope coordination software in the different deployments.

From the perspective of the users of the Telescope Simulation, the most relevant sections in this chapter are hlcctelsimui and hlcctelsimui_mon, describing the user interface, and telif, describing the front-ent process for the instruments.

All sections describe how the applications can be tuned/configure/extended and provide some insight on the architecture and on the design of the system.

7.1 hlcctelsimui documentation

This is the HLCC Simulator operator UI.

For the time being it provides only limited functionality:

- Connect/disconnect telif, telmon and eltpk processes
- Monitor logs in the system
- Send standard commands to telif, telmon and eltpk as well as some of the specific commands implemented.
- Monitor a few telemetry values
- Display the position of the telescope in the "dartboard"
- Execute sequences/OBs

To start the application:

```
$ hlcctelsimui
```

Note:

When the application is started up, it reads the current deployment configuration from the OLDB data point:

```
cii.oldb:///elt/cs/telif/ccs/info/deployment
```

and it adapts some menus and features based on this information (see the OneNote page: Implemented strategy for multiple deployment configurations).

When the system is started for the first time after a cleanup of the OLDB, there is no current deployment.

The startup sequences set the current deployment by writing the proper value in the OLDB point mentioned above.

Since the deployment is not supposed to change frequently, we have not considered necessary to implement dynamic reconfiguration and in case of change of deployment it is necessary to restart the UI.

Note:

Some of the commands presented in the UI are not yet implemented in the underlying server applications. In this case an error is returned. See the list of implemented features in section 6.3.

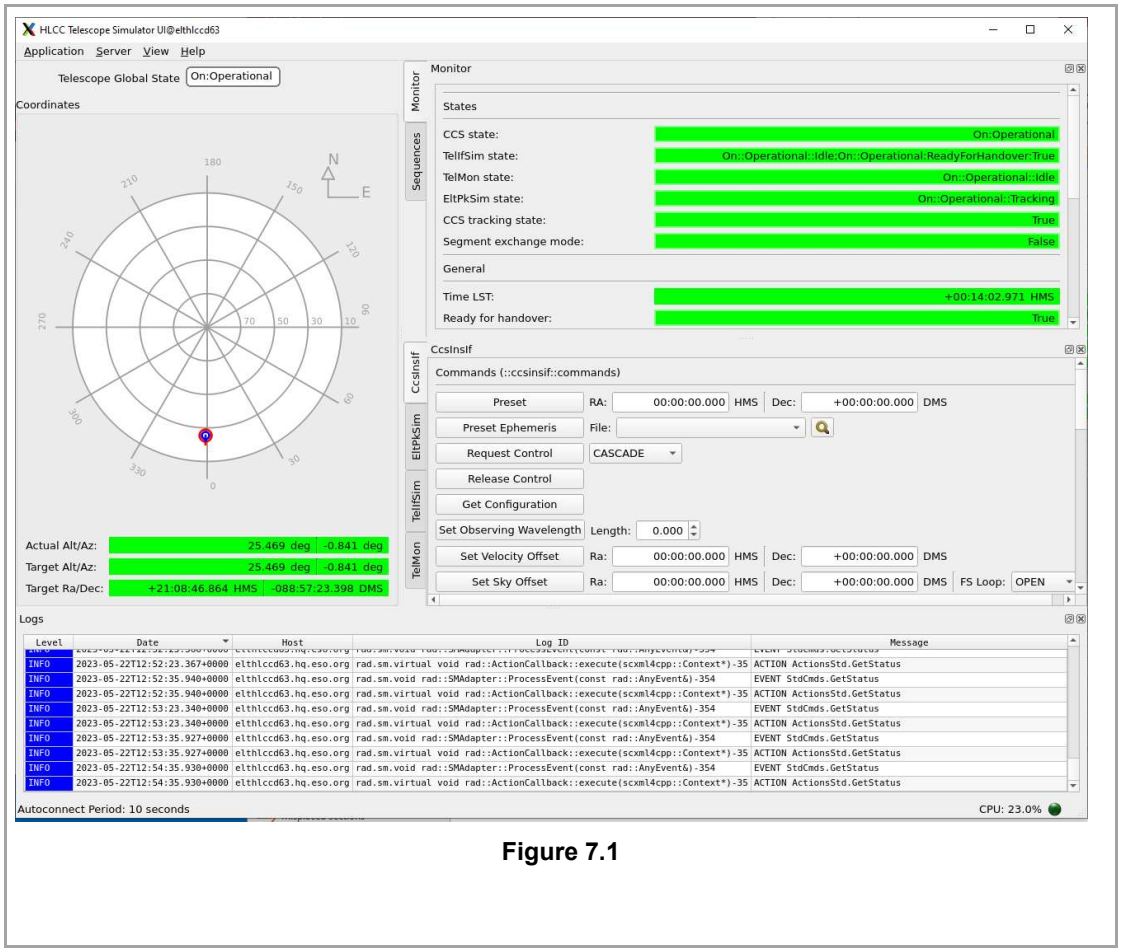


Figure 7.1

Here below the available command line options as from the output of the command:

```
$ hlcctelsimui -help Usage: hlcctelsimui [options]
```

HLCC Telescope Simulator UI is an application used to control the ELT

Telescope Simulator.

Options:

-h, --help show this help message and exit

-s, --subscription Uses subscription instead of polling for OLDB

datapoints updates

--polling-period=POLLINGPERIOD

Polling period in ms. Default: 100[ms]

-v, --verbose Sets LogLevel to Debug. Output more information on application procedures.

-V, --extra-verbose Changes the LogLevel of the application to Trace. Outputs everything.

Taurus Options:

Basic options present in any taurus application

--taurus-log-level=LEVEL

taurus log level. Allowed values (case insensitive): critical, error, warning, info, debug, trace

--taurus-polling-period=MILLISEC

taurus global polling period in milliseconds

--taurus-serialization-mode=SERIAL

taurus serialization mode. Allowed values (case insensitive): serial, concurrent (default)

--tango-host=TANGO_HOST

Tango host name (either HOST:PORT or a Taurus URI, e.g. tango://foo:1234)

--remote-console-port=PORT

enables remote debugging using the given port

--default-formatter=FORMATTER

Override the default formatter

7.1.1 Connection

From the Server menu (Figure 1.1) it is possible to connect/disconnect from the Server, if for example they are executed from another application or the command line.

If the Server Autoconnect checkbox is selected (default) the ui will periodically try to automatically connect.

If HLCC is not running for a longer time, the time between two autoconnect attempts increases, to be reset as soon as a new autoconnect attempt succeed or a manual connection is requested.

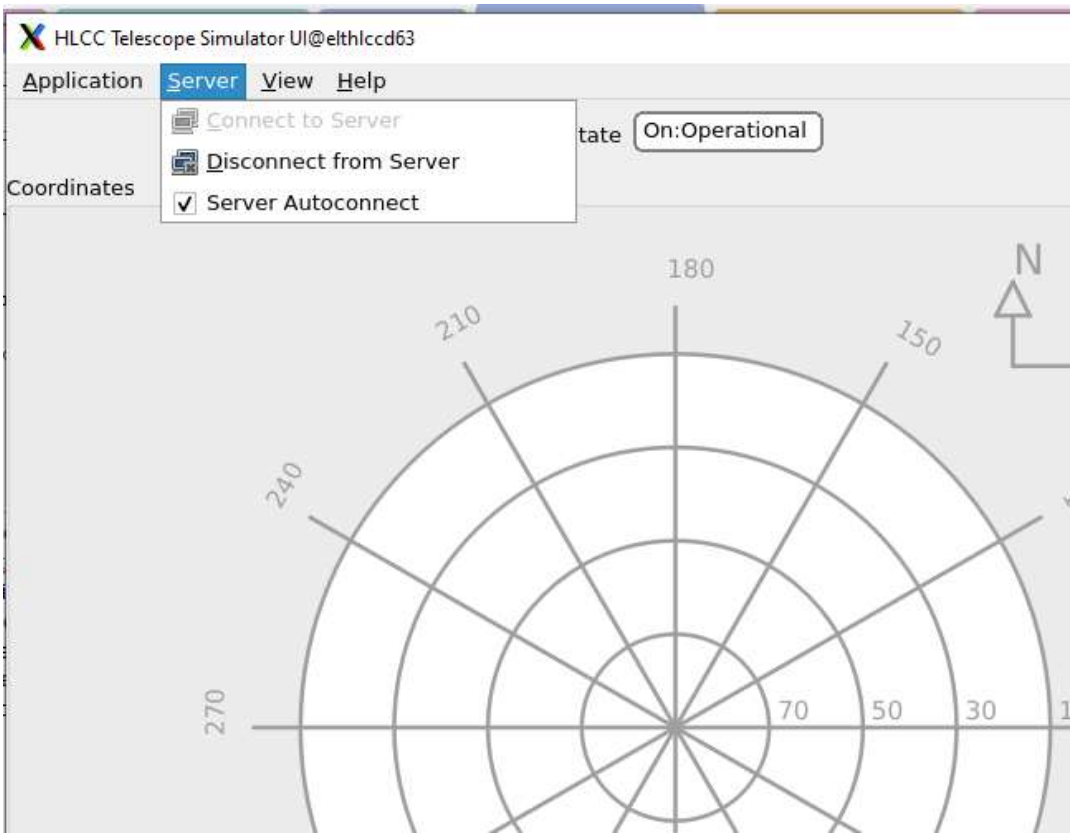


Figure 7.2

7.1.2 Command tabs

Clicking on a process tab on the left side of the "Commands" Panel, presents the commands for that particular application.

The CssInsIf tab provides all commands defined in the CCS-INS ICD (these commands are sent to the telif application, that is the frontend for the CCS-INS interface of the telescope).

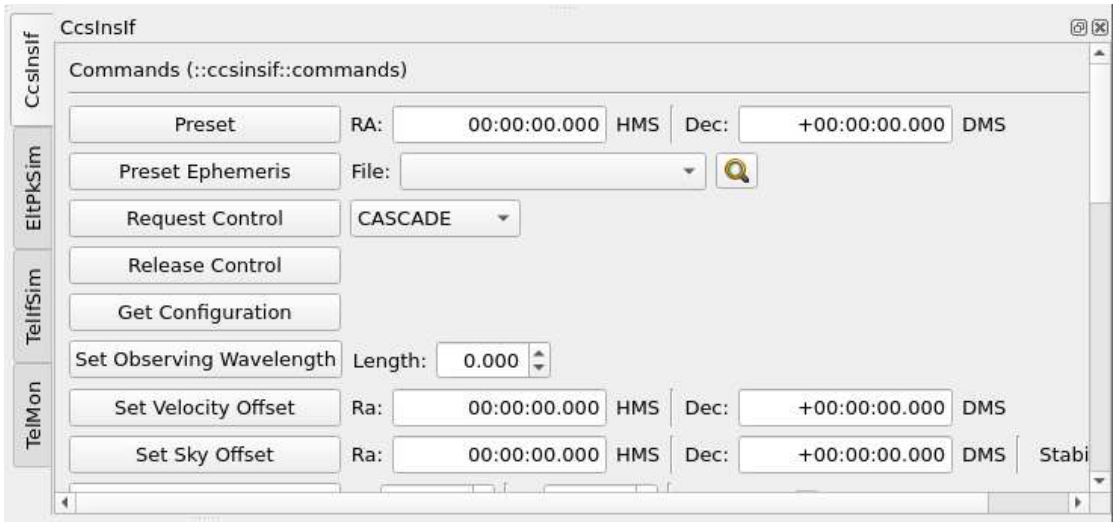


Figure 7.3

7.1.3 Monitor panel configuration

The content of the monitor panel can be tuned by changing the configuration file:

```
hlcc/software/telsimui/resource/config/hlcctelsimui/monitors.json
```

or its copy installed in INTROOT or in the RPM location.

For more details see section 7.1.9.

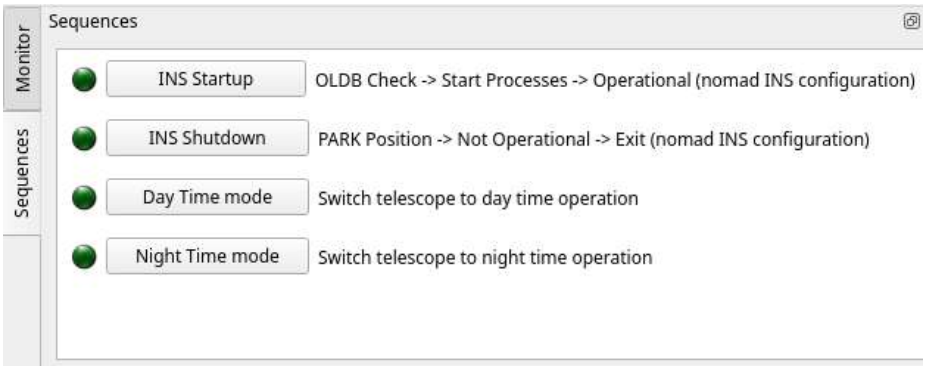
7.1.4 Sequences panel

The Sequences panel from the UI dock widget helps interact with pre-implemented scripts (Sequencer scripts or OBs in particular) and to execute them **detached** from the main process.

The contents of this panel depends on the deployment.

As can be seen in Figure 7.4, a Led indicates the running state of the process running the script and a simple description is also attached to the row entry configured for documentation. If anything fails, it is possible to see the failure return code in the logging widget. The success log entry is also added to the widget.

When a Sequence/OB is selected, a Sequencer GUI is started with an own Sequencer server and the Sequence/OB is loaded in the server. The user/operator can then start the sequence and interact with it using the Sequencer GUI.



loo

Figure 7.4

It's important to notice that only one script sequence of the same type can be executed at a given time, therefore if one is already running an information log will notify the user. There are instead no limitations in running multiple different sequences at the same time. It is responsibility of the user to avoid running in parallel conflicting sequences.

For details on the configuration of this panel see section 7.1.9.3.

7.1.5 Telescope State panel

The Telescope State panel is accessible form the `Application -> Telescope State menu`:

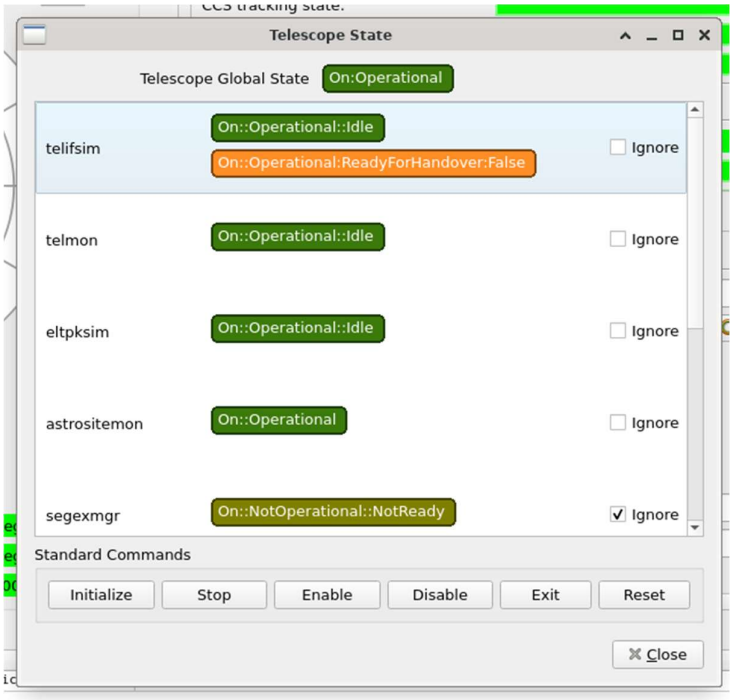


Figure 7.5

This panel allows to monitor the state of the processes that are part of the current deployment of HLCC (for example the INS telescope simulator deployment).

- It is possible to select each entry and send the standard lifecycle commands using the buttons in the bottom bar.
- The ignore toggle allow to ignore an individual process in the estimation of the global telescope state, that is shown on top of the panel.

Note: The "Exit" command will request the process to exit, but if the process is managed by Nomad, the process will be typically restarted automatically

7.1.6 List of HLCC Servers configuration

The list of HLCC servers, as they can be controlled with the application Server menu, can be tuned by changing the configuration file:

`hlcc/software/telsimui/resource/config/hlcctelsimui/processes.json`

or its copy installed in INTROOT or in the RPM location

For more details see section 7.1.9

7.1.7 Known Issues

This section lists some issues you might encounter, and we know of.

- None now

7.1.8 ToDo

Missing/incomplete important functionality:

- None now

7.1.9 UI Configuration (Telescope Monitor, Status, Scripts….)

In this section you can find a summarized explanation of how to configure some parameters of hlcctelsimui such as Logging, Monitor panel entries, Sequencer Script UI list, Global Telescope State window, etc...

At the moment it is possible to configure the application using the .json files:

- logging.json
- monitors.json
- sequences.json
- processes.json
- tools.json

These files are located under "`telsimui/resource/config/hlcctelsimui`" and are installed following the standard CCS conventions.

It is possible to use your own files, for example for a usage specific configuration, by:

- using your own resources folder *`<resource_folder>`* (for example `/home_local/eltdev/resource`)
- add it on top of *`$CFGPATH`*:

  export CFGPATH=<resource_folder>:$CFGPATH
- copying the resource file(s) you need to modify in
  - `<resource_folder>/config/hlcctelsimui/`
    for example:

    `/home_local/eltdev/resource/config/hlcctelsimui/monitors.json`
- edit the files there according to your needs

At this point when you restart `hlcctelsimui`, the application will find first your modified files and use them.

7.1.9.1 Monitor panel configuration:

The image below shows the monitor dock widget that displays a set of configured entries to be viewed by the user.

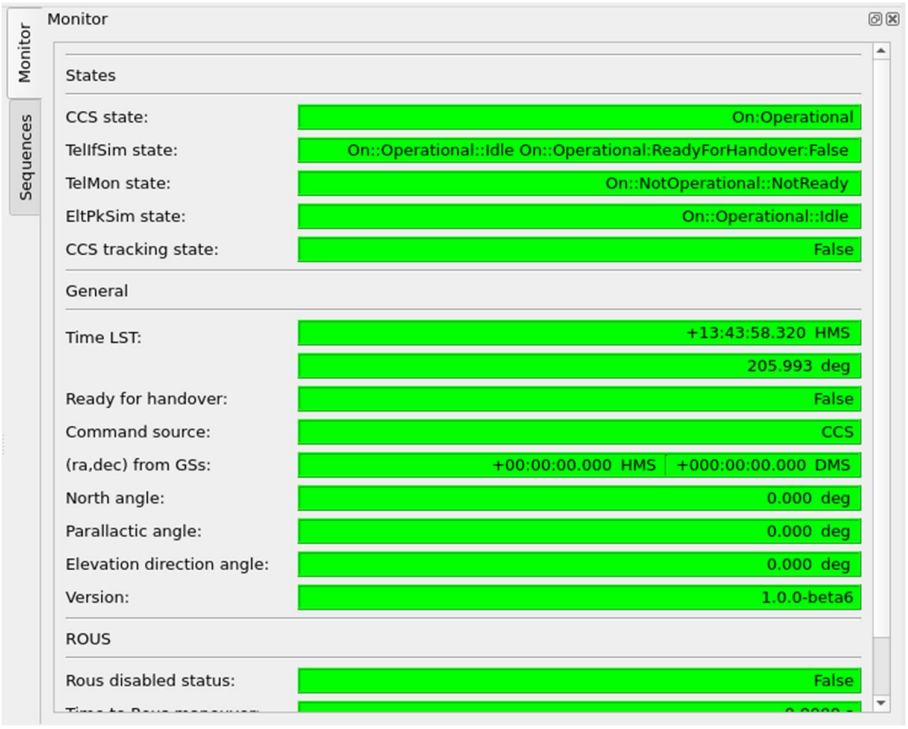


Figure 7.6

The "`monitor.json`" file responsible for this configuration will dynamically affect the panel view when instantiated.

The configuration can be divided into different '**groups**' and that will be reflected in the application window. For each group, it is possible to specify the label text to be displayed.

Inside the groups, monitor entries can now be added with the following data:

- **'label'**: Name of the entry to be displayed
- **'uri'**: OLDB datapoint identifier from where the value to be monitored shall be retrieved
- **'indexes'**: In case the datapoint value is a container it is possible to select one or more elements from it. Just add a list of indexes (ex. [0, 2, 4]).
- **'widgets'**: List of Widgets responsible to represent the value itself.

This **'widgets'** parameter can have different types depending on the python implementation.

For example if you have `["TaurusRadiansLabel", "TaurusHmsLabel", "TaurusSexagecimalLabel"]`, the 3 labels will be instantiated but only one will be showed at a

time. This logic is handled by the Qt Dynamic Property regarding the **Unit Type** choosen to be displayed.

The set of displayed units can be selected for the whole application using the Numerical Units menu:

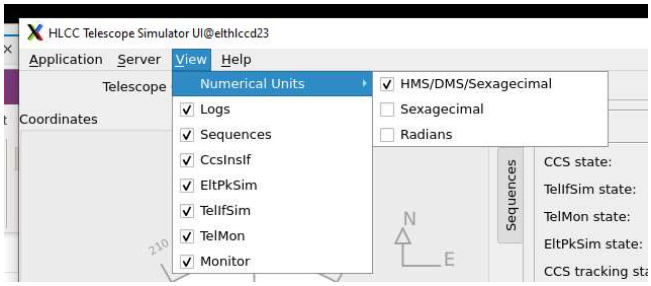


Figure 7.7

Since the types of widgets supported can change in the future it is advised to check **`'monitorpanel_item.py'`** for valid types.

Figure 7.8

7.1.9.2 Processes configuration:

The processes required for the application to work on (TelIf, Eltpk, etc, …) are dynamically retrieved from the telmon OLDB datapoint:

cii.oldb:///elt/cs/hlcc/telmon/cfg/monitored_apps

The `telmon` process is responsible for populating this OLDB datapoint based on the current deployment, using Consul to retrieve the corresponding URIs.

7.1.9.3 Sequences Script UI List configuration:

In "`sequences.json`" you can configure the sequences/OBs or other executables that can be launched from the Sequences panel.

The list of sequences from the UI dock widget helps interact with the **scripts** pre-implemented and to execute them **detached** from the main process.

From the image below, a Led indicates the running state of the process running the script and a simple step description is also attached to the row entry configured for documentation. If anything fails, it is possible to see the failure return code in the logging widget. The success log entry is also added to the widget.

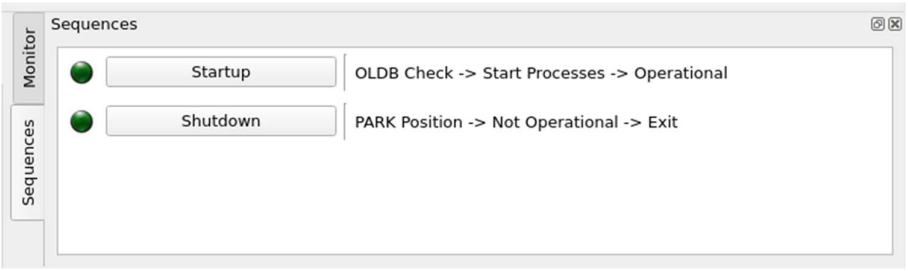


Figure 7.9

It's important to notice that only one script sequence of the same type can be executed at each time, therefore if one is already running an information log will notify the user. There are instead no limitations in running multiple different sequences at the same time. It is responsibility of the user to avoid running in parallel conflicting sequences.

Beyond the name and step description configuration, the way to indicate which sequence to associate to the Qt button is via a **command** call (normally is the **"seqtool gui",** that will start a sequencer GUI and server, or **"seqtool run"**, that will run the sequence unattended)

```
"sequences": [
  {
    "name": "DEV Startup",
    "command": "seqtool gui",
    "sequence": "config/seq/hlccStartupDev.json",
    "description": "OLDB Check -> Start Processes -> Operational (nomad DEV configuration)"
  },
  {
    "name": "DEV Startup on ECM System",
    "command": "seqtool gui",
    "sequence": "config/seq/hlccStartupDevOnEcm.json",
    "description": "OLDB Check -> Start Processes -> Operational (nomad DEV configuration on ECM)"
  },
```

Figure 7.10

The *sequence* field can contain alternatively:

- A path (relative to search in CFGPATH) to an OB .json file, like:

```
"sequence": "config/seq/hlccStartupEcmOnDev.json"
```

- A path (relative to search in CFGPATH) to a source python sequence in a .py file, like:

```
"sequence": "hlccseq/simSegmentExchangeMode.py"
```

- A python module in the PYTHONPATH of a sequence, like:

```
"sequence": "hlccseq.simSegmentExchangeMode"
```

7.1.9.4 Telescope Global State configuration:

Figure 7.11

This panel is automatically instantiated based on the information retrieved from the telmon OLDB data point

```
cii.oldb:///elt/cs/hlcc/telmon/cfg/monitored_apps
```

Each deployment provides a specific configuration for this data point.

When the Ignore flag is modified in the UI panel, the information is stored in the OLDB point.

7.1.9.5 External tools configuration:

"`tools.json`" contains configuraton of external applications which can be launched from Main Window tool menu under "**Launch**" option.

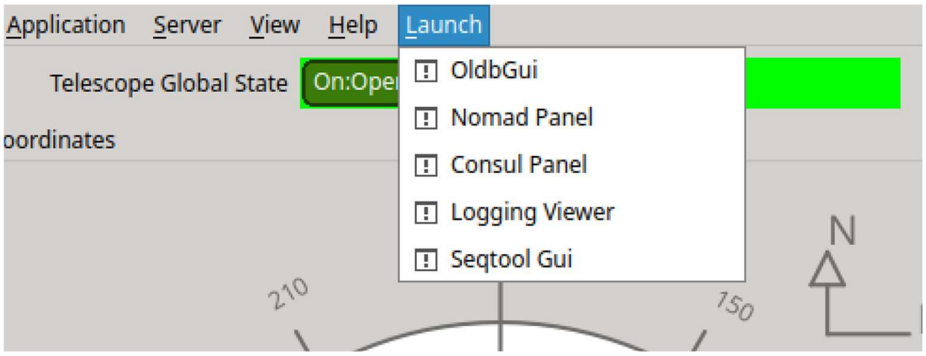


Figure 7.12

Each launcher entry is represented by an object in in "`tools.json`". During the application startup, these objects are used to create Launch menu entries.

```
{
    "tools": [
        {
            "name": "OldbGui",
            "command": "oldbGui",
            "param_type": "",
            "parameters": "",
            "description": "Starts Oldb Gui application"
        },
        {
            "name": "Nomad Panel",
            "command": "xdg-open",
            "param_type": "Environmental",
            "parameters": "$NOMAD_ADDR",
            "description": "Starts browser with Nomad panel"
        },
        {
            "name": "Seqtool Gui",
            "command": "seqtool gui",
            "param_type": "consulGetUri",
            "parameters": {
                "prefix": "--address",
                "service": "seqserver",
                "request": "ipport"
            },
            "description": "Starts browser with Consul panel"
        },
```

Figure 7.13

Following fields need to be defined for each launcher entry:

- **name** – tool name that is displayed in GUI
- **command** – command that is executed to launch a tool
- **param_type** – type of parameter that is passed with the command
- **parameters** – parameters of the command – depending on **param_type** this field is string or JSON object (explanation below)
- **description** – short description of the tool that shown in tooltip

It is necessary to define "**param_type**", because based on it parameters are interpreted. There are 3 possible values for "**param_type**" field:

- "consulGetUri" – set when parameter is actually an URI retrieved from Consul via command **consulGetUri** (example shown in Figure 15 for "Seqtool Gui"). Then "**parameters**" field is declared as JSON object with the following structure :

```
"parameters": {
        "prefix": string,
        "service": string,
        "request": string
}
```

- "Environmental" – set when parameter(s) is passed as plain string, e.g. in CLI. Individual parameters should be separated with space, whereas environmental variables which value has to retrieved first, should be typed with "$" sign (example shown in Figure 15 for "**Nomad Panel**")
- " " (empty) – set if command is invoked without parameters. In that case "**parameters**" filed should remain empty as well (example shown in Figure 15 for "**OldbGui**")

7.2 hlcctelsimui_mon documentation

This is an HLCC Simulator small monitoring UI, to be used as long as the main hlcctrelsim_ui panel is not implemented up to level of allow monitoring all simulator processes and OLDB attributes.

For the time being it provides only very limited functionality:

- Monitoring state of telif, telmon and eltpk rad applications
- Monitoring target and current positions as in the OLDB
- Monitoring a few flags

To start the application:

$ hlcctelsimui_mon

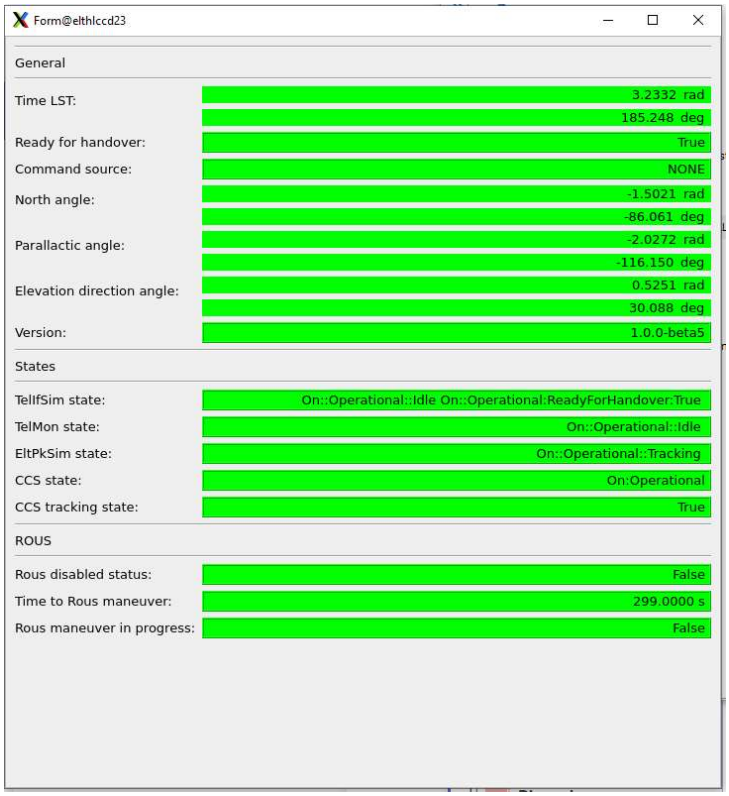


Figure 7.14

The content of the monitor panel can be tuned by changing the configuration file:
`hlcc/software/telsimui_mon/resource/config/telsimui_mon/monitors.json`
or its copy installed in INTROOT or in the RPM location.

7.3 telif documentation

This is the main entry point application to interact with the simulator.
All commands defined in the CCS-INS ICD are sent here.

7.3.1 Sending commands

To send a standard command using the `msgsend` utility:

```
$> msgsend --uri=zpb.rr://telif/StdCmds ::stdif::StdCmds::GetState
```

To send a specific command defined in the `::ccsinsif::Commands:` interface:

```
$> msgsend --uri=zpb.rr://telif/Commands
::ccsinsif::Commands::RequestControl '"CASCADE"'
```

To send a specific command defined in the `::telifif::SimCmds:` interface:

```
$> msgsend --uri=zpb.rr://telif/TelifCmds
::telifif::TelifCmds::MoveToNamedPos '"PARK"'

$> msgsend --uri=zpb.rr://telif/TelifCmds
::telifif::TelifCmds::MoveToAltAzPos '{"alt": 0.5, "az": 0.2}'

$> msgsend --uri=zpb.rr://telif/AppCmds ::appif::AppCmds::GetConfig
'"cfg/pub/dds/profile cfg/rous/timer_period_s"'

$> msgsend --uri=zpb.rr://telif/AppCmds ::appif::AppCmds::SetConfig
'"cfg/rous/timer_period_s: 123"'
```

The standard commands supported are described in this ICD:

- std/if/src/stdif.xml · master · ecs / ecs-interfaces · GitLab (eso.org)ul

The standard commands supported by all RAD Applications are described in this icd:

- rad/cpp/appif/src/appif.xml · master · ifw / rad · GitLab (eso.org)

The specific ccs-ins interface commands supported are described in this ICD:

- ccsinsif/icd/src/ccsinsif.xml · master · ccs / hlcc · GitLab (eso.org)

Additional commands to provide control of the simulation are described in this ICD:

- telif/telifsim/telifsimif/icd/src/telifsimif.xml · master · ccs / hlcc · GitLab (eso.org)

The application also support the metadaq commands described in this ICD:

- metadaq/if/src/metadaqif.xml · master · ecs / ecs-interfaces · GitLab (eso.org)

The `preset` command has a parameter's structure that is currently not supported by the msgsend tool.

The python utility **telif_preset** allows to send the command in an easy way and can be used as an example to write more complex code. See HLCC utilities.

You can find a set of additional python examples in the jupyter notebooks.

7.3.2 telif State Machine

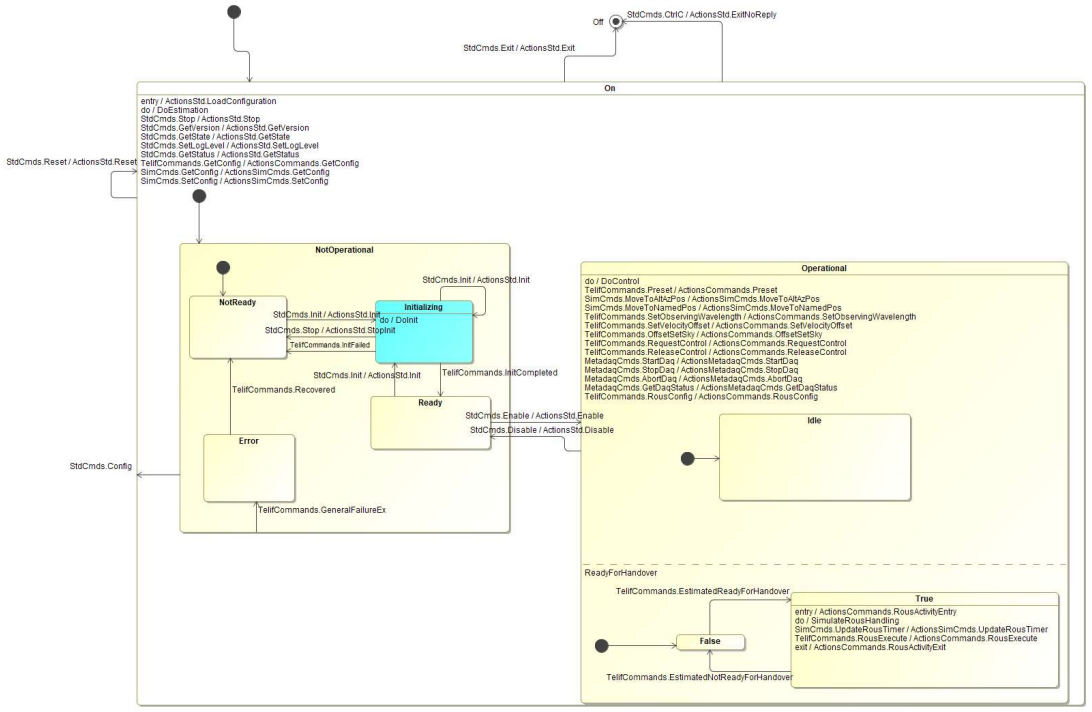


Figure 7.15

7.3.3 Rous Documentation

7.3.3.1 Rous Description

Rous takes place when the telescope is tracking and providing good image quality.

Recurrent Optimization of Unit's Stroke (ROUS) events can be disabled or executed immediately to align with observation needs. ROUS corrects the quasistatic aberrations and manages the distribution of strokes between the internal degrees of freedom. The dynamics of the positioning units are specified such that the transient perturbations remain within the limits described in [RD3], and it is expected that the instrument side control loops are robust against those disturbances

- The simulator executes a Rous maneuver at a given interval of time (specified in the configuration database).
- Commands:

- `RousConfig ENABLE/DISABLE`
  command that disables/avoids the automatic execution of ROUS maneuvers by CCS. Exact implementation to be agreed during design. It must be understood that delaying a ROUS cycle comes with the risk of degraded image quality.
  ENABLE = ROUS maneuvers are executed as scheduled by CCS
  DISABLE = Automatic execution of ROUS maneuvers is disabled
  Note that the telescope may force a ROUS maneuver if deemed necessary for system safety.
  For example:
  ```
  $ msgsend --uri=zpb.rr://telif/Commands
  ::ccsinsif::Commands::RousConfig '"ENABLE"'
  ```
- `RousExecute`
  command that executes a ROUS maneuver immediately.
  ===> This resets the counter
  For example:
  ```
  $ msgsend --uri=zpb.rr://telif/Commands
  ::ccsinsif::Commands::RousExecute
  ```

- Monitor signals on Control Network:
  - `rous:counter_act`
    Time to next ROUS maneuver. The algorithm and the amount of pre-warning time before the next ROUS are TBD.
    ===> The Rous control loop manages the counter
  - `rous:status`
    Boolean
    Status flag indicating whether ROUS is enabled or disabled:
    0 = enabled (i.e. not disabled)
    1 = disabled
    1 Hz
  - `rous:maneuver_in_progress`
    Boolean
    Duplicate of the ROUS maneuver status flag in ao:m4_setpoint_applied indicating whether a ROUS maneuver is currently being executed by CCS:
    0 = no maneuver
    1 = ROUS maneuver in progress

7.3.3.2 Rous Operation

By default Rous is enabled and as soon the telescope starts tracking, Rous starts its operation:

1. Loads Rous timer period from configuration
2. Waits until timer expires
3. Executes Rous Maneuver (currently Rous maneuver is not defined so Rous stays in this state for 1 second, so its possible to observe the state change)
4. Starts from step 1 again.

When Rous is disabled it resets the Rous timer and idles until it is enabled again.

To change the Rous disabled state, use the hlcctelsimui 'Ccsinsif' tab. See Figure 7.16 below:

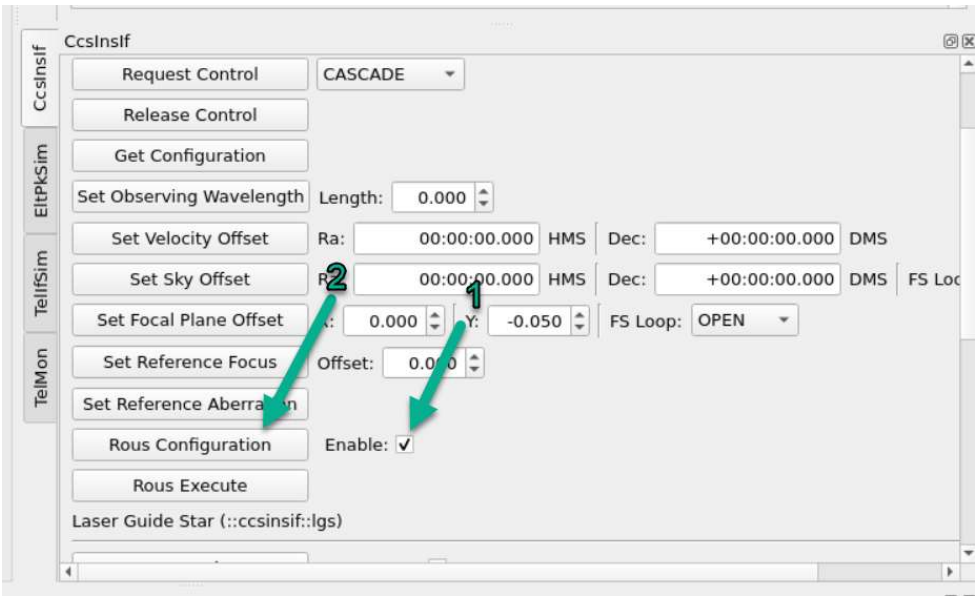


Figure 7.16

Set first the Rous status in the '***Enable***' box and then press '***Rous Configuration***': only when the button is pressed the command is sent to telif to change state.

Regardless of the Rous Enabled state or operation phase when Enabled, a Rous maneuver can be started manually by the operator by pressing in the '***Rous Execute***' Button. See picture above.

If Rous is enabled it will reset the Rous timer and start a new cycle.

If Rous is disabled it will return to the disabled state after the rous maneuver.

Its possible to check the current Rous status by observing the Rous information in the Oldb. That information is updated every second. See Figure 7.17 below.

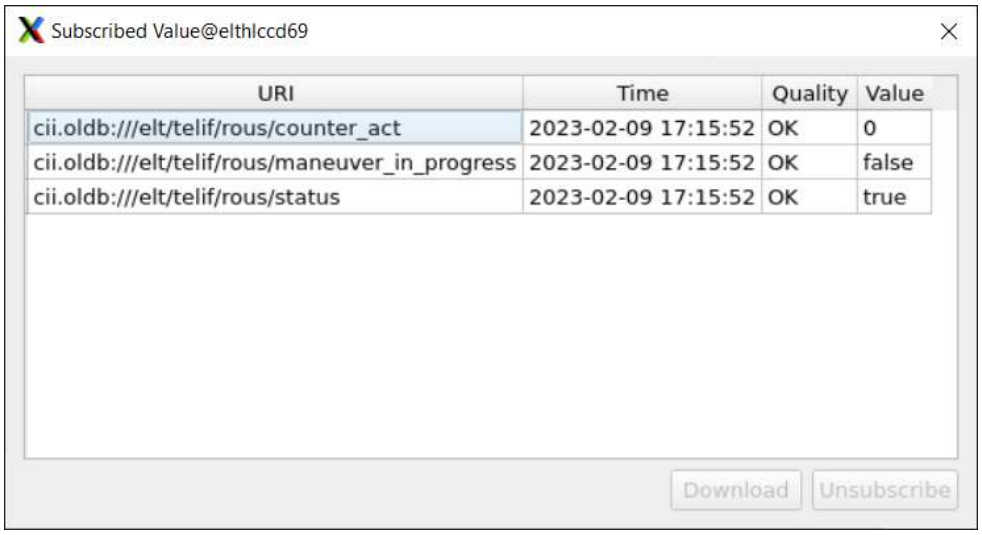


Figure 7.17

The ***status*** datapoint in the oldb is in fact the disabled status, meaning disabled when true.

7.3.4 Preset Documentation

From the telif perspective, the Preset operation can now be configured to run in one of multiple possible modes.

For usage by instrumentation, the default mode is sufficient.

The selection of modes is to explore the various possibilities and foster discussion with the users to converge toward an agreed strategy and will be used mainly with the dev and ecm deployments. Check below the available preset modes.

7.3.4.1 HEADLESS mode

The system will preset without involving the operator, with a fully automatic sequence. Essentially it will run a pre-defined sequence that is supposed to take care of the complete preset procedure.

The sequence is started using the `seqtool run` command, i.e. with a standalone instance of the sequencer.

For this mode the pre-defined sequence should not require any user interaction, otherwise the sequence will block indefinitely. Check further below how to change/define the sequence to be executed.

7.3.4.2 GUI mode

Will load the sequence in the common instance of the Sequencer server (*seqserver*) that is started by the Nomad job that starts all basic HLCC processes.

To do this telif:

- Queries Consul for the IP and "telnet" port of the *seqserver* service/process
- The *seqserver* provides a private "telnet" interface to be used by the GUI that allows to interact with it. Using this interface....
    - load the sequence.
    - run the sequence.

      Since the sequence will request operator interaction, it is necessary to have running a sequencer GUI.

      The sequencer GUI can now be started with the command:

      ```
      seqtool gui --address `consulGetUri -s seqserver -r ipport`
      ```

      that will ask Consul for the seqserver IP and port. If the sequencer GUI is running and connected.
    - The loaded sequence will appear in the Gui.
    - You will see it started.
    - It is advisable to have a dialog requesting the user input to continue with the sequence execution.

7.3.4.3 Preset Sequences/OBs

Currently we have two engineering OBs available as initial templates for discussion:

- hlccPreset.json – This OB is built to run automatically without any user intervention. It can be used in 'HEADLESS' or 'GUI' modes.
- hlccPresetInteractive.json – This OB should only be configured to run in 'GUI' mode because it will request user interaction and a Sequencer Gui shall be available otherwise the OB will hang indefinitely. It implements the same behavior of hlccPreset but pops up a dialog at the beginning asking for user input.

7.3.4.4 Sequence and mode configuration

To define how the Preset sequence/OB will be executed we must configure correctly the preset which now is done in two Oldb datapoints. To change them we can edit directly the Oldb or use the correspondent telif commands.

- Sequence mode
    - Oldb address
        - `cii.oldb:///elt/cs/hlcc/telif/preset/mode`
    - Available Modes:
        - HEADLESS

        - GUI
    - Telif command:
        - `msgsend --uri=zpb.rr://telif/TelifCmds ::telifif::TelifCmds::SetPresetSequenceMode '"GUI"'`
        - `msgsend --uri=zpb.rr://telif/TelifCmds ::telifif::TelifCmds::SetPresetSequenceMode '"HEADLESS"'`
- Sequence/OB script
    - Oldb Address:
        - `cii.oldb:///elt/cs/hlcc/telif/preset/sequence`
    - Default sequence
        - `hlccseq.hlccPreset`
    - Telif Command
        - `msgsend --uri=zpb.rr://telif/TelifCmds ::telifif::TelifCmds::SetPresetSequence '"config/seq/hlccPreset.json"'`

7.4 telmon documentation

This is the telescope monitoring applications.

Once started, the estimators (implemented as python scripts) periodically estimate the status of CCS and publish all status information available.

7.4.1 Sending commands

To send a standard command using the `msgsend` utility:

```
$> msgsend --uri=zpb.rr://telmon/StdCmds
::stdif::StdCmds::GetStatus
```

To send a comand defined in the RAD Application standard interface:

```
$> msgsend --uri=zpb.rr://telmon/AppCmds
::appif::AppCmds::GetConfig '"cfg/estimation_period_ms"'
```

To send a specific command defined in the `::telmon::MonCmds:` interface:

```
$> msgsend --uri=zpb.rr://telmon/MonCmds
::telmonif::MonCmds::Reload
```

In a deployment managed by nomad/consul the uri to be used is retrieved with a query to consul using the consulGetUri command line utility instead of being hardcoded.

The standard commands supported are described in this ICD:

- std/if/src/stdif.xml · master · ecs / ecs-interfaces · GitLab (eso.org)

The specific interface commands supported are described in this ICD:

- telif/telmon/telmonif/icd/src/telmonif.xml · master · ccs / hlcc · GitLab (eso.org)

For example,

to **change the ignore flag** of a monitored application:

```
 $> msgsend --uri=zpb.rr://telmon/MonCmds
::telmonif::MonCmds::SetAppIgnore '{"app_name": "telif", "ignore":
false}'
```

to **change the ignore flag** of a monitored subsystem and all its applications:

```
 $> msgsend --uri=zpb.rr://telmon/MonCmds
::telmonif::MonCmds::SetSubsystemIgnore '{"subsystem_name": "hlcc",
"ignore": false}'
```

To reload the the monitoring estimators/scripts:

```
 $> msgsend --uri=zpb.rr://telmon/MonCmds ::telmonif::MonCmds::Reload
```

to retrieve a configuration parameter:

```
 $>  msgsend  --uri=zpb.rr://telmon/AppCmds  ::appif::AppCmds::GetConfig
'"cfg/estimation_period_ms"'
   REPLY: "cfg: \n  estimation_period_ms: 1000\n"
```

to set a configuration parameter:

```
   $>  msgsend  --uri=zpb.rr://telmon/AppCmds  ::appif::AppCmds::SetConfig
'"cfg/estimation_period_ms: 1500"'
```

to load confiuration from file:

```
   $> msgsend --uri=zpb.rr://telmon/AppCmds ::appif::AppCmds::LoadConfig
'"config/telmon/config.yaml"'
```

7.4.2 telmon State Machine

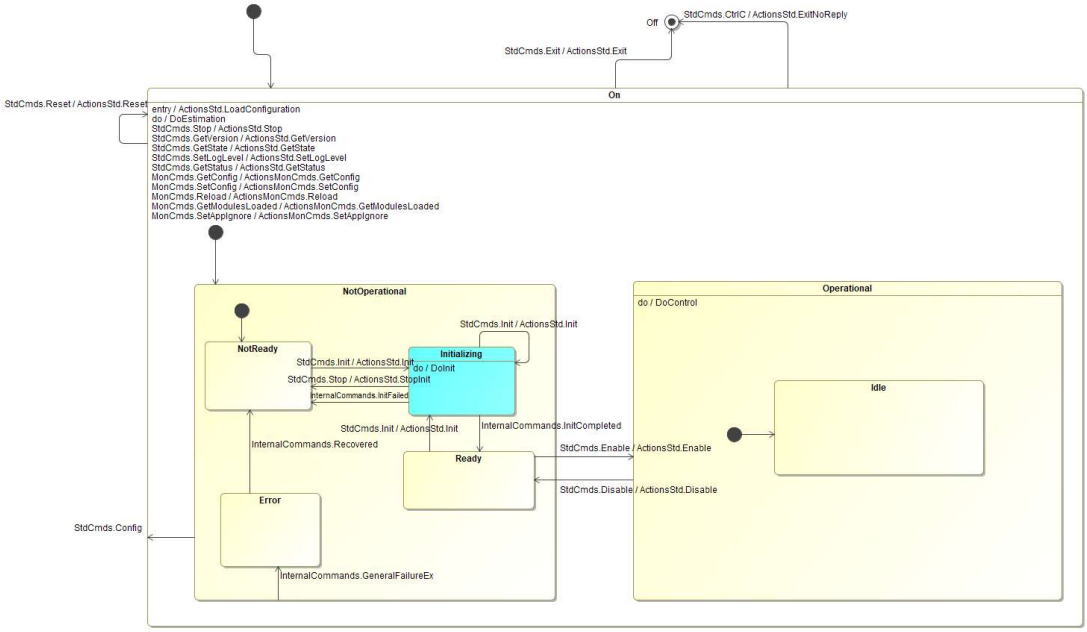


Figure 7.18

7.4.3 Adding new estimation scripts to Telmon

Estimation scripts will be loaded and executed by telmon, which is using the pybind11 dynamic python interpreter.

7.4.3.1 Telmon directory structure

The scripts directly distributed with the HLCC package are stored together with the telmon code:

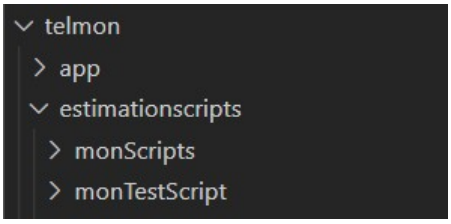


Figure 7.19

- The `app` directory contains the 'telmon' c++ source files.

- The `estimationscripts/monTestScripts` contains scripts only used in testing.
- The `estimationscripts/monScripts` contains all the estimation scripts executed by telmon.
  These can be used as good examples when creating new scripts.

7.4.3.2 Adding a new python script file

To add a new estimation script, create a python file (in the estimationscripts/monScripts/src/MonScripts directory if it has to be part of the telmon distribution) with a descriptive name.

The file should contain a class with the name matching the name of the file, the class should contain at least two methods which will be called by the telmon application:

- `def Execute(self, arg):` This method will be executed at every estimation cycle, it takes one argument which is a dictionary shared between all scripts and the cpp application as a `pybind::dict` object. It can be used by each estimation script to store data which will persist while the telmon estimation activity is running, even if the python script restarts.
- `def Terminate(self, arg):` This method will be executed when the telmon estimation activity is terminating (for example when the application is exiting). The purpose is to allow the estimation script to clean up in the end. It also takes one argument which is the same object argument in the Execute method.

7.4.4 Listing the scripts to be run by telmon

Configuration for the script is in:

`resource/config/telmon/config.yaml'`

Add the new estimation script to the list in `cfg: estim_scripts:`

The name of the script should be preceeded by the name of main module ('MonScripts' for the scripts distributed with telmon); check the example below:

```
cfg:
  estim_scripts                :
                                 - MonScripts.ReadyForHandoverEstimation
                                 - MonScripts.CcsStateEstimation
                                 - MonScripts.TrackingEstimation
                                 - MonScripts.SegmentExchangeModeEstimation
```

Figure 7.20

After restarting the application (reset command for example) the new script will be picked up and executed by telmon.

7.4.5 Reloading Scripts

For the time being, there is a command (see the telmonIf interface) for reloading the scripts that are already running, *it will not integrate new commands*. This command will not restart the telmon application or reload the configuration, it just reloads all the scripts that were in the configuration file when the application started. By "reloading" it means that it will pick any code changes that were applied to the script.

It is a fast way to develop and test scripts:

```
$> msgsend --uri zpb.rr://localhost:11003/MonCmds ::telmonif::MonCmds::Reload
```

7.4.6 Error handling

If python scripts have any problem or the class / method names are not correct, the telmon application will not load the script and will log an Error message for each script not successfully loaded.

After being loaded, if scripts throw any exception it will be logged by telmon but will not impact the execution of other scripts.

7.4.7 Telmon Estimation Scripts

The telmon application supports installing python scripts to be used for the estimation of state variables for the telescope. It is in this way possible to easily and dynamically define status information of interest for users of the system, without having to modify the code of the applications.

7.4.7.1 Ccs state estimation

This estimation script estimates the global ccs state by combing the state of all applications listed in the Oldb uri '*cii.oldb:///elt/cs/hlcc/telmon/cfg/monitored_apps*'.

The applications with the ignore flag = true will be left out of the ccs state estimation.

The ccs state estimation is then published in the Oldb uri '*cii.oldb:///elt/cs/telif/ccs/state*'. It is also being published in dds '*dds.ps://1://TELIF_STATUS'*.

It can be found in the telescope simulator monitor panel:

7.4.7.2 Ready for handover estimation

It estimates if the telescope is ready to handover the control to instruments.

The ready for handover estimation is then published in the Old uri '*cii.oldb:///elt/cs/telif/ccs/ready_for_handover'*. It is also being published in dds '*dds.ps://1/TELIF_RDY_FOR_HANDOVER'*.

It can be found in the telescope simulator monitor panel:

7.4.7.3 Segment exchange mode estimation

Estimates if the telescope is in Segment exchange mode and publishes the estimation in '*cii.oldb:///elt/cs/hlcc/telmon/mon/segment_exchange_mode*'

It can be found in the telescope simulator monitor panel:

7.4.7.4 Tracking estimation

Estimate the tracking state of the telescope and publishes the result in '*cii.oldb:///elt/cs/hlcc/telmon/mon/tracking'*.

It can be found in the telescope simulator monitor panel:

7.4.7.5 Night time estimation

Estimates if the telescope is in night time mode and publishes the estimation result in 'cii.oldb:///elt/cs/telif/ccs/night_time'.

It can be found in the telescope simulator monitor panel:

7.5 eltpk documentation

This is the pointing and tracking high level coordination process.

When running in simulation mode, it will communicate with the **`trksim`** process, that is implementing a tracking control loop (this is currently done in the INS and DEV deployments).

When running in "real telescope mode", it will instead communicate with the Main Structure LSV (and later on with other subsystem involved in pointing and tracking.)

7.5.1 Sending commands

To send a standard command using the msgsend utility:

```
$> msgsend --uri=zpb.rr://eltpk/StdCmds ::stdif::StdCmds::GetState
$> msgsend --uri=zpb.rr://eltpk/StdCmds ::stdif::StdCmds::Exit
```

To send a specific command defined in the `::eltpkif::PointingKernelCommands` interface:

```
$> msgsend --uri=zpb.rr://eltpk/Commands
::eltpkif::PointingKernelCommands::SetTargetAltAzPos '{"alt": 0.2,
"az": 0.1}'
```

To send a comand defined in the RAD Application standard interface:

```
$> msgsend --uri=zpb.rr://eltpk/AppCmds ::appif::AppCmds::GetConfig
'"cfg/procname"'

REPLY: "cfg: \n  procname: \"eltpk\"\n"

$> msgsend --uri=zpb.rr://eltpk/AppCmds ::appif::AppCmds::SetConfig
'"cfg/params/alt_speed_deg_per_s: 7.5"'

REPLY: "OK"

msgsend --uri=zpb.rr://eltpk/Commands
::eltpkif::PointingKernelCommands::SetTargetRaDec '{"command":
"FULL_PRESET", "preset_data": {"ra":5.5003, "dec":-1.5192,
"system":"J2000.0", "proper_motion_ra":0.0,
"proper_motion_dec":0.0, "epoch":"J2000.0", "parallax":0.0,
"radvel":0.0, "rshift":0.0, "velocity_offset_ra":0.0,
"velocity_offset_dec":0.0, "object_name":"Polaris Australis",
"guide_stars":[] }}'
```

The standard commands supported are described in this ICD:

- std/if/src/stdif.xml · master · ecs / ecs-interfaces · GitLab (eso.org)

The standard commands supported by all RAD Applications are described in this icd:

- rad/cpp/appif/src/appif.xml · master · ifw / rad · GitLab (eso.org)

The specific commands supported are described in this ICD:

- https://gitlab.eso.org/ccs/hlcc/-/blob/master/software/telif/eltpk/eltpkif/icd/src/eltpkif.xml

The preset command has a complex parameter structure that makes it look awkward when sending it to the eltpk application (method ::eltpkif::PointingKernelCommands::SetTargetRaDec) using the msgsend tool, for example with argument

```
'{"command": "FULL_PRESET", "preset_data": {"ra": 5.5003, "dec": -1.5192,
"system": "J2000.0", "proper_motion_ra": 0, "proper_motion_dec": 0,
"epoch": "J2000.0", "parallax": 0, "radvel": 0, "rshift": 0,
"velocity_offset_ra": 0, "velocity_offset_dec": 0, "object_name":
"Polaris Australis", "guide_stars": []}}'
```

Instead, for most uses we recommend to use the jupyter notebook EltpkPreset (see 7.12), which allows to send the command in an easy way and can be used as an example to write more complex code.

You can find a set of additional python examples in the jupyter notebooks

7.5.2 eltpk State Machine

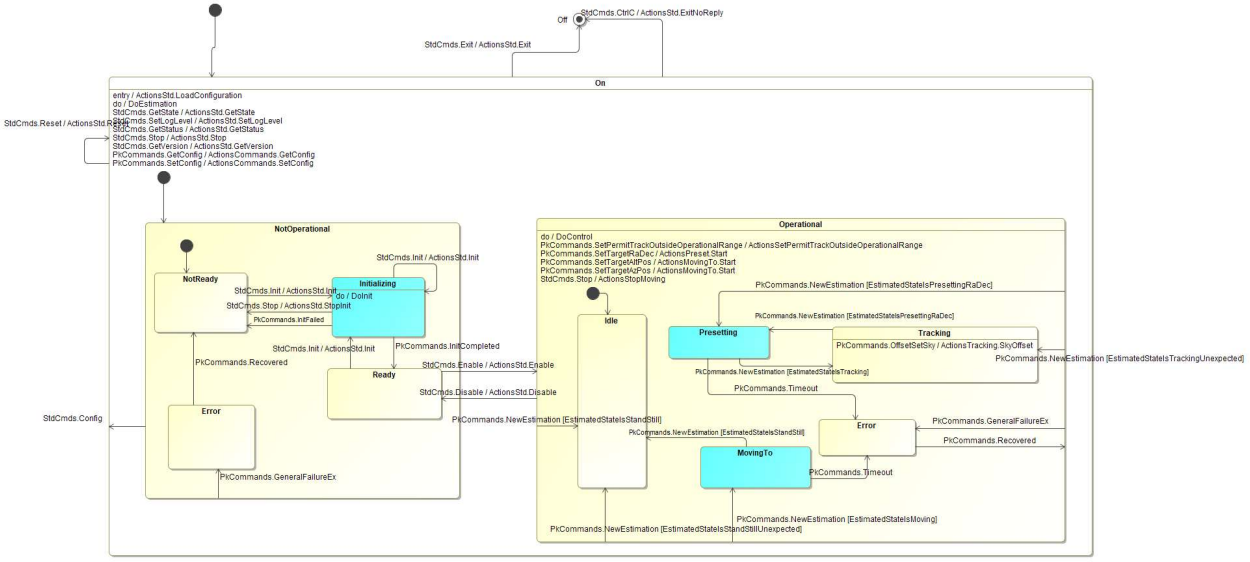


Figure 7.21

7.5.3 Configuration options

Application configuration is in:

```
[<INTROOT>|/elt/hlcc]/resource/config/eltpk/config.yaml
```

- Networking
  - The eltpk publishes positions to a multicast address on the deterministic network. Per the factory settings, this publishing does not leave the localhost. To enable traffic to other hosts, set a real multicast-enabled network interface.
    For example: `cfg.pub.determ.nic = 192.168.100.161`
  - The eltpk publishes measurements to the non-deterministic network. Per the factory settings, this publishing does not leave the localhost. To enable traffic to other hosts, unset the DDS profile that is responsible for this.
    For example: `cfg.pub.dds.profile = ""`
- Support for PTP synchronization
  - If health checks by the ELT Time Reference System (TRS) are enabled in the configuration (`cfg.trs_health_enabled = 1`), then eltpk will receive notifications about the PTP health status.
  - Presetting and Tracking will then only be possible when PTP health is good:

- The preset (SetTargetRaDec) command gets rejected otherwise;
- An ongoing preset or tracking will be stopped when PTP health goes bad, and the state machine will consequently go to state Operational/Idle.

  - The OLDB shows
    - Configuration: cii.oldb:///elt/cs/hlcc/eltpk/cfg/trs_health_enabled
    - Actual health: cii.oldb:///elt/cs/hlcc/eltpk/mon/trs/health
    - Reason for health state: cii.oldb:///elt/cs/hlcc/eltpk/mon/trs/reason

- Tracking simulation mode
  - If simulation of MS LSv is enabled (`cfg.simulate_lsv_ms = 1`), then eltpk will communicate with the trksim process and not with the MS Mount controller.
  - The OLDB shows:
    - Configuration: `cii.oldb:///elt/cs/hlcc/eltpk/cfg/simulate_lsv_ms`
  - It is possible to check the simulation configuration also using the command:

    ```
    $ msgsend --uri=zpb.rr://eltpk/AppCmds
    ::appif::AppCmds::GetConfig '"cfg/simulate_lsv_ms"'
    ```

  - This option is normally set to `true` in the dev and ins deployments, assuming that the ms software is not running, while the trksim is. It can be used to create test deployment configurations.
  - It is possible to change mode at run time using the `SetConfig` command for specific testing purposes (see for example the internal OneNote page: Switching HLCC from internal simulation to interfacing with the MS (Web view))

7.6 trksim documentation

This is the pointing kernel simulation process, implementing the tracking and pointing control loops.

It is used in tracking simulation mode and at the moment it implements the subset MS LSV interfaces used by the eltpk to talk to the MS LSV. It will be extended to cover simulation of the PFS and other subsystems.

7.6.1 Sending commands

To send a standard command using the msgsend utility:

```
$> msgsend --uri=zpb.rr://trksim/StdCmds ::stdif::StdCmds::GetState
$> msgsend --uri=zpb.rr://trksim/StdCmds ::stdif::StdCmds::Exit
```

To send a comand defined in the RAD Application standard interface:

```
$> msgsend --uri=zpb.rr://trksim/AppCmds
::appif::AppCmds::GetConfig '"cfg/procname"'
"cfg/procname : trksim\n"
```

The standard commands supported are described in this ICD:

- std/if/src/stdif.xml · master · ecs / ecs-interfaces · GitLab (eso.org)

The standard commands supported by all RAD Applications are described in this icd:

- rad/cpp/appif/src/appif.xml · master · ifw / rad · GitLab (eso.org)

The MS lsv commands that are (partially) supported are described in this ICD:

- https://gitlab.eso.org/lsv/ms/-/blob/main/interface/msif/src/msif.xml

The specific simulation commands supported are described in this ICD:

- software/trksim/interface/trksimif/icd/src/trksimif.xml · master · ccs / hlcc · GitLab (eso.org)

7.6.2 trksim State Machine

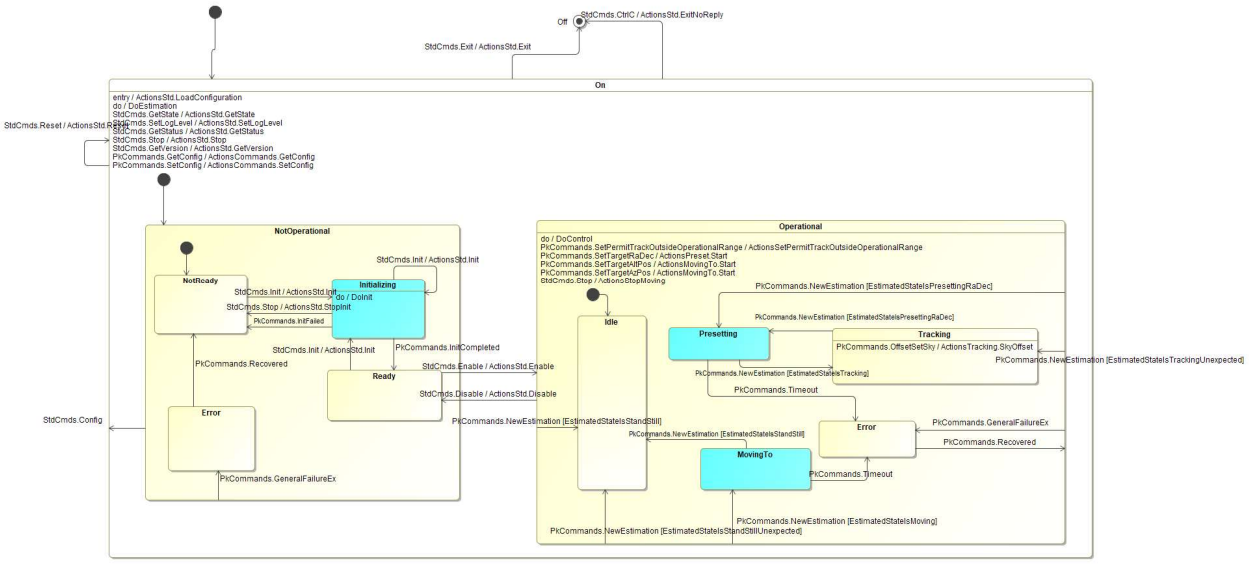


Figure 7.22

7.7 pfssimhlcc documentation

This is the pre-focal station simulation process, implementing receiving guide probes commands.

It is used to receive GpCmds from eltpk during presets.

7.7.1 Sending commands

To send a standard command using the msgsend utility:

```
$> msgsend --uri=zpb.rr://pfssimhlcc/StdCmds
::stdif::StdCmds::GetState

$> msgsend  --uri=zpb.rr://pfssimhlcc/StdCmds
::stdif::StdCmds::Exit
```

To send a comand defined in the RAD Application standard interface:

```
$> msgsend --uri=zpb.rr://pfssimhlcc/AppCmds
::appif::AppCmds::GetConfig '"cfg/procname"'

REPLY: "cfg/procname : pfssimhlcc\n"
```

The standard commands supported are described in this ICD:

- std/if/src/stdif.xml · master · ecs / ecs-interfaces · GitLab (eso.org)

The standard commands supported by all RAD Applications are described in this icd:

- rad/cpp/appif/src/appif.xml · master · ifw / rad · GitLab (eso.org)

The lsv pfs Guide Probes commands that are (partially) supported are described in this ICD:

- interface/pfsif/src/pfsif.xml · main · lsv / pfs · GitLab (eso.org)

The specific simulation commands supported are described in this ICD:

- https://gitlab.eso.org/ccs/hlcc/-/blob/master/software/pfssimhlcc/interface/pfssimhlccif/icd/src/pfssimhlccif.xml

7.8 lsvsim documentation

lsvsim is a generic process that can be used to simulate the behavior of an LSV or of other processes that HLCC might need to interface to.

The present implementation provides minimal features and can be extended in the future, if needed, to provide more features.

Currently, the DEV and the ECM deployments include some lsvsim processes, for the example a process instantiated as **`astrositemon`** to provide a basic emulation of an ELT Armazones Site Monitor. The following examples will be based on this lsvsim instance.

7.8.1 Sending Commands

The standard commands supported are described in this ICD:

- std/if/src/stdif.xml · master · ecs / ecs-interfaces · GitLab (eso.org)

To send a command:

```
$> msgsend --uri=zpb.rr://astrositemon/StdCmds
::stdif::StdCmds::GetState
```

Lsvsim app implements the comands defined in the RAD Application standard interface:

- rad/cpp/appif/src/appif.xml · master · ifw / rad · GitLab (eso.org)

To send a command:

```
$> msgsend --uri=zpb.rr://astrositemon/AppCmds
::appif::AppCmds::GetConfig '"cfg/sim_activity_period_ms"'
```

Check the list below and also examples on how to send them using the msgsend:

7.8.1.1.1 SetConfig

Writes, in process configuration, the value(s) for the given configuration parameter(s).

```
$>  msgsend --uri=zpb.rr://astrositemon/AppCmds
::appif::AppCmds::SetConfig '"cfg/sim_activity_period_ms : 120"'
```

7.8.1.1.2 GetConfig

Retrieves the value(s) of the given configuration parameter(s) identified via the key(s). If multiple keys are given they must be ‘‘space’’ separated, if no keys are given (empty string) the whole configuration will be returned.

```
$> msgsend --uri=zpb.rr://astrositemon/AppCmds
::appif::AppCmds::GetConfig '"cfg/sim_activity_period_ms"'
```

7.8.1.1.3 LoadConfig

Writes, in process configuration, the value(s) for the given configuration file.

```
$> msgsend --uri=zpb.rr://astrositemon/AppCmds
::appif::AppCmds::LoadConfig '"config/lsvsim/config_m1.yam"'
```

7.8.1.1.4 LoadStateMachine

It loads a new State Machine model from file. If the loading fails, the old one is restored.

```
$> msgsend --uri=zpb.rr://astrositemon/AppCmds
::appif::AppCmds::LoadStateMachine '"lsvsim/sm.xml"'
```

7.8.1.2 Lsvsim interface

Lsvsim app implement the lsvsim interface commands that are described in this interface:

- lsvsim/interface/lsvsimif/icd/src/lsvsimif.xml · master · ccs / hlcc · GitLab (eso.org)

Check the list below and also examples on how to send them using the msgsend:

7.8.1.2.1 SetSim

Writes, in configuration parameter"cfg.sv_specific_confi", the value given, it is possible to change only one value per command and only scalar nodes are allowed to change. This command is similar to the SetConfig command from the lsv interface but in this case it will change the second layer Yaml configuration contained in the config parameter cfg.sv_specific_config, this configuration will be used by the SVs simulation scripts.

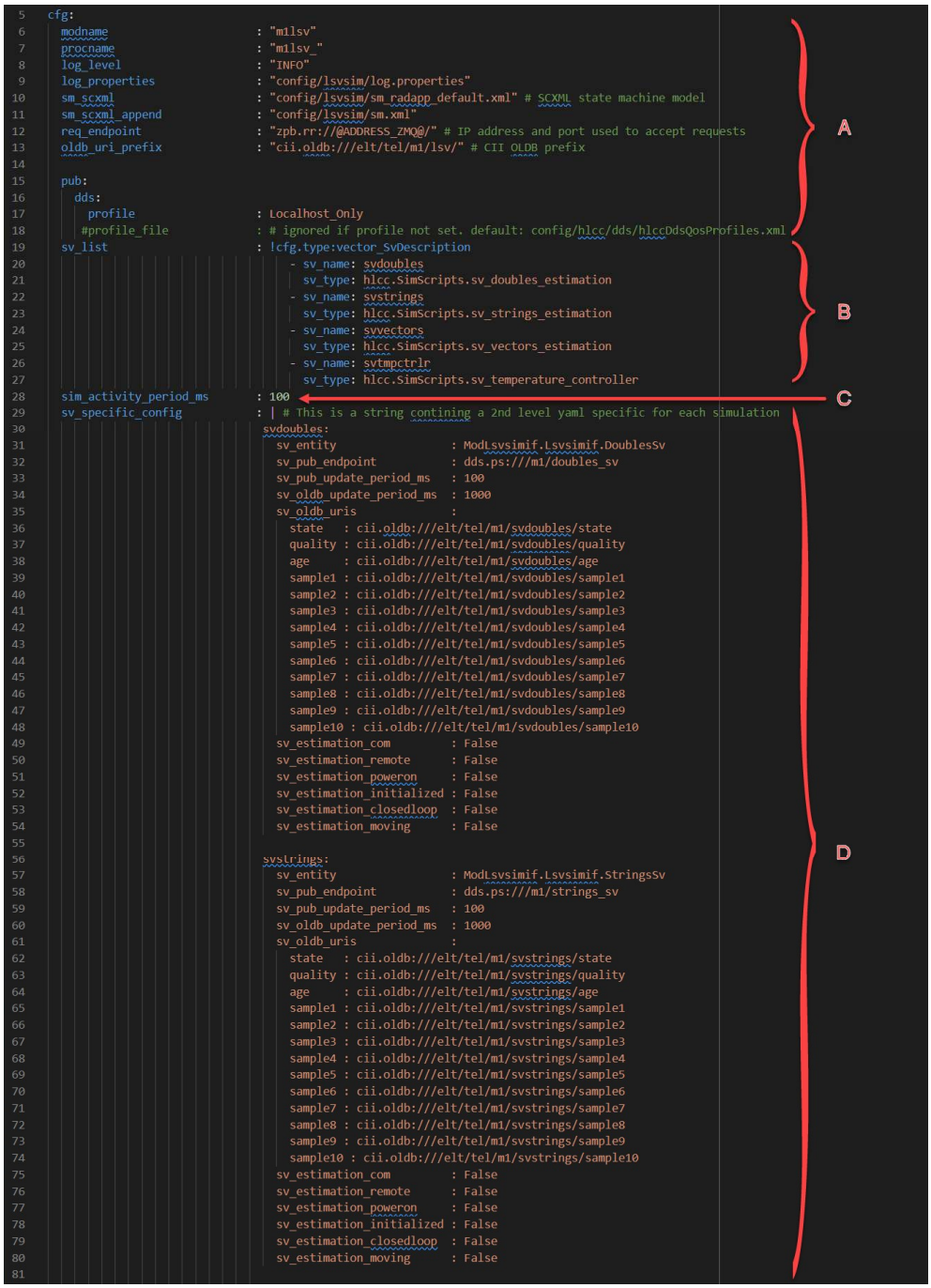
```
$> msgsend --uri=zpb.rr://astrositemon/SimCmds
::lsvsimif::SimCmds::SetSim
'"astrositemon.site_dps.air_temperature.value: 200.0"'
```

7.8.2 Configuration

Check below how the configuration of the lsvsim application looks like.

The file attached below is actually the configuration file

software/lsvsim/app-config/src/config_m1.yaml.in · master · ccs / hlcc · GitLab (eso.org)




```
cfg:
  modname                  : "m1lsv"
  procname                 : "m1lsv_"
  log_level                : "INFO"
  log_properties           : "config/lsvsim/log.properties"
  sm_scxml                 : "config/lsvsim/sm_radapp_default.xml" # SCXML state machine model
  sm_scxml_append          : "config/lsvsim/sm.xml"
  req_endpoint             : "zpb.rr://@ADDRESS_ZMQ@/" # IP address and port used to accept requests
  oldb_uri_prefix          : "cii.oldb:///elt/tel/m1/lsv/" # CII OLDB prefix

  pub:
    dds:
      profile              : Localhost_Only
     #profile_file          : # ignored if profile not set. default: config/hlcc/dds/hlccDdsQosProfiles.xml
  sv_list                  : !cfg.type:vector_SvDescription
                               - sv_name: svdoubles
                                 sv_type: hlcc.SimScripts.sv_doubles_estimation
                               - sv_name: svstrings
                                 sv_type: hlcc.SimScripts.sv_strings_estimation
                               - sv_name: svvectors
                                 sv_type: hlcc.SimScripts.sv_vectors_estimation
                               - sv_name: svtmpctrlr
                                 sv_type: hlcc.SimScripts.sv_temperature_controller
  sim_activity_period_ms   : 100
  sv_specific_config       : | # This is a string contining a 2nd level yaml specific for each simulation
                             svdoubles:
                               sv_entity                : ModLsvsimif.Lsvsimif.DoublesSv
                               sv_pub_endpoint          : dds.ps:///m1/doubles_sv
                               sv_pub_update_period_ms  : 100
                               sv_oldb_update_period_ms : 1000
                               sv_oldb_uris             :
                                 state   : cii.oldb:///elt/tel/m1/svdoubles/state
                                 quality : cii.oldb:///elt/tel/m1/svdoubles/quality
                                 age     : cii.oldb:///elt/tel/m1/svdoubles/age
                                 sample1 : cii.oldb:///elt/tel/m1/svdoubles/sample1
                                 sample2 : cii.oldb:///elt/tel/m1/svdoubles/sample2
                                 sample3 : cii.oldb:///elt/tel/m1/svdoubles/sample3
                                 sample4 : cii.oldb:///elt/tel/m1/svdoubles/sample4
                                 sample5 : cii.oldb:///elt/tel/m1/svdoubles/sample5
                                 sample6 : cii.oldb:///elt/tel/m1/svdoubles/sample6
                                 sample7 : cii.oldb:///elt/tel/m1/svdoubles/sample7
                                 sample8 : cii.oldb:///elt/tel/m1/svdoubles/sample8
                                 sample9 : cii.oldb:///elt/tel/m1/svdoubles/sample9
                                 sample10 : cii.oldb:///elt/tel/m1/svdoubles/sample10
                               sv_estimation_com        : False
                               sv_estimation_remote     : False
                               sv_estimation_poweron    : False
                               sv_estimation_initialized : False
                               sv_estimation_closedloop : False
                               sv_estimation_moving     : False

                             svstrings:
                               sv_entity                : ModLsvsimif.Lsvsimif.StringsSv
                               sv_pub_endpoint          : dds.ps:///m1/strings_sv
                               sv_pub_update_period_ms  : 100
                               sv_oldb_update_period_ms : 1000
                               sv_oldb_uris             :
                                 state   : cii.oldb:///elt/tel/m1/svstrings/state
                                 quality : cii.oldb:///elt/tel/m1/svstrings/quality
                                 age     : cii.oldb:///elt/tel/m1/svstrings/age
                                 sample1 : cii.oldb:///elt/tel/m1/svstrings/sample1
                                 sample2 : cii.oldb:///elt/tel/m1/svstrings/sample2
                                 sample3 : cii.oldb:///elt/tel/m1/svstrings/sample3
                                 sample4 : cii.oldb:///elt/tel/m1/svstrings/sample4
                                 sample5 : cii.oldb:///elt/tel/m1/svstrings/sample5
                                 sample6 : cii.oldb:///elt/tel/m1/svstrings/sample6
                                 sample7 : cii.oldb:///elt/tel/m1/svstrings/sample7
                                 sample8 : cii.oldb:///elt/tel/m1/svstrings/sample8
                                 sample9 : cii.oldb:///elt/tel/m1/svstrings/sample9
                                 sample10 : cii.oldb:///elt/tel/m1/svstrings/sample10
                               sv_estimation_com        : False
                               sv_estimation_remote     : False
                               sv_estimation_poweron    : False
                               sv_estimation_initialized : False
                               sv_estimation_closedloop : False
                               sv_estimation_moving     : False
```

Figure 7.23

A - These configuration parameters are the base configuration for the lsvsim RAD based app.

B - This parameter lists all the State Variables (SVs) to be simulated. For every SV we must specify a unique name (sv_name) and a script (sv_script) that defines the type of variable to be simulated.

- sv_name: is a string and must be a unique name to identify that SV.
- sv_type: is a string and is the path to the python SV script as python will need to import it.

C - This defines in milliseconds the period in which all the state variable scripts will run.

D - This is a string containing a Yaml configuration (2nd layer yaml config). This configuration will only be used by the SV simulation scripts and we must have a section (map) for each SV listed in the `cfg.sv_list`. Each section name must match the sv_name of each SV.

The configuration of each SV is flexible and depends on the sv_type.

The content of this configuration can be changed at runtime using the `'SetSim'` command like described above.

7.8.3 Other documentation

There is also a design documentation in OneNote for the lsvsim app and coding procedures on how to add state variables to lsvsim processes.

7.8.4 Code Execution Statistics

Code execution statistics are collected in the lsvsim application and also in the python scripts running on top of lsvsim by means of a class `'ExecTimeStats'` (We will refer to this as *Timer*) that will gather information about the execution times. Below the way it works:

- Each *Timer* will collect information about the execution time between 2 points in code. To collect that information we need to use the start and stop methods and place them in the beginning and at the end of the code chunk we need to measure.
    - Timer_start();
    - Timer_stop();

Measured times will be stored in a circular buffer from all the iterations of that code.

- Each *Timer* has a circular buffer that will store the last iterations, when the buffer is full the older data will be overridden. The size is configurable in the instantiation.
- We can have multiple *Timers*, also in python. When the lsvsim process is terminating a table is created with some statistics about all the *Timers* and printed in the console, check below an example:

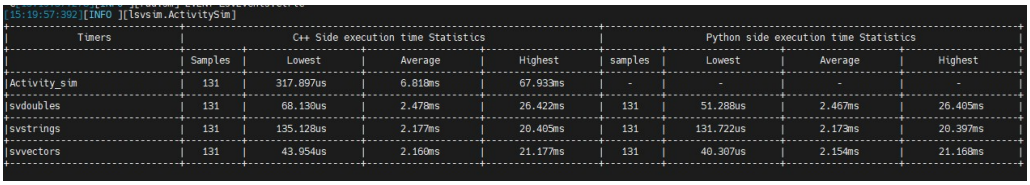


Figure 7.24

In this example we have the following Timers:

- Activity_sim - This measures the time that the 'ActivitySim' iteration takes to complete. It is implemented on the C++ side.
- svdoubles, svstrings and svvectors - Timers implemented on the C++ and python sides. These are independent timers but have the same name, so in the table they will be aggregated in the same line.

  We measure the time the 'Execute' function takes to run, seen from the c++ side when invoking the python method, and also seen from the python method. This way we can measure the overhead from the c++ side when running python.

- The table includes the following fields:
  - Samples–- Is the number of samples in the buffer used to build the statistics data.
  - Lowest–- The lowest measured time.
  - Average–- Simple arithmetic average from all measured time values in the buffer.
  - Highest–- The highest measured time .

For each new SV simulation script added to the lsvsim process:

- Newly added SV simulation scripts will be added to the table automatically on the C++ side;
- If we also want the python side to measure the iteration time the following code needs to be added to the SV Python scripts:
  - __init__ method:
    - Create an instance on the ExecTimeStats class

```
# Instantiate Execution statistics object
# that will measure the time 'Execute' method takes to complete
self.exec_stats = ExecTimeStats(p_sv_name, 1000)
```

  - Execute method:
    - Call the 'timer_start()' before executing the method code and then call 'timer_stop()' just before exiting the method.

```python
def Execute(self, arg, iteration_start_timestamp_ms):

    # Start measuring the execution time
    self.exec_stats.timer_start()

    # Method code goes here
    ...

    # Stop measuring execution time
    self.exec_stats.timer_stop()

    return {}
```

- Terminate method:
  - We need to add the execution statistics data to the shared dictionary with c++ before the current object is destroyed, so add the following code to the end of the 'Terminate' method:

```python
def Terminate(self, arg):

    # Terminate code here
    ...

    # Write execution statistics in the shared object
    if 'exec_stats' not in arg:
        arg['exec_stats'] = {}
    arg['exec_stats'][self.exec_stats.stats_name] = self.exec_stats.get_stats_data()
```

7.8.5 PID Temperature Controller example

This page documents the State Variable example based on an emulation of a temperature controller based on the PID algorithm.

The idea is to have the PID algorithm to reach the predefined temperature on the (emulated) system under control.

Implemented as a consequence of this ticket: [ETCS-1146] lsvsim-- implement PID based state variable example-- ESO JIRA Projects

This SV simulation example is implemented in the HLCC git repository:

lsvsim/simulationscripts/simScripts/src/hlcc/SimScripts/sv_temperature_controller.py · master · ccs / hlcc · GitLab (eso.org)

Figure 7.25 below shows a simplified diagram of the main blocks for the process:

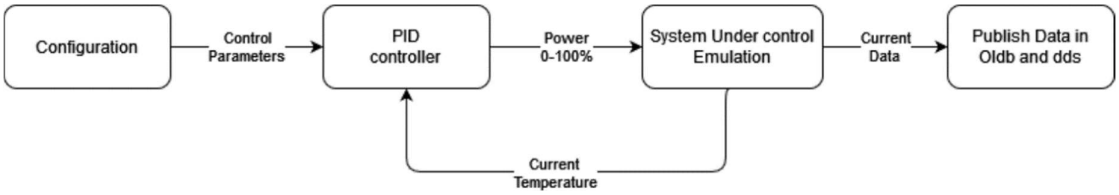


Figure 7.25

- The PID control parameters are stored in the configuration, and can be changed at runtime, if needed, using the command 'SetSim'.
- PID control algorithm to control a temperature of a system. It is implemented in a separate file | https://gitlab.eso.org/ccs/hlcc/-/blob/master/software/lsvsim/simulationscripts/simScripts/src/hlcc/SimScripts/PID.py and is basically a module that can be installed with pip, but in this case, for simplicity, we have added it directly to the repo because it is just 1 file (GitHub—- ivmech/ivPID: Python PID Controller).
- The System under control is a simple emulation of a heating element which is in an environment with a certain temperature and the element itself also has an initial temperature that can be different from the environment. The heating element will receive the power level (0-100%) and will compute the current temperature based on the energy received (power x duration). This emulation will consider things like:
    - power received from the controller
    - inertia, because in a real system the power is not instantly absorbed by the load
    - system losses, due to energy transfer to the environment.
- Publish at every iteration the system data in Oldb and dds.

7.8.5.1 To start the application

To start the simulation run the following command (in this case we run the simulator directly on the command line, without asking Nomad to deploy it, and therefore the simulator will use a predefined port for communication):

```
$ lsvsim -c config/lsvsim/config_m1.yaml
```

7.8.5.2 Sending commands

The SV starts with the PID switched Off and the system under control @25º.

We need to switch On the PID process using the msgsend command:

```
$ msgsend --uri zpb.rr://localhost:11012/SimCmds
::lsvsimif::SimCmds::SetSim '"svtmpctrlr.controller_poweron: True"'
```

To tune the PID parameters use the following commands:

```
$ msgsend --uri zpb.rr://localhost:11012/SimCmds
::lsvsimif::SimCmds::SetSim '"svtmpctrlr.target_temperature_c: 50"'
$ msgsend --uri zpb.rr://localhost:11012/SimCmds
::lsvsimif::SimCmds::SetSim '"svtmpctrlr.pid_kp: 10"'
$ msgsend --uri zpb.rr://localhost:11012/SimCmds
::lsvsimif::SimCmds::SetSim '"svtmpctrlr.pid_ki: 3.5"'
$ msgsend --uri zpb.rr://localhost:11012/SimCmds
::lsvsimif::SimCmds::SetSim '"svtmpctrlr.pid_kd: 0.4"'
```

To subscribe to dds use `telif_subscriber`:

```
$ telif_subscriber --uri dds.ps://1/m1/tmp_ctrlr_sv --entity
ModLsvsimif.Lsvsimif.TmpCtrlSv --verbose --endafter 600
```

The data is also available in the OLDB in this node:

```
$ cii.oldb:///elt/tel/m1/svtempctrlr/
```

The PID controller can be used as a base to implement other examples or real applications.

Notice that when the application `lsvsim` terminates it writes execution time statistics that can be used to evaluate the performance of the application.

7.9 Segexmgr documentation

Segment Exchange Manager process (segexmgr) is based in the lsvsim application and will manage the M1 segment exchange procedure.

Currently this process is still dummy and not performing any useful tasks.

7.9.1 Sending commands

This process supports all the commands implemented in lsvsim application.

7.10 Astronomical site monitor simulator documentation

The astronomical site monitor application (astrositemon) is meant to provide a simulation of site monitor data and will in the future evolve into the application that interfaces with the ASM hardware to bring data to the telescope and make it available to instruments through the interfaces defined in the standard CCS-INS ICD.

It is based on the lsvsim application and is currentlty providing the following features:

- Publish in Oldb a set of values defined in configuration:
  - uri: cii.oldb:///elt/cs/telif/site/air_temperature
  - uri: cii.oldb:///elt/cs/telif/site/air_temperature_lapse_rate
  - uri: cii.oldb:///elt/cs/telif/site/ambient_pressure
  - uri: cii.oldb:///elt/cs/telif/site/relative_humidity
  - uri: cii.oldb:///elt/cs/telif/site/seeing
  - uri: cii.oldb:///elt/cs/telif/site/wind_direction
  - uri: cii.oldb:///elt/cs/telif/site/wind_speed
- Compute the current moon data using the astropy lib and publish in Oldb:
  - uri: cii.oldb:///elt/cs/telif/site/moon/altaz
  - uri: cii.oldb:///elt/cs/telif/site/moon/phase
  - uri: cii.oldb:///elt/cs/telif/site/moon/radec
  - uri: cii.oldb:///elt/cs/telif/site/moon/target_distance

7.10.1 Sending commands

This process supports all the commands implemented in lsvsim application.

7.10.2 Changing parameters

It is possible to change at runtime parameters that are defined in the configuration to create simulations with changing values.

Check below the example command used to change the air temperature to 120.

On a system running with Consul, we shall ask Consul for the service uri:

```
$>  msgsend  --uri=zpb.rr://astrositemon/SimCmds  ::lsvsimif::SimCmds::SetSim
'"astrositemon.site_dps.air_temperature.value: 120"'
```

Below the list of all parameters that can be updated:

- astrositemon.site_dps.air_temperature.value
- astrositemon.site_dps.air_temperature_lapse_rate.value
- astrositemon.site_dps.ambient_pressure.value
- astrositemon.site_dps.relative_humidity.value
- astrositemon.site_dps.seeing.value
- astrositemon.site_dps.wind_direction.value
- astrositemon.site_dps.wind_speed.value

The following example applies if we are running the application manually:

```
$> lsvsim -c config/lsvsim/config_astrositemon.yaml

$> msgsend --uri zpb.rr://localhost:11005/SimCmds ::lsvsimif::SimCmds::SetSim
'"astrositemon.site_dps.air_temperature.value: 120"'
```

7.11 Utilities

This section documents a set of utilities/clients used for testing and as examples.

Some of these utilities are part of hlcc, other are generla utilities available from ECOS projects.

Some hlcc utilities have a –test option that is used by integration tests and is meant to work in the dev deployment.

7.11.1 $ consulGetUri --help

usage: consulGetUri [-h] [--version] [-v VERBOSE] [-c CONSUL_HOST] [-p CONSUL_PORT] [-s SERVICE] [-i IFACE] -r {server,ipport,uri,ifuri}

Queries consul for information on a requested service. The command line arguments allow to select the information to be returned and the output format.

optional arguments:

-h, --help show this help message and exit

--version show program's version number and exit

-v VERBOSE, --verbose VERBOSE

Verbose mode: 0=WARNING, 1=INFO, >1=DEBUG

-c CONSUL_HOST, --chost CONSUL_HOST

Host where consul is running. Default: None (will parse $CONSUL_ADDR env var or localhost if undefined)

-p CONSUL_PORT, --cport CONSUL_PORT

Port used by consul. Default: None (will parse $CONSUL_ADD env var or use 8500 if undefined)

-s SERVICE, --service SERVICE

Service for which we are querying consul (requires -r port/uri/ifuri)

-i IFACE, --interface IFACE

Inteface for which we are querying consul (requires -r ifuri)

-r {server,ipport,uri,ifuri}, --request {server,ipport,uri,ifuri}
The requested uri information: server={server ip},
ipport={server ip}:{port}, uri=zpb.rr://{serverIp}:{port}, ifuri=full uri for
the requested interface.
Default = ifuri

7.11.2 $ msgsendSub --help

usage: msgsendSub [-h] [-v] [--version] [-t SECONDS] [-n COUNT] [-l] [-T SECONDS] -u URI [--no-ccs-defaults] [--malprops JSON] FQN

Receive structs via CII MAL Pub-Sub

positional arguments:
FQN fully qualified name of struct type (e.g. ::pkg::struct)

options:
-h, --help show this help message and exit
-v, --verbose show more log output; repeat for more
--version show program's version number and exit
-t SECONDS terminate after so many seconds
-n COUNT terminate after so many received structs
-l display results in long-format (tab-separated)
-T SECONDS the subscriber timeout in seconds (default: 1.0)
-u URI, --uri URI the URI of the CII MAL Topic, e.g. dds.ps:///YOUR_TOPIC
--no-ccs-defaults do not load CCS default MAL properties even if they are available
--malprops JSON mal-specific properties as JSON object (string->string), e.g. '{"zpb.ps.slowJoinerDelayMs": "500"}'

Notes:
1. When this tool's output gets piped (to grep, head, ...), it will
try to react gracefully to "closed pipe". Due to shortcomings in
python this is not guaranteed to work. Thus, always additionally
use -t, -n, or both.

Examples:

msgsendSub ::stdif::State -u zpb.ps://127.0.0.1:65001/MSGSEND

msgsendSub ::stdif::State -u dds.ps:///MSGSEND

msgsendSub ::stdif::State -u dds.ps:///MSGSEND -t10 | jq '.["state"]'

CCS default MAL properties can be overridden by extra mal-specific properties (--malprops).

Examples:

msgsendSub ... --malprops '{"dds.qos.profile.name.participant": "Localhost_Only"}'

msgsendSub ... --malprops '{"dds.qos.profile.library": "/tmp/dds_profile.xml"}'

msgsendSub ... --malprops '{"dds.qos.participant.interfaceWhiteList": "cnd lo"}'

For a table describing the data published by HLCC, see section 6.4.3

Here a couple of specific examples for hlcc:

```
# Assuming hlcc is tracking and publishing, we should get entities on domain 2
msgsendSub -u dds.ps://2/TELIF_CCS_TRG_OBS_ALTAZ -t 5 ::ccsinsif::AltAz
msgsendSub -u dds.ps://2/TELIF_STATUS -t 10 ::stdif::Status
```

7.11.3 $ msgsendPub --help

usage: msgsendPub [-h] [-v] [--version] [-t SECONDS] [-T SECONDS] -u URI [--no-ccs-defaults] [--malprops JSON] FQN STRUCT [STRUCT ...]

Send one or more structs via CII MAL Pub-Sub

positional arguments:

FQN	fully qualified name of struct type (e.g. ::pkg::struct)
STRUCT	struct content, e.g. '{"a":1, "b":2.0}'

options:

-h, --help	show this help message and exit
-v, --verbose	show more log output; repeat for more
--version	show program's version number and exit

-t SECONDS interval between publishing STRUCTs (default: 2.0)

-T SECONDS the publisher timeout in seconds (default: 1.0)

-u URI, --uri URI the URI of the CII MAL Topic, e.g. dds.ps:///YOUR_TOPIC

--no-ccs-defaults do not load CCS default MAL properties even if they are available

--malprops JSON mal-specific properties as JSON object (string->string), e.g. '{"zpb.ps.slowJoinerDelayMs": "500"}'

Examples:

msgsendPub ::stdif::State -u zpb.ps://127.0.0.1:65001/MSGSEND '{"source": "A", "state": "idle"}'

msgsendPub ::stdif::State -u dds.ps:///MSGSEND '{"source": "A", "state": "idle"}' '{"source": "B", "state": "err"}'

CCS default MAL properties can be overridden by extra mal-specific properties (--malprops).

Examples:

msgsendPub ... --malprops '{"dds.qos.profile.name.participant": "Localhost_Only"}'

msgsendPub ... --malprops '{"dds.qos.profile.library": "/tmp/dds_profile.xml"}'

msgsendPub ... --malprops '{"dds.qos.participant.interfaceWhiteList": "cnd lo"}'

Here an example specific for hlcc:

```
msgsendPub -u dds.ps://2/TELIF_CCS_TRG_OBS_ALTAZ ::ccsinsif::AltAz '{"alt":1.0, "az":9.0}' '{"alt":2.0, "az": 9.0}' '{"alt":3.0, "az": 9.0}'
```

7.11.4 pkp_llnetio_subscriber --help

Subscribe to llnetio – rtms to receive pointing kernel positions data.

Allowed options:

-h [--help] Show help message

-p [--port] arg (=10001) Listen UDP Port

-c [--count] arg (=1) Data to read count

7.11.5 test utilities

If hlcc is installed in INTROOT from sources, uncluding the test sub-project, a few utilities normally used only in integration tests are installed.

They might be useful also to perform other tests, event though their functionality is demonstrated in Jupyter Notebooks and we think it is easier to achieve the functionality they provide by looking at the Notebooks.

Where present, the –test option is used in integration tests to exercise the functionality.

There are:

- telif_metadaqcmds
- telif_publisher
- telif_sky_offset

7.12 Jupyter notebooks

The code repository includes a set of jupyter notebooks with examples written in python.

There are two frontends to execute jupyter notebooks (see Project Jupyter | Home):

- Jupyter Notebook
- JupyterLab (not available in Fedora 38, See: [ELTDEV-1090] Package jupyterlab - ESO JIRA Projects)

It is also possible to run Jupytes Notebooks directly on the command line, without a separate server, as described in 7.12.4

The notebooks are in this folder in the GitLab repository:

- software/jupyter_notebooks · master · ccs / hlcc · GitLab (eso.org)

If you select any of the notebooks in GitLab, the viewer will properly format the contents, providing you with well readable examples.

The notebooks are installed by waf in:

```
$PREFIX/jupyter_notebooks
```

and in the RPM distribution are in:

```
/elt/hlcc/jupyter_notebooks
```

Existing jupyter notebooks:

Name	Description	Intergrational test

BasicInteractionExample	Basic interaction in python with a client	
ChangeTelescopeSite	Change the telescope site, to Paranal or to Armazones, for example	
EltpkPreset	Execute preset command directly to eltpk. Preset the telescope to ra and dec position in radians, according to ESO coordinate system	x
GuideStarHelper	Read and handle the Guide star information that is being published in the Oldb, like target guide stats	
LoadSetConfig	Examples of usage of the LoadConfig and SetConfig commands	
LoggingConfigExample	CII Logging configuration in python	
Metadaq_Commands	Send metdaq (Data Acquisition Commands) to the telif	
MsReadDdsData	Collect one sample of data from each of the MS DDS data publishers	x
OffsetRaDec	Examples of sending an Offset command	x
OffsetTests	Examples of sending Offset commands some real or potential issues.	x
PresetPolarisAustralis	Preset to Polaris Australis (sigma Octantis)	x
PresetPolarisAustralisWithGS	Preset to Polaris Australis (sigma Octantis) And sets 3 Guide Probes to Guide Stars	
PresetSequencerSetup	Check the status and configuration of the Sequencer used to manage Preset commands	
PresetTests	Examples of sending Preset commands and explores some real or potentialissues	x
PresetToSkyAtAltAz	Preset the telescope to sky coordinates corresponding to the given (alt, az) position in degrees, according to ESO coordinate system	x
PresetToSkyAtRaDec	Preset the telescope to sky coordinates corresponding to the given (ra, dec) position.	x

PubSubCollectOneForEach	Collect samples of data from each of the HLCC publishers (both dds nondeterministic and mudpi deterministic)	
RaDecSimpleCalculator	Simple calculator to get the (Ra,Dec) coordinates for a given (alt, az) at the specified location (Armazones, by default) and time (now, by default)	
StartupShutdown	Startup/shutdown and do some control of the simulator processes.	
StellariumApiTests	Examples with the basic functions and interaction between HLCC features and Stellarium	
StellariumConfig	Configures stellarium to be used with HLCC by using the rest API to control the application	
StellariumFollowsTelescope	Stellarium is first initialized to display the sky on Armazones. Then loop is running and you preset the hlcc simulator telescope to a position in the sky or to a fixed position in (alt, az) the view of stellarium will move to follow the pointing of the telescope.	
StellariumPointTelescopeToSelected	Examples to preset the HLCC to the object selected in stellarium	
TelemetryDataCollectDDS	Collect nondeterministic data published by the telescope simulator on the pub/sub channels.	x
TelemetryDataCollectMUDPI	Collect deterministic data published by the telescope simulator as MUDPI packets.	x
TelifGetConfig	Get configuration from telif	x
TelifPreset	Example to preset telescope using telif	x
TelifRequestReleaseControl	Requesting and Releasing Control. These commands are being sent to trexsup process	x
TelifRous	Executes Rous commands	x
AIV COMM UseCases		
MountPointingModelGeneration	The pointing model for the ELT is divided into two components: mount and optics, as per the Nighttime operations scenario	

	https://pdm.eso.org/kronodoc/HQ/ESO-324987 This note addresses the mount pointing model.	

7.12.1 Start jupyter server and browser client on the same machine with one command

In order to actually execute the code in the notebooks on a machine with the system installed:

```
cd <hlcc repository folder/software>   # go where your git repository for
                                       # hlcc has been extracted
                                       # or where it has been installed
jupyter-notebook
        or:
jupyter-lab
```

A jupyter server will be started and the Firefox browser will open on the folder containing the code:

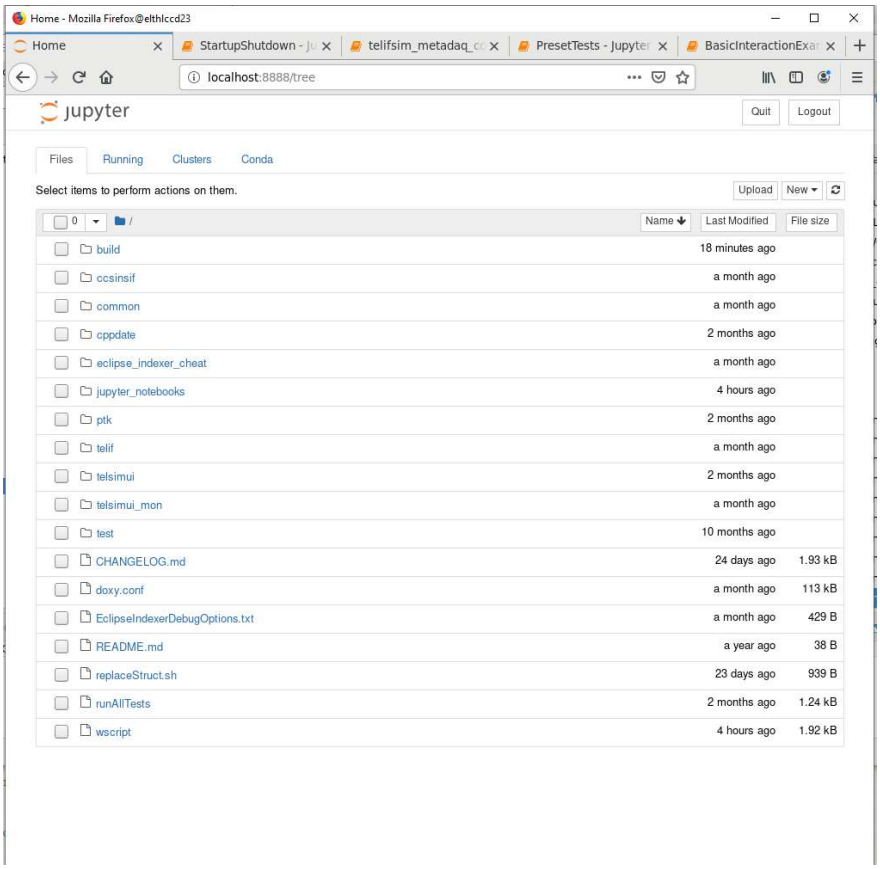


Figure 7.26

7.12.2 Start jupyter server and browser client on different machines

It is also possible to run server and browser client in different machines.

On the machine where you want to run the server, cd to the root folder for your notebooks and issue the command

```
cd <hlcc repository folder/software>   # go where your git repository for hlcc
                                       # has been extracted
                                       # or where it has been installed


jupyter-notebook --no-browser --ip=0.0.0.0
```

or:

```
jupyter-lab --no-browser --ip=0.0.0.0
```

This start just the jupyter-server. The console where the server runs will tell how to attach the client:

```
[I 11:55:48.397 LabApp] The Jupyter Notebook is running at:
[I 11:55:48.397 LabApp]
http://elthlccd23:8888/?token=019b2a3cc9f8610b88d7d6b40976844fea310cbc4469ae34
[I 11:55:48.397 LabApp]  or
http://127.0.0.1:8888/?token=019b2a3cc9f8610b88d7d6b40976844fea310cbc4469ae34
[I 11:55:48.397 LabApp] Use Control-C to stop this server and shut down all kernels (twice
to skip confirmation).
[C 11:55:48.401 LabApp]

To access the notebook, open this file in a browser:
        file:///home/eltdev/.local/share/jupyter/runtime/nbserver-394842-open.html
    Or copy and paste one of these URLs:
        http://elthlccd23:8888/?token=019b2a3cc9f8610b88d7d6b40976844fea310cbc4469ae34
     or http://127.0.0.1:8888/?token=019b2a3cc9f8610b88d7d6b40976844fea310cbc4469ae34

To connect from any browser, just open the browser and and go to the URL with the given
token. In this case:

http://elthlccd23:8888/?token=019b2a3cc9f8610b88d7d6b40976844fea310cbc4469ae34
```

You can for example run the server on the machine where the hlcc is running and the client on a Microsoft Windows desktop with Microsoft Edge browser instead of Firefox.

In some cases this would be much more responsive.

7.12.3 Executing Jupyter Notebooks in the web browser

You can select the `jupyter_notebooks` folder and open in the browser any of the noteboooks stored there.

For example:

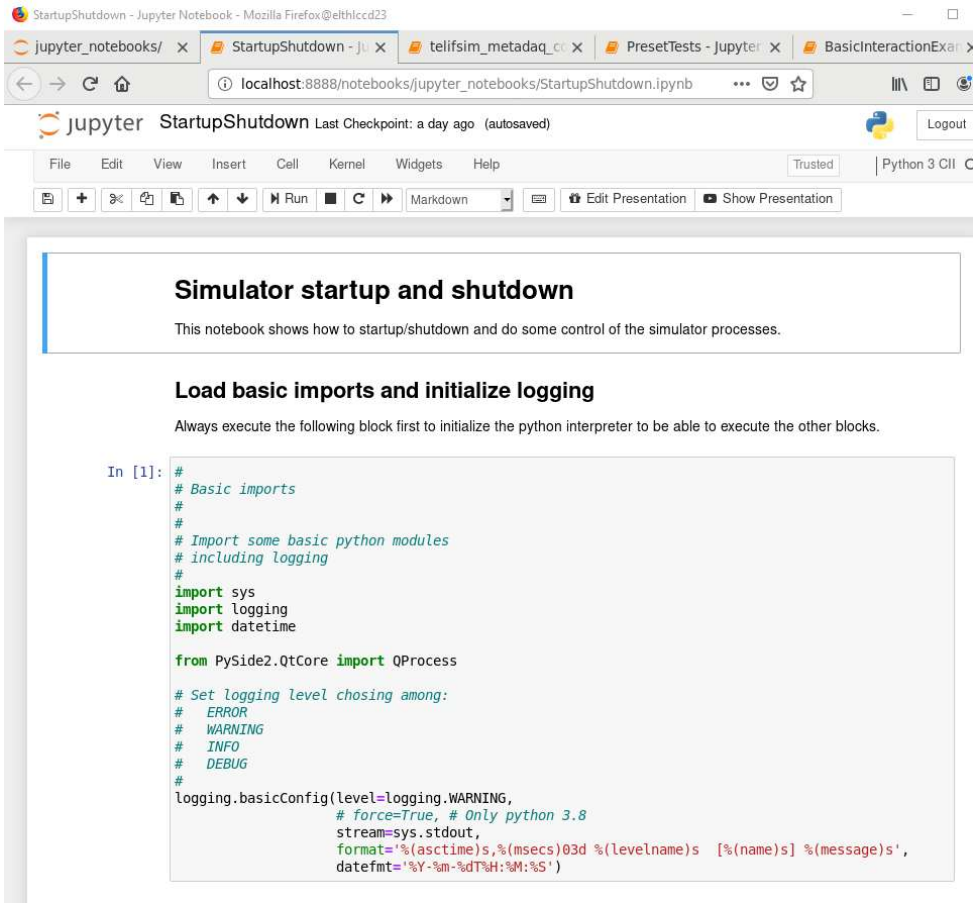


Figure 7.27

You can then run any single cell by:

- Selecting the cell you want to run
- Clicking the **Run** button

Read carefully the documentation of the notebooks.

Some cells are meant to setup the python kernel environment and shall be executed in any case before the other cells with specific actions can be executed.

A major advantage of using these notebooks is that you can (almost always) select any cell and execute them in any order or changing just some values. This makes running interactive tests and learning how the system behaves very easy.

The code can then be copied in your own scripts.

7.12.4 Executing Jupyter Notebooks on the command line

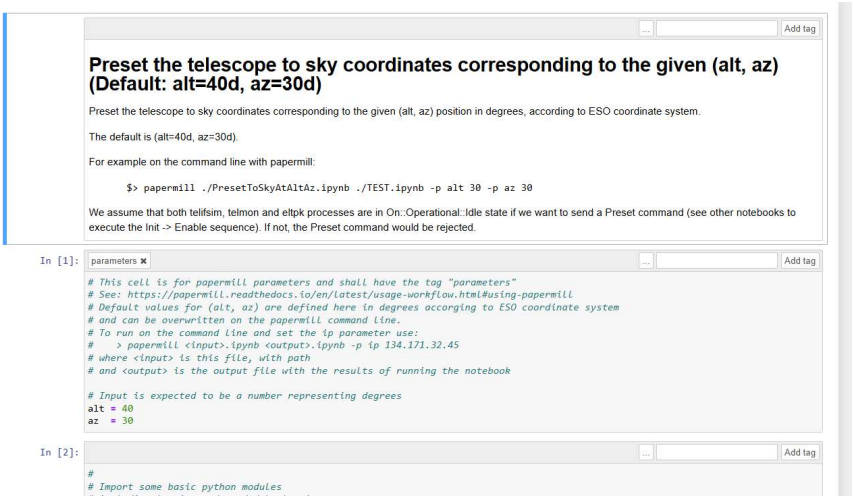
You can use the **papermill** tool to run notebooks directly on the command line, passing parameters when needed.

Typical command lines are:

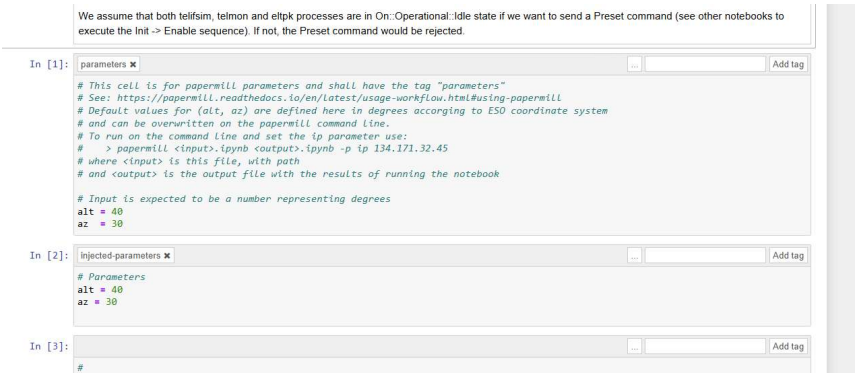
```
> papermill ./PresetToSkyAtAltAz.ipynb ./TEST.ipynb -p alt 40 -p az 30
> papermill ./PresetPolarisAustralisWithGS.ipynb ~/temp/PresetPolarisAustralisWithGS.ipynb
```

- The first parameter is the notebook to be executed
- The second parameter is the output notebook, that will be a copy of the notebook executed but including also all the results for each cell executed
- The -p option is repeated for each input parameter, followed by the name of the parameter and the corresponding value

  Parameters are defined in a cell with the `parameters` tag (tags are a jupiter feature and there is a plugin also for Visual Studio Code) containing the definitions for all parameters:



  Replaced parametrs are inserted in a cell with the tag `injected-parameters` following the `parameters` cell and can be inspected in the output notebook:



Using papermill to run notebooks allows to execute the full notebook in a single command line without having to open the notebook in the browser. Using parameters also allows to run automatic tests and generate automatic reports.

7.12.5 Running notebooks through the Generic Jupyter Notebook Runner GUI

The hlccJupyterRunnerGui is a convenience GUI to run the Jupyter Notebooks.

The Notebooks under INTROOT (or CFGPATH like locations) are listed via the combobox in the left side of the GUI.

The parameters extracted from the Notebook (using the conventions of papermill with a parameters cell), are displayed at the bottom right section for editing.

The documentation of the selected notebook (i.e. the contents of the first cell in the notebook) is displayed at the top right section.

The Notebook can be started using the RUN button.

At the start of the Notebook the output, of the Notebook is displayed simultaneously, in the area under the documentation.

The Notebook can be aborted using the ABORT button. The status of the execution of the Notebook is displayed above the Run button.

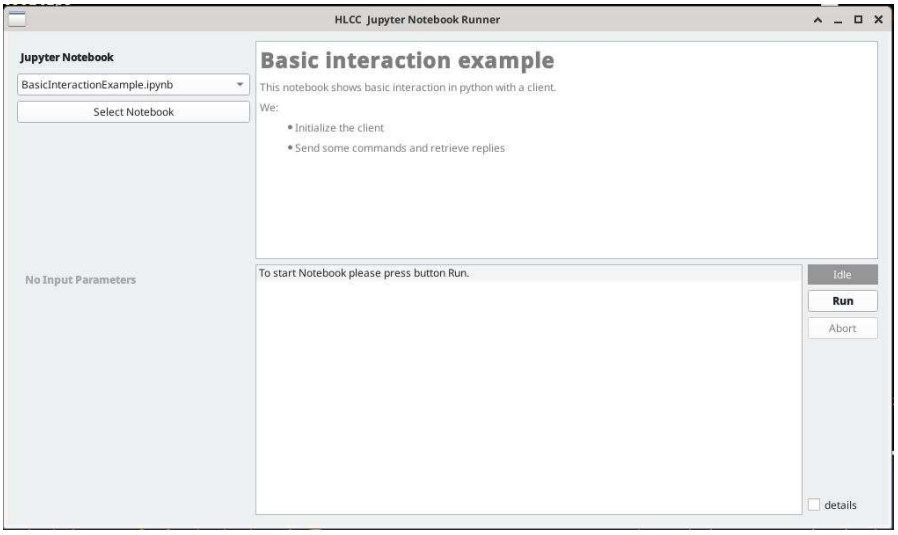


Figure 7.28 This example shows a Notebook without input parameters. This Notebook has not yet been started its status is IDLE and the output area is empty.

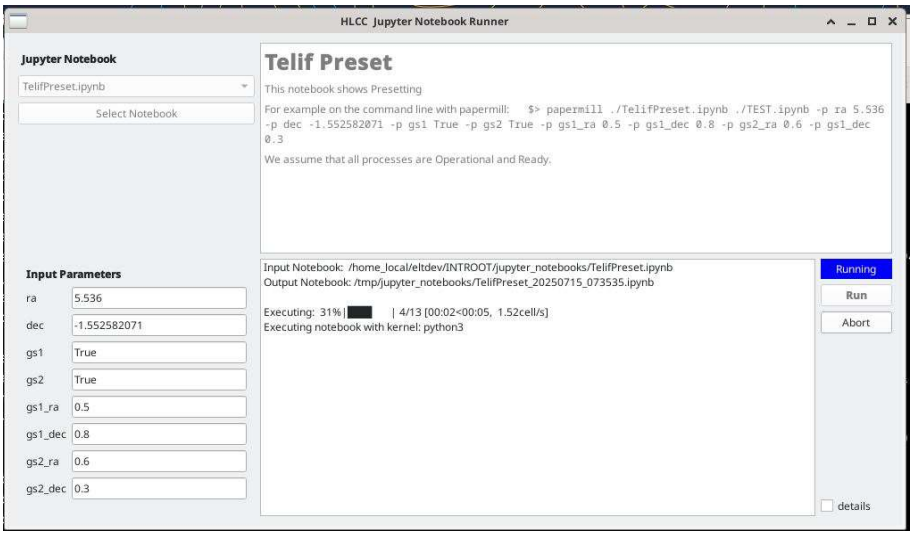


Figure 7.29 This example shows a running Notebook with parameters.

This gui runs the Notebooks via papermill, with the optional detailed output.

The executed notebook is indicated at the first line of the output area.

The produced Notebook output file (with the selected parameters) file is stored at the location shown by the second line.

7.13 Sequencer scripts and OBs

The code repository includes a set of ELT Sequencer scripts and OBs:

The Sequencer scriptsare in this folder:

https://gitlab.eso.org/ccs/hlcc/-/tree/master/software/seq/src/hlccseq

and are installed as the `hlccseq` python module.

The OBs are in the folder:

https://gitlab.eso.org/ccs/hlcc/-/tree/master/software/seq/resource/config/seq

The scripts are python programs written using the sequencer libraries and can be executed in the seq server using the seg gui or manually from the command line.

It is possible to execute an individual sequence or an OB on the command line or using the sequencer server and the sequencer user interface.

7.13.1 Run an individual sequence on the command line

In order to run an individual sequence on the command line use the **`seqtool run`** command like:

```
> seqtool run hlccseq.simStartup
```

7.13.2 Run through sequencer server and GUI

In order to use the scripts you can start an independent the sequencer server, if not already running:

```
> seqtool server
```

In order to use the scripts, you can then start the sequencer graphical user interface

```
> seqtool gui
```

If no default sequencer server is running, the GUI will automatically start one.

You can start an independent the sequencer server, if not already running:

```
> seqtool server
```

A sequencer gui would then directly connect to this server.

HLCC deployment includes a sequencer server that is used for handling typically preset commands. You can open a sequencer GUI connected to this server by asking Consul for its uir, using the following command:

```
> seqtool gui --address `consulGetUri -s seqserver -r ipport`
```

Once the gui is started, there are two ways to load and run sequences from the sequencer gui:

1. Use the Application -> Load Script menu with the file chooser.
   In this way you have to locate the sequencer script source
2. Load them in the sequencer server from a python sequences library available in the PYTHON_PATH:
   - activate the debug mode from the Application->Debug mode menu
   - issue in the Console tab the commands to load specific sequences, like

     load hlccseq.simStartup

     load hlccseq.simShutdown

The sequencer will look for the sequencer scripts in the INTROOT or in the system folders and load them from there.

See the following screenshot:

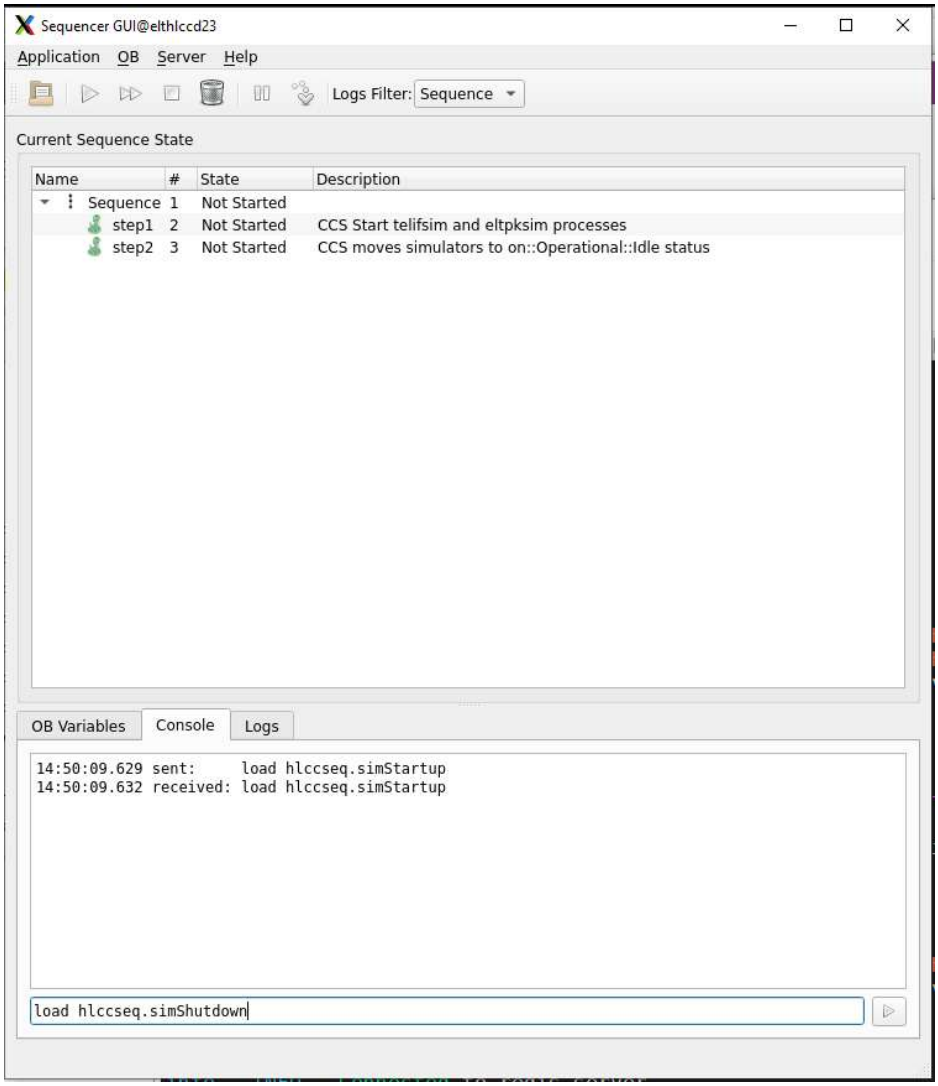


Figure 7.30

For more details on sequences and OBs, look at the Sequencer documentation in ELT Sequencer scripts

7.13.3 Known Issues

- The seqtool gui sometimes does not show a loaded sequence/OB.

In this case, restart the tool or reload the sequence.

See: [EICSSW-1879] seqtool gui sometimes does not show a loaded sequence - ESO JIRA Projects

7.13.4 ToDo

None listed here at the moment

7.14 Change Telescope site coordinates

By default the telescope is configured to be located in Armazones, at the actual ELT coordinates.

For testing purposes, it might be useful to "move the telescope" in a different location, for example the locations of UT1 in Paranal.

The change can be done

1. in the configuration deployment files or
2. dynamically sending a configure command

The configuration for the site location is under responsibility of the eltpk process, that is also responsible to make it available in the CCS-INS interface in the OLDB.

While the system is running, you can find the parameters for the site location in these OLDB branches:

cii.oldb:///elt/cs/telif/site/info (public CCS-INS interface)

cii.oldb:///elt/cs/hlcc/eltpk/cfg/site/info/elevation (eltpk private OLDB)

7.14.1 Change telescope location in configuration files

- The configuration is stored in the file https://gitlab.eso.org/ccs/hlcc/-/blob/master/software/telif/eltpk/app-config/src/config.yaml.in or in any of the corresponding deployment configuration files
- Edit the keys:

  cfg.site.info.elevation : 3046.0 # meters

  cfg.site.info.latitude : -0.429164 # Radians

  cfg.site.info.longitude : 1.225075 # Radians

  cfg.site.info.id : "ELT"

- Start the system with the new configuration

7.14.2 Change telescope location with configuration commands

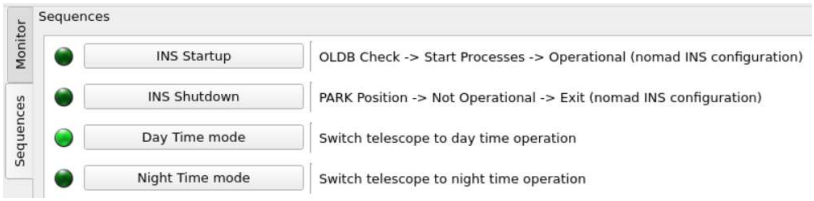
- The procedure is documented in details in the Jupyter notebook https://gitlab.eso.org/ccs/hlcc/-/blob/master/software/jupyter_notebooks/ChangeTelescopeSite.ipynb
- In summary:
  - The site location configuration keywords can be changed only when the `eltpk` process is in `On::NotOperational` state, therefore send a Disable command if it is `On::Operational`.
  - Prepare a file with the new site location keywords and send the LoadConfig command

    Or
  - Prepare a string with the same keywords and send the SetConfig command

7.15 Change Telescope operation mode

The normal daily cycle of the telescope is composed for 2 different operation modes: the daytime mode and the nighttime mode. The first will be active during the day for maintenance operations the second will be active during the night for astronomical observations.

To switch from one to the other we have implemented the skeleton for two sequences that will help in those procedures.



7.15.1 DayTime sequence

This sequence must be executed by the sequencer Gui because it will make use of dialogs.

To start the sequencer, with the preloaded sequence, press the “Day Time mode” button on the sequences panel of the telescope simulator UI (check picture above).

Note: Due to a potential problem in the sequencer sometimes the sequencer Gui does not come pre-loaded with the sequence, in that case close the sequencer Gui and try again.

The main purpose of this sequence is to:

- Move the telescope into segment exchange position.

- Disable the telescope axes and enable the processes used in the daytime mode.

7.15.2 NightTime sequence

To start the sequencer, with the preloaded sequence, press the “Night Time mode” button on the sequences panel of the telescope simulator UI (check picture above).

Note: Due to a potential problem in the sequencer sometimes the sequencer Gui do not come pre-loaded with the sequence, in that case close the sequencer Gui and try again.

The main purpose of this sequence is to:

- Disable all the processes used only in day time mode and set them to be ignored in the ccs state estimation.
- Enable all processes needed in Night mode to make observations.

7.15.3 Extending Daytime and NightTime sequences

If we need to add other processes to be managed by the daytime and nighttime sequences check below what needs to be done:

In the sequence file ‘hlccDayTime.py’:

- add the new process details to the list ‘self.hlcc_proc_list’;
- list that process in the lists: ‘self.daytime_not_operational’ or ‘daytime_operational’ according to what is the function of that process.

In the sequence file ‘hlccNightTime.py’:

- add the new process details to the list ‘self.hlcc_proc_list’;
- list that process in the lists: ‘self.nighttime_not_operational’ or ‘self.nighttime_operational’ according to what is the function of that process.

8. Known issues

Look in each section for specific issues and important ToDos.

This page list some general issues you might encounter and we know of.

- 20241008 – trex v0.1.1

If the trexup application does not start, there might be a problem with the configuration of the shared memory.

If you cannot start it by end and get this error:

```
> trexsup -c config/trexsup/config.yaml -lTRACE

[12:56:11:274][ERROR][trexsup] Dynamic exception type:
boost::interprocess::interprocess_exception
std::exception::what: Permission denied
```

Cleaning up the trex shared memory should fix the problem:

```
> rm -f /dev/shm/*
```

The issue is related to permissions when the process is started sometimes by nomad and sometimes manually.

The issue will be fixed in an upcoming version of trex.

9. Other HLCC applications

This section contains information on other HLCC applications not directly involved in the Telescope Simulator.

The section will be refactored and extended as the real system applications will be developed, to cover the functionality provided by the real system and specific needs of other deployments.

10. Stellarium: Installation and configuration

Stellarium is a free GPL software (Source code is available in GitHub: Stellarium GitHub) which renders realistic skies in real time with OpenGL. It is available for Linux/Unix, Windows and macOS. With Stellarium, you really see what you can see with your eyes, binoculars or a small telescope.

It is possible to integrate Stellarium with the HLCC Telescope Simulator, so that Stellarium view is centered at the place where the telescope is pointing and also to select objects in Stellarium and send the telescope pointing and tracking to that objects.

At the moment HLCC provides Jupyter Notebooks that demonstrate such integration and a UI more integration features are foreseen as future developments.

10.1 Installation

Stellarium can be installed on any machine accessible on the network from the HLCC machines.

For better performance it is advisable to install stellarium on the machine where the main operator display is running, for example on the MS Windows machine that is used as the main console.

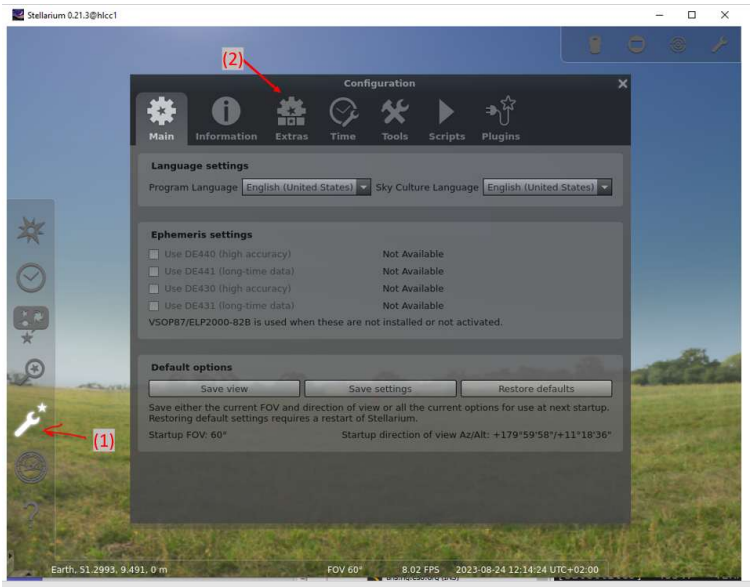
- Links to the packages for Linux, MS Windows and macOS are here: https://stellarium.org/
- Installation on ELT Fedora linux machines can be done directly as root with the dnf command
  - `dnf install stellarium`
- If you experience problems running stellarium in a Linux (virtual) machine, the reason might be the graphic capabilities of your machine. Try to run in log-graphic mode to see if this solves the problem (but you will pay with lower performance):
  - `stellarium --low-graphics`

10.2 Configuration

In order to optimally use Stellarium with HLCC it is necessary to configure it.

Most configurations can be also done programmatically and it is foreseen to implement applications to take care of the configuration, but for the time being what follow are the most important configuration steps (The numbers in parenthesis correspond to numbers in the screenshots).

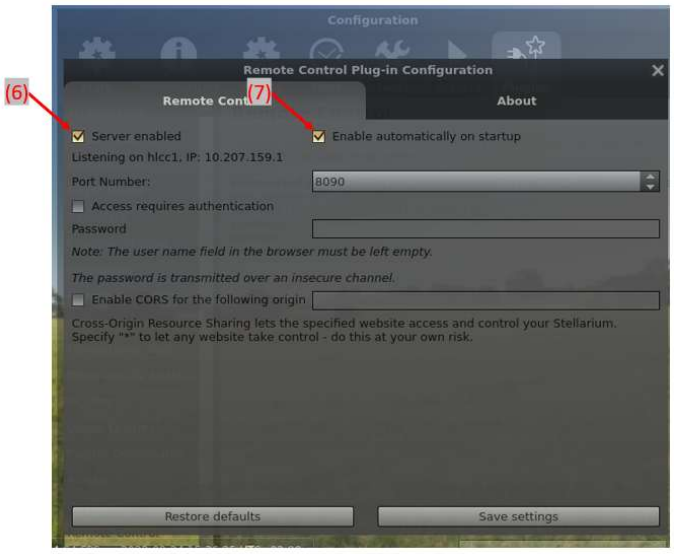
- Start Stellarium
- To exit from the default full-screen view press F11
- Open the cascade command bar on the left, by getting near to the left frame border
- Selet Configuration (1) to open the configuration panel
- Select Extras (2)



- Load all star catalogues (3)
  to get a more realistic view of what would be visible from the ELT.
  With all catalogues it is possible to see stars up to magnitude 17.

  - Select Plugins (4)

- Remote control plugin (plugin managing the web server with rest API)



  - Load at startup (5)
  - restart Stellarium if not already selected
  - Configure -> Server enabled (6), enable automatically at startup (7)

- Oculars

  - Load at startup (8)
  - restart Stellarium if not already selected
- Sky and viewing options (10)

- Landscape
  - Download the Paranal landscape (we do not have a landscape for Armazones yet) from this SharePoint link: Paranal landscape
  - Add/remove landscapes -> add the downloaded landscape (11)

- Select Paranal and use as default (12)

10.3 Running the Jupyter notebooks

At this point it is possible to run the Jupyter Notebooks.

The more convenient way is just to use the Stellarium HLCC User Interface. See: 11.

All Notebooks provided by HLCC for usage with Stellarium are named

```
Stellarium*.ipynb
```

Read carefully the documentation inside the notebooks and first of all set the URI with the proper hostname or IP to access the Stellarium server.

The initialization cells will take care of setting some additional configuration parameters, like the telescope location coordinates.

To complete the configuration of stellarium, run first of all the notebook

```
StellariumConfig.ipynb
```

Two other very useful notebooks for usage with stellarium are:

```
StellariumFollowsTelescope.ipynb
```

Keeps the view of stellarium aligned with the line of sight of the hlcc telescope by reading in a loop the current (alt,az) and moving the display of the sky to that position

```
StellariumPointTelescopeToSelected.ipynb
```

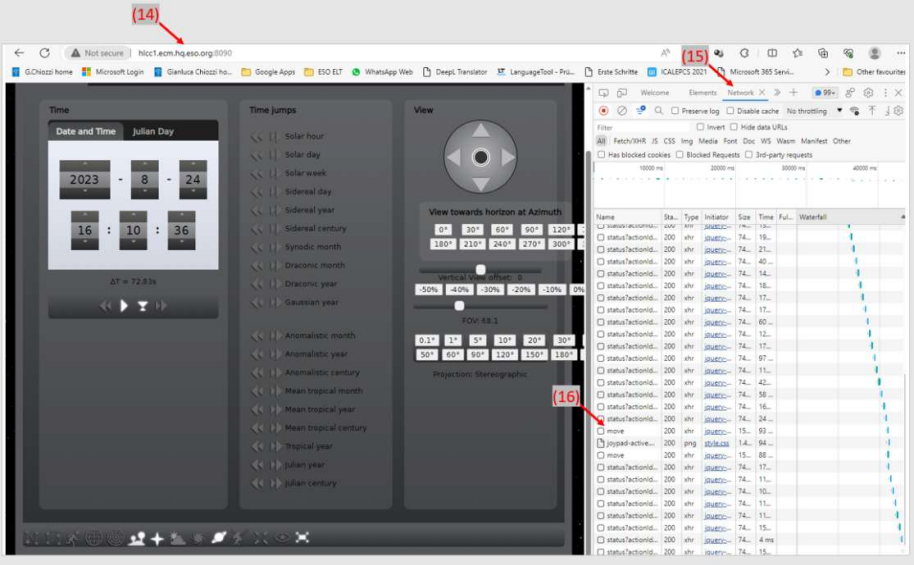
Running the notebook once an object is selected in the ui of stellarium points the telescope to that object.

ToDo: *notice that for the time being location coordinates need to be set every time the application is started, since stellarium cannot set a default position to locations not contained in the database of named locations.*

10.4 Learning and debugging

A very good way to learn the Stellarium Remote Control API and to debug it, is

- open the Remote Cntrol Web API, accessible on port 8090 on the host where Stellarium is running with the web browser (here below the commands with Edge, other browsers have similar options)(14)
- right click and select "inspect" to open the debugger
- select network (15)
- watch header and payload: (16)



10.5 To know more

Here some llinks to know more about Stellarium,application, scripting and rest API:

- Stellarium Astronomy Software home page
- Stellarium/stellarium: Stellarium sources (github.com)
- Scripting Stellarium – planetmaker.de

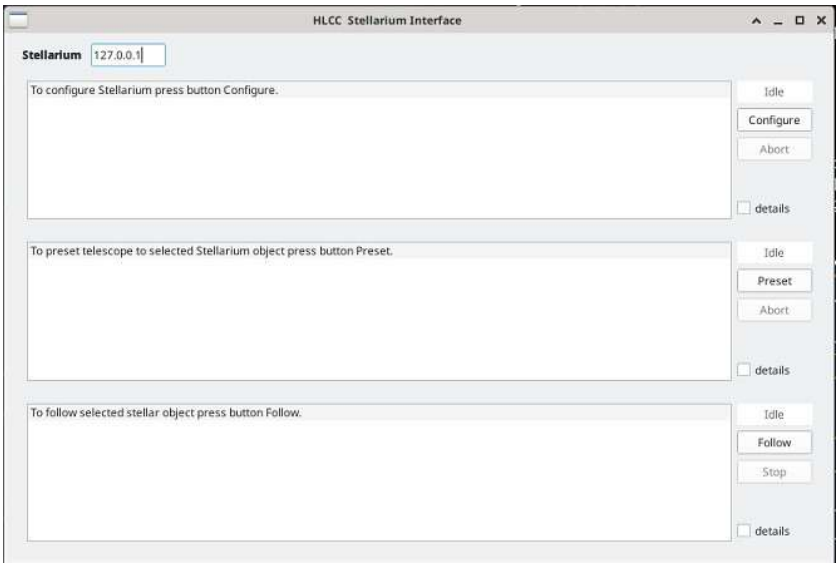
- How to use Stellarium's Oculars plugin to match your optics (howtoforge.com)
- AstroQuest1: How I Set Up a Custom Landscape in Stellarium & Why!
- Stellarium: RemoteControl plugin HTTP API description
- Stellarium: StelMainScriptAPI Class Reference

11. HLCC Stellarium User Interface

The hlccStellariumGui provides graphical interface, which allows the user

- to configure the Stellarium application
- to preset the telescope to the object the user selected via Stellarium, and
- to command the telescope to follow the selected object.

This essentially executes the Stellarium Jupyter notebooks described in the previous chapter.

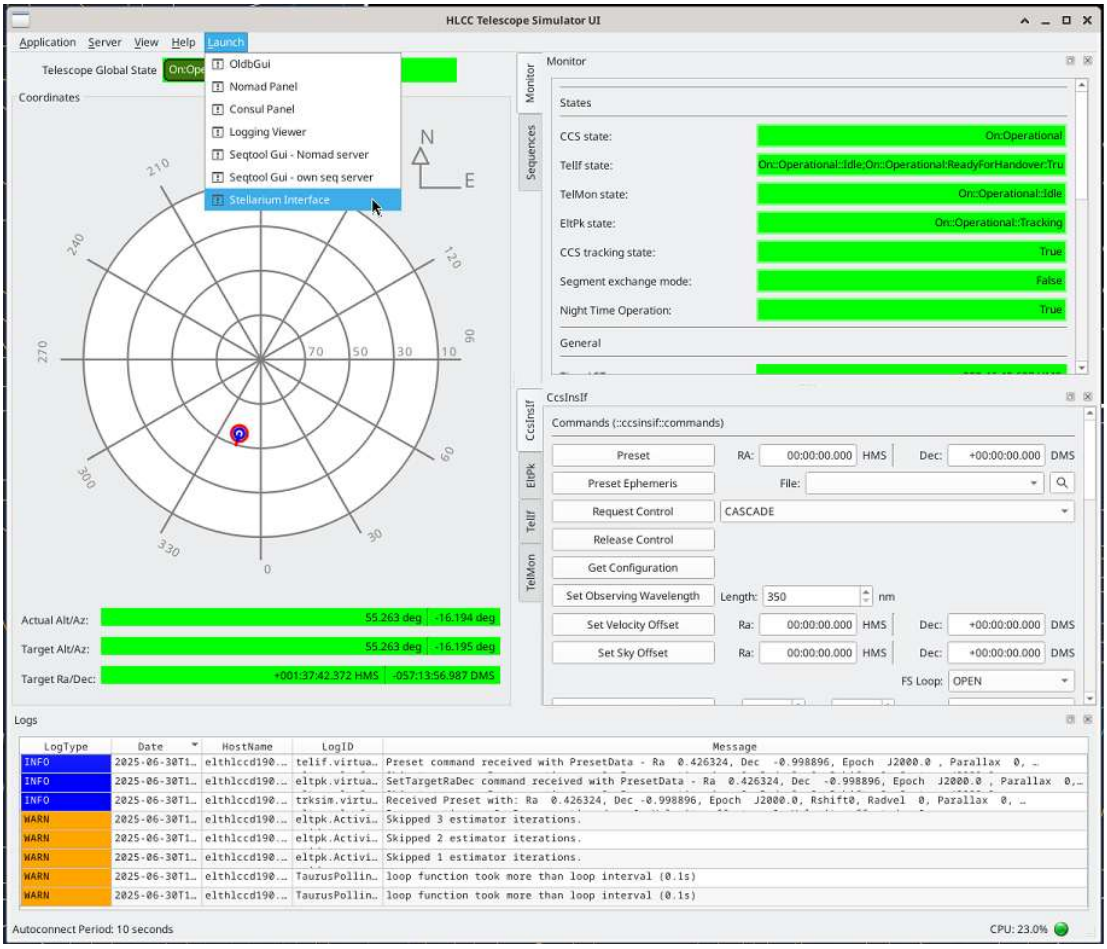


11.1 Startup the GUI

The gui can be started up either from commandline :

% hlccStellariumGui

Or from the menu of hlcctelsimui:



11.2 Features

11.2.1 Parameters

The GUI has one parameter, the IP address of the host where Stellarium is running, which can be set at the top of the GUI.

11.2.2 Layout and actions

The GUI is divided into three main sections, to Configure Stellarium, to Preset Telescope and to Follow a selected object.

Each section is composed of the following items:

- an area where the output of the corresponding action is displayed (on the left side)
- control section (on the right side)

The control section contains:

- status widget
- and two control buttons: the first button starts the Notebook, the second button stops/aborts running action

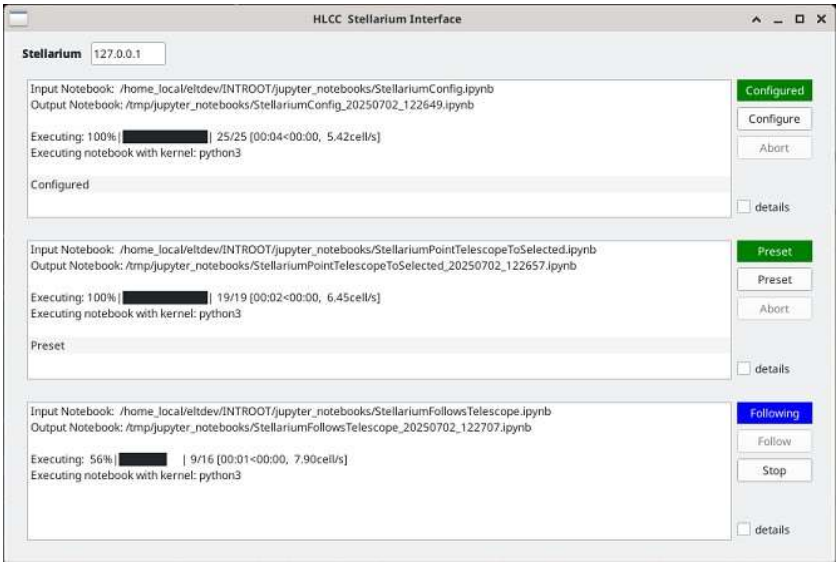
The status widget shows the state of the execution of the related notebook. Initially the state is Idle. A running notebook is indicated with blue color and displays the name of the action. When the notebook is successfully completed, the status turns green. When the notebook has failed or aborted the background of the status widget turns red.

The named action (i.e. Jupiter Notebook) can be started via the button, under the status display and aborted via the button underneath.

When the action is started, the underlying mechanism executes the corresponding hlcc Jupyter Notebook via the papermill tool. The output of this tool is displayed in the white output area.

As default papermill displays the progress in terms of percentage. For more detailed output, select the ‘details’ option using the checkbox.

When the checkbox is on, all the output of the notebook is displayed.



Failure is detected by the return code of papermill. For the cause of the failure check the error messages on the output area. Lines containing text: 'error', 'failure' or 'exception' are highlighted.

Note that the follow action is a continues loop, it runs until the user stops it or there is a failure, e.g. connection error, or selected object is not visible.

During ‘following’ action, it is allowed to select another object via Stellarium and run the preset action to change the object that is followed. The telescope will automatically follow the new object. (feature of hlcc/telif)Temporary files

11.2.3 Generated files

Papermill generates a Notebook with the selected parameter.

The papermill tool generates notebooks with the actual parameters used, these files are stored with the name original notebook extended with timestamp under the directory: /tmp/jupyter_notebooks/

E.g. Output Notebook:

/tmp/jupyter_notebooks/StellariumConfig_20250630_083310.ipynb

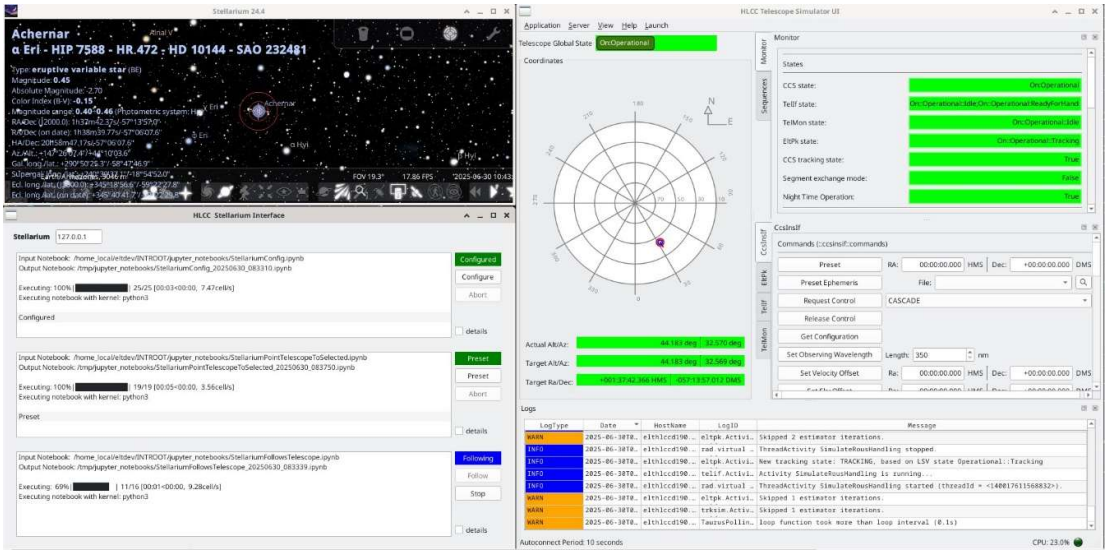


Figure 11.1 Stellarium, hlcctelsimui and hlccStellariumGui working together

11.3 Closing the GUI

Any running action is stopped/aborted when the GUI is shuts down via the X button in the top right corner.