



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:** 2.0

**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
M.Schilling	ESO
H.Sommer	ESO
P.Szubiakowski	ESO
L.C.Goncalves	Critical SW
R.Leao	Critical SW

## Change Record from previous Version

Affected Section(s)	Changes / Reason / Remarks
1.0	First version
1.5	Updated for HLCC <u><b>v1.2.0-alpha4</b></u>
1.5	Updated for HLCC <u><b>v1.2.0-alpha5</b></u>
1.5	Updated for HLCC <u><b>v1.2.0</b></u>



## Contents

1. Introduction.....	7
1.1 Scope.....	7
1.2 Related documents.....	7
1.2.1 Applicable Documents.....	7
1.2.2 Reference documents .....	7
1.3 Acronyms.....	8
1.4 Overview.....	8
1.4.1 HLCC Deployment .....	8
1.4.2 HLCC Architecture .....	9
2. Pointing Kernel functional design.....	10
2.1 Preset/track with target given in catalog (mean) coordinates .....	11
2.2 Published data.....	14
2.3 Preset/track with target given in ephemeris .....	15
2.4 Handling of time.....	15
2.5 Backward calculation of mean (ra,dec) .....	15
3. Install and Build HLCC .....	16
3.1 Prerequisite.....	16
3.2 Installation from RPM .....	19
3.3 Installation from sources.....	19
3.4 Dependencies.....	22
4. Run the HLCC code standalone .....	23
5. Configuring Nomad and Consul .....	26
5.1 Environment setup.....	27
5.2 Nomad and Consul configuration .....	27
5.3 Nomad and Consul startup.....	28
5.4 Loading the OLDB .....	28
5.5 Startup/shutdown of HLCC services using Nomad jobs .....	30
6. Telescope Simulator.....	31
6.1 Overview.....	31
6.2 Telescope Simulator deliverables .....	31
6.3 Startup/Shutdown with Nomad/Consul Deployment .....	32
6.3.1 Deployment files and structure.....	32
6.3.2 Available deployments .....	32



6.3.3	Deployment startup .....	33
6.3.4	Deployment shutdown.....	33
6.4	Implemented features.....	34
6.4.1	Standard Commands .....	34
6.4.2	CCS-INS ICS Commands on control network .....	34
6.4.3	Monitor data .....	35
6.4.4	MEDADAQ Data Acquisition commands on control network.....	37
6.4.5	RAD Application commands .....	37
6.4.6	Simulation specific commands.....	38
6.4.7	Supported commands and replies/error replies .....	38
6.5	hlcctelsimui documentation .....	42
6.5.1	Connection .....	45
6.5.2	Command tabs .....	46
6.5.3	Monitor panel configuration.....	47
6.5.4	Sequences panel.....	47
6.5.5	Telescope State panel.....	48
6.5.6	List of HLCC Servers configuration.....	49
6.5.7	Known Issues .....	50
6.5.8	ToDo.....	50
6.5.9	UI Configuration (Telescope Monitor, Status, Scripts....) .....	50
6.6	hlcctelsimui_mon documentation .....	54
6.7	telifsim documentation.....	56
6.7.1	Sending commands.....	56
6.7.2	telifsim State Machine .....	57
6.7.3	Rous Documentation.....	58
6.7.4	Preset Documentation.....	60
6.8	telmon documentation .....	62
6.8.1	Sending commands.....	62
6.8.2	telmon State Machine.....	64
6.8.3	Adding new estimation scripts to Telmon.....	64
6.8.4	Listing the scripts to be run by telmon.....	65
6.8.5	Reloading Scripts .....	66
6.8.6	Error handling.....	66
6.8.7	Telmon Estimation Scripts .....	66
6.9	eltpksim documentation.....	67



6.9.1	Sending commands.....	67
6.9.2	eltpksim State Machine .....	68
6.9.3	Configuration options .....	69
6.10	lsvsim documentation .....	70
6.10.1	Sending Commands.....	70
6.10.2	Configuration .....	71
6.10.3	Other documentation.....	73
6.10.4	Code Execution Statistics.....	73
6.10.5	PID Temperature Controller example .....	75
6.11	Segexmgr documentation.....	77
6.11.1	Sending commands.....	77
6.12	Astronomical site monitor simulator documentation.....	77
6.12.1	Sending commands.....	78
6.12.2	Changing parameters.....	78
6.13	HLCC utilities.....	79
6.13.1	\$ consulGetUri --help .....	79
6.13.2	\$ eltpksim_preset --help .....	80
6.13.3	\$ telifsim_preset --help .....	80
6.13.4	\$ telif_subscriber --help.....	81
6.13.5	\$ telif_publisher --help.....	83
6.13.6	\$ telifsim_get_config --help .....	85
6.13.7	\$ telifsim_metadaqcmds --help .....	85
6.13.8	\$ telifsim_offset_setsky --help .....	86
6.13.9	\$ telifsim_request_release_control --help .....	87
6.13.10	\$ telifsim_rous --help .....	87
6.14	Jupyter notebooks .....	88
6.14.1	Start jupyter server and browser client on the same machine with one command ...	88
6.14.2	Start jupyter server and browser client on the different machines.....	89
6.14.3	Executing Jupyter Notebooks .....	90
6.15	Sequencer scripts.....	92
6.15.1	Run an individual sequence on the command line .....	92
6.15.2	Run through sequencer server and GUI .....	92
6.15.3	Known Issues .....	94
6.15.4	ToDo.....	95
6.16	Change Telescope site coordinates .....	95



6.16.1 Change telescope location in configuration files .....	95
6.16.2 Change telescope location with configuration commands .....	95
6.17 Change Telescope operation mode .....	96
6.17.1 DayTime sequence .....	96
6.17.2 NightTime sequence .....	97
6.17.3 Extending Daytime and NightTime sequences .....	97
7. Known issues .....	97
8. Other HLCC applications .....	98
8.1 telif documentation .....	98
8.1.1 Start the application:.....	98
8.1.2 Sending commands.....	98
9. Stellarium: Installation and configuration .....	99
9.1 Installation.....	99
9.2 Configuration .....	99
9.3 Running the notebooks.....	104
9.4 Learning and debugging.....	104
9.5 To know more.....	105



# 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 access to a page you would like to read, ask the HLCC team or your 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
PFS	Pre-Focal Station

## 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 below.

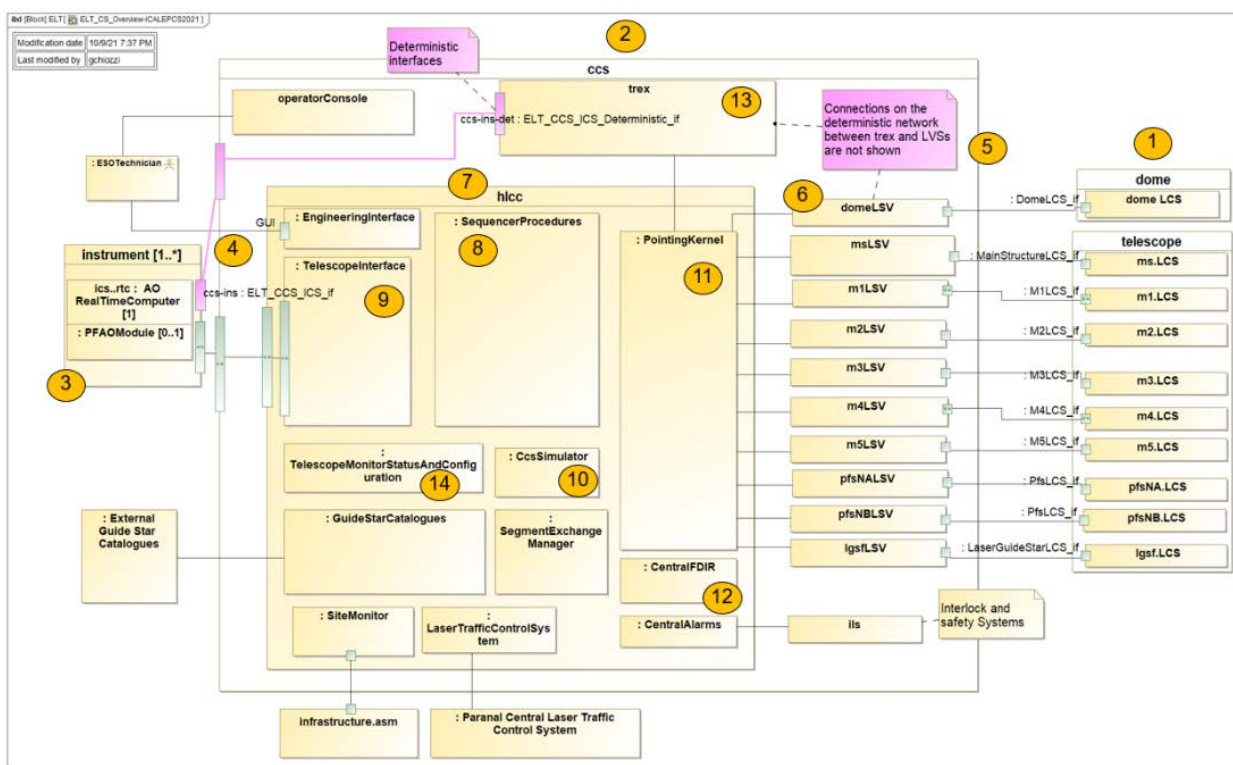


Figure 1

Among these:

- The interface toward the instruments, defined in the CCS-INS ICD [RD1], is allocated to the single component in Figure 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-(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-(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-(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-(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 [ptk](#) module.

The code includes plenty of comments, but a description of the algorithms will be added here.

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/>).

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 ([Document Classification: ESO Internal \[Confidential for Non-ESO Staff\]](http://star-</a></p></div><div data-bbox=)



[www.rl.ac.uk/docs/sun67.htx/sun67.html](http://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) (<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  $[\alpha, \delta]$ ) 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  $[h, \delta]$ ) 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](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 subsystem:*
  - 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 =  $0.65 \times 10^{-6}$  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
Target RA/DEC	x	x	x	
Target ALT/AZ	x	x	x	
Current ALT/AZ	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	
Time TAI, UTC	x			
Parallactic angle, north angle etc.	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: 20231210 – Git tag: [v1.2.0](#)

### 3.1 Prerequisite

These instructions are targeted at building and installing HLCC on a development Virtual Machine  
These instructions are tested with ELT DevEnv

- 4.8.0

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.

CentOS with ELT DevEnv 3.X is NOT supported any more.

- This document describes how to install hlcc and the hlcc telescope simulator
  - From RPM (3.2)
  - From sources extracted from the git repository (3.3)

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 \(eso.org\)](#)
- The standard ELT virtual machines are updated periodically by eltmgr.  
In case manual updates are necessary, instructions are in this document:

[E-ELT Linux Installation Guide](#)

In order to install the latest official DevEnv release, the short story is:

As root:

```
$ dnf -y update puppet-elt
```

```
$ /root/elt/puppet-force-align
```

```
$ /root/elt/puppet-check # to verify the installation
```

```
$ reboot # to make sure all services are properly restarted, if needed
```





To update instead to the latest **ELT DevEnv Beta** package issue as root the following command:

```
$ /root/bin/update-puppet-beta
```

*Notice that RPM packages you might have installed by hand, like rad or seq should be updated as well automatically because of their dependencies. After the update, check the installed versions of these packages (see below)*

- 16 GB of RAM are required for compilation, due to [KB: g++: internal compiler error: Killed](#). Please check this first.
- After a clean installation of DevEnv on a new machine (and in some cases after a DevEnv upgrade):

*It is necessary to perform some additional configuration as described here:*

<https://gitlab.eso.org/ecs/eltsw-docs/-/wikis/KnowledgeBase/CII#how-to-post-install-cii-services>

Typically you need the following commands on a developer VM, **as root**:

```
cii-services stop all
# Run the postinstall scripts
/elt/ciisrv/postinstall/cii-postinstall role_ownserver
cii-services start all
cii-services info

....logout and login again
```

If

```
    cii-services stop all
```

hangs, kill -9 the logstash process. It might hang now if it tries to send logs to elastic after elastic has been already stopped.

- The accounts used are:
  - eltdev - Development account, where the software runs
  - eltmgr - ELT development environment configuration account
- If you are working on a completely clean machine, you first need to configure the eltdev account. For a typical/basic development VM, it is sufficient to add to **.bash\_profile**:



# User specific environment and startup programs

```
# echo "Greeting from your ~/.bash_profile: setting CONFIG_LOCAL_DB, INTROOT and  
PREFIX for you ..."
```

```
export CONFIG_LOCAL_DB=$HOME/localdb
```

```
export INTROOT=$HOME/INTROOT
```

```
export PREFIX=$INTROOT
```

# On a development machine, in order to run integration tests

# without having to stop and cleanup CII services, add also

# See: <https://jira.eso.org/browse/ETCS-630>

```
export ELT_DEV_HOST=1
```

module load introot

Alternatively these variables can be set in the lua file in modulefiles/private.lua

- 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

- 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>
- After having configured the machine, logout from all shells and login again

## 3.2 Installation from RPM

- The following steps describe how to install/upgrade an hlcc installation starting from rpm packages.
- Using RPM, all dependencies are installed automatically, but since packages are evolving it might be necessary to execute additional steps.
- To install hlcc with one single command, **as root**:

```
> dnf install elt-hlcc-if* elt-hlcc-sw*
```

## 3.3 Installation from sources

- The following 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
```

- 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 and from the parent folder where you have extracted the source code from git, verify if all required develop RPM packages needed to build from sources are installed, in the proper versions, using the commands:
  - > dnf builddep --spec hlcc/elt-hlcc-if.spec
  - > dnf builddep --spec hlcc/elt-hlcc-sw.spec

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

**Since you are installing from sources, `elt-hlcc-sw.spec` will normally report that the other `elt-hlcc` packages are missing and will propose to install them.**

**Do NOT do it, since you want to work only with sources (see: [\[ETCS-1311\]](#) RPM spec files used when calling `dnf builddep` for source installation shall not check hlcc RPMs - ESO JIRA Projects).**

Here the expected output of the commands with the currently required dependencies:

```
[root@elthlccd64 hlcc]# dnf builddep --spec elt-hlcc-if.spec
Last metadata expiration check: 0:00:11 ago on Thu 25 Jan 2024 01:02:24 PM UTC.
Package Lmod-8.4.23-1.fc34.x86_64 is already installed.
Package elt-common-system-1.1.0-1.fc34.noarch is already installed.
Package elt-ecsif-devel-1.1.0-6.fc34.x86_64 is already installed.
Package elt-mal-devel-3.1.4-1.fc34.x86_64 is already installed.
Package elt-wtools-3.3.1-2.fc34.noarch is already installed.
Package gcc-11.3.1-2.fc34.x86_64 is already installed.
Package gcc-c++-11.3.1-2.fc34.x86_64 is already installed.
Package pkgconf-1.7.3-6.fc34.x86_64 is already installed.
Package gmock-devel-1.10.0-6.fc34.x86_64 is already installed.
Package gtest-devel-1.10.0-6.fc34.x86_64 is already installed.
Package python-unversioned-command-3.9.13-1.fc34.noarch is already installed.
Package python3-devel-3.9.13-1.fc34.x86_64 is already installed.
Package python3-pytest-6.2.2-1.fc34.noarch is already installed.
Package python3-pytest-custom_exit_code-0.3.0-1.fc34.noarch is already installed.
Dependencies resolved.
Nothing to do.
Complete!
```



```
[root@elthlccd64 hlcc]# dnf builddep --spec elt-hlcc-sw.spec
Last metadata expiration check: 0:00:14 ago on Thu 25 Jan 2024 01:06:09 PM UTC.
Package Lmod-8.4.23-1.fc34.x86_64 is already installed.
Package elt-ciisrv-client-api-devel-3.1.2-1.fc34.x86_64 is already installed.
Package elt-ciisrv-config-ng-devel-3.1.2-1.fc34.x86_64 is already installed.
Package elt-ciisrv-oldb-client-devel-3.1.2-1.fc34.x86_64 is already installed.
Package elt-ciisrv-robot-library-3.3.0-1.fc34.noarch is already installed.
Package elt-common-system-1.1.0-1.fc34.noarch is already installed.
Package elt-cut-devel-2.0.8-4.fc34.x86_64 is already installed.
Package elt-ecsif-devel-1.1.0-6.fc34.x86_64 is already installed.
Package elt-hlcc-if-devel-1.2.0-1.fc34.x86_64 is already installed.
Package elt-msgsend-0.5.0-2.fc34.noarch is already installed.
Package elt-oldbloader-0.1.0-4.fc34.noarch is already installed.
Package elt-ptk-devel-3.0.0-1.fc34.x86_64 is already installed.
Package elt-rad-devel-5.2.0-5.fc34.x86_64 is already installed.
Package elt-seq-devel-4.1.0-1.fc34.x86_64 is already installed.
Package elt-taiclock-devel-0.5.0-4.fc34.x86_64 is already installed.
Package elt-trs-common-devel-0.1.0-2.fc34.x86_64 is already installed.
Package elt-wttools-3.3.1-2.fc34.noarch is already installed.
Package gcc-11.3.1-2.fc34.x86_64 is already installed.
Package gcc-c++-11.3.1-2.fc34.x86_64 is already installed.
Package qt5-qtbase-devel-5.15.2-30.fc34.x86_64 is already installed.
Package qt5-qttools-devel-5.15.2-5.fc34.x86_64 is already installed.
Package qt5-qtbase-devel-5.15.2-30.fc34.x86_64 is already installed.
Package qt5-qtsvg-devel-5.15.2-4.fc34.x86_64 is already installed.
Package qt5-qtbase-devel-5.15.2-30.fc34.x86_64 is already installed.
Package azmq-devel-1.0.3-3.fc34.x86_64 is already installed.
Package gmock-devel-1.10.0-6.fc34.x86_64 is already installed.
Package gtest-devel-1.10.0-6.fc34.x86_64 is already installed.
Package guidelines-support-library-pc-3.1.0-1.fc34.noarch is already installed.
Package python3-pyside2-devel-1:5.15.2-3.fc34.x86_64 is already installed.
Package yaml-cpp-devel-0.6.3-4.fc34.x86_64 is already installed.
Package python-unversioned-command-3.9.13-1.fc34.noarch is already installed.
Package python3-devel-3.9.13-1.fc34.x86_64 is already installed.
Package python3-astropy-4.2-3.fc34.x86_64 is already installed.
Package python3-mock-3.0.5-14.fc34.noarch is already installed.
Package python3-nose-1.3.7-33.fc34.noarch is already installed.
Package python3-pytest-6.2.2-1.fc34.noarch is already installed.
Package python3-pytest-custom_exit_code-0.3.0-1.fc34.noarch is already installed.
```



```
Package python3-pytest-qt-4.2.0-1.fc34.noarch is already installed.
```

```
Package python3-consul2-0.1.5-1.fc34.noarch is already installed.
```

```
Package python3-aurus-5.1.4-1.fc34.noarch is already installed.
```

```
Dependencies resolved.
```

```
Nothing to do.
```

```
Complete!
```

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

```
> cd hlcc/interface
> waf configure build install
> cd ../software
> waf configure build install
```
- *it is also very often necessary to cleanup and reload the OLDB as described in section 5.4 (after having installed the latest HLCC).*

## 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.
- 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. Run the HLCC code standalone

This section describes how to run HLCC as a standalone application, typically on a development machine.

- Initial configuration

On a newly installed DevEnv machine, it is necessary to run the CII Post Install procedure that configures the plain CII services coming from the DevEnv RPMs. Details are described here: <https://gitlab.eso.org/ecs/eltsw-docs/-/wikis/KnowledgeBase/CII#how-to-post-install-cii-services>

We configure the HLCC hosts as "own servers" (= single-developer set-up):

Note you will need the root password.

```
$ su -c "/elt/ciisrv/postinstall/cii-postinstall role_ownserver"  
Password: <enter root password>  
CII PostInstall (1.1-4)
```

- Start CII services

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

Details are described here:



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

This excerpt assumes you have already done initial configuration (see 3.1 above).

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

This is the status I have now and that seems to work fine:

```
$ cii-services info
CII Services Tool (20231129)
Collecting information.....
Collecting information.....

Installations on this host:
ElasticSearch |active:yes |install:/usr/share/elasticsearch/bin/elasticsearch
Redis         |active:yes |install:/usr/bin/redis-server
MinIO         |active:yes |install:/usr/local/bin/minio
ConfigService |active:yes |install:/elt/ciisrv/bin/config-service |ini:
TelemService  |active:no  |install:
Filebeat      |active:no  |install:/usr/share/filebeat/bin/filebeat
Logstash      |active:no  |install:/usr/share/logstash/bin/logstash
Kibana        |active:no  |install:/usr/share/kibana/bin/kibana
Jaeger        |active:no  |install:/usr/local/bin/jaeger-all-in-one
Alarm         |active:no  |install:
Kafka         |active:no  |install:
ConfigClient  |ini:

Addresses of Services:
IntCfg / Local-DB |access:no |file:/home_local/eltdev/localdb
IntCfg / ElasticSearch |access:yes |host:ciielastichost(IP:127.0.0.1)
IntCfg / MinIO     |access:yes |host:ciihdfshost(IP:127.0.0.1)
```





```
IntCfg / Service      |access:yes      |host:ciiconfservicehost(IP:127.0.0.1)
|host2:ciiconfservicehost2(IP:127.0.0.1)
OLDB / Redis          |access:yes      |host:localhost(IP:::1)
OLDB / MinIO          |access:yes      |host:ciihdfshost(IP:127.0.0.1)
Telem / Service       |access:no       |host:ciiarhivehost(IP:127.0.0.1)
```

**Statistics of Services:**

```
IntCfg / Service      |NrOfRequests: 2102496 |UpTime: 64655 minutes
OLDB / Redis          |total_connections_received:72614   |rejected_connections:0
Telemetry / Service
```

```
$ cii-services status
```

```
CII Services Tool (20221103)
```

```
Collecting information.....
```

**Status Summary:**

```
(Note: '--' means 'not available')
```

```
OK InternalConfig from Central Server
```

```
OK OLDB Normal Mode
```

```
OK OLDB with Blob Values
```

```
-- Telemetry
```

As an alternative to starting redis as a service, it is possible to use a low footprint stand-alone OLDB server:

```
$ supervisoryappOldbServer-start.sh
```

*Attention: you have to wait some time (~1 minute at least) before Redis is up and Config Service is available.*



- **IMPORTANT**

**Problems might surface with memory usage for Elastic (being investigated, see: <https://jira.eso.org/browse/ECII-575> ).**

If the Elastic process dies or uses a lot of CPU and therefore the OLDB fails or is very slow, you have to increase the memory following this procedure:

1. stop elastic (sudo ci-services stop elastic)
- 2 As root, edit /etc/elasticsearch/jvm.options:  
in line 22,23: set 1gb for min and max heap (PSZ says 2Gb)
3. start elastic (sudo ci-services start elastic)

and see if that helps.

More over, keep an eye at disk usage. Elastic might fill up the disk with time.

See this KB note:

<https://gitlab.eso.org/ecs/eltsw-docs/-/wikis/KnowledgeBase/CII#cannot-create-datapoints-oldb>

- Run the Configuration Service GUI to monitor and inspect the configuration (works only if using the real configuration database, not the embedded one):

```
$ config-gui
```

- Run the OLDB Service GUI to monitor and inspect the OLDB:

```
$ oldbGui
```

- 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: [Loading the OLDB](#)
- The first thing to do is in any case to load the OLDB; see: [Loading the OLDB \(section \(5.4\)\)](#)
- 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.

## 5. Configuring Nomad and Consul

Deployment using Nomad and Consul is the standard way to run the system with respect to standalone manual startup.



Manual startup of one or more processes might still be necessary/useful during development or debugging.

This page contains instructions for the configuration of Nomad and Consul on a development machine, with Nomad and Consul running on the machine itself.

Other deployments (like instrumentation or ECM) have a specific setup, not explicitly considered here.

More general instructions for using HLCC with Nomad and Consul are in this page:

- [software/nomad/examples/README · master · ccs / hlcc · GitLab \(eso.org\)](#)

On the short (using as example the elthlccd64 host, used for RPM and Nomad testing):

## 5.1 Environment setup

The environment variables `NOMAD_ADDR` and `CONSUL_HTTP_ADDR` must be set, for example in your `~/modulefiles/private.lua` file (see elthlccd63 for an example):

```
setenv ("NOMAD_ADDR", "http://elthlccd63.hq.eso.org:4646")
```

```
setenv ("CONSUL_HTTP_ADDR", "http://elthlccd63.hq.eso.org:8500")
```

## 5.2 Nomad and Consul configuration

Nomad and Consul configuration files should be prepared automatically by the installation procedures / puppet scripts.

If there are problems with Nomad/Consul configuration, a basic knowledge on the configuration of the tools is necessary. The baic steps to follow should be very simple:

1. Edit the `/etc/nomad.d/nomad.hcl` file (that shall be owned by eltdev)
2. Compare the current file with the example/template and check in particular that the IP addresses of your host and the `host_network` sections for deterministic and not deterministic networks are correct.  
Use the template files in [software/nomad/examples · master · ccs / hlcc · GitLab \(eso.org\)](#) as a starting point.
3. If the 'nomad.hcl' was modified in the previous step we need to restart the Nomad service (still as root) so Nomad can pick up the new file.

```
root$ systemctl restart nomad
```

4. Follow exactly the same procedure for the `/etc/consul.d/consul.hcl` file



## 5.3 Nomad and Consul startup

To have the Nomad and Consul services starting automatically at boot time, as root:

```
$ systemctl enable consul  
$ systemctl enable nomad
```

Nomad and Consul services shall be otherwise started manually as root:

```
$ systemctl start consul  
$ systemctl start nomad
```

## 5.4 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 if 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.



- `--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
  - inst - instruments
- `--abort` - Aborts on error (by default would continue with the next item instead)

Therefore.....

To load the Oldb:

```
hlccOldbloader -d dev
```

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 oldb-gui
- enable write mode: Check options -> write enabled
- select the `elt\telif` node
- press the delete button (or delete just the nodes that give problems) and confirm deletion.

#### 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.

See:

<https://gitlab.eso.org/ecs/eltsw-docs/-/wikis/KnowledgeBase/CII#datapoint-already-exists-oldb>,

Solution 3 is the most radical but safer option:

```
# Clean up the OLDB databases
config-initEs.sh
oldb-initEs
```



```
redis-cli flushall
sudo cii-services stop config
sudo cii-services start config
```

**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 there are no leading or trailing "/" 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

## 5.5 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, as described below in section (6) for the telescope simulator.

Therefore this section can be normally safely skipped.

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

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

or

```
$ nomad job run $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 machine on any cluster, assuming [NOMAD\\_ADDR](#) is properly set.

See also (5.5)

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>

## 6. Telescope Simulator

### 6.1 Overview

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

### 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:
  - telifsim
  - eltpksim
  - telmon
  - lsvsim (lsv simulator configurable/generic process)
- User interfaces
  - hlctelsimui
  - hlctelsimui\_mon



- Utilities
  - See: (6.13)

## 6.3 Startup/Shutdown with Nomad/Consul Deployment

### 6.3.1 Deployment files and structure

**20230515 - Attention: if upgrading from a previous installation, verify and update the nomad configuration for changes implemented with HLCC v1.1.0-beta4**

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 documentat6ion.

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. Every deployment has a startup sequence and a shutdown sequence, all sequences are located in the hlcc repository path: "hlcc/software/seq/src/hlccseq/"

Also in the same path we find other 2 helper files:

- `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.
- `hlccseq_lib.py` - Contains methods that are common to all deployments and which will be used by sequences.

Deployments 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 as described in section 5.

### 6.3.2 Available deployments

Currently 3 different deployments are available:

- DEV: Development environment on one single virtual machine
- ECM: ELT Control Model
- INS: Integration with instrumentation software for HLCC simulation

Probably in a near future more will be added, with different configuration options.

For these deployments, the startup/shutdown sequence can be launched directly from the `hlccsimui`. Once a deployment is configured in the OLDB, the UI will present them in the Sequences tab and will start an interactive Sequencer GUI when selected.





The sequences can be loaded directly into any Sequencer GUI or executed non interactively with the `seqtool run` command line tool.

### 6.3.3 Deployment startup

Deployment startup 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 sequence and launch it manually:

```
$ seqtool run hlccseq.hlccEcmStartup
```

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

- o `hlccDevStartup.py` - Development VM deployment
- o `hlccEcmStartup.py` - ECM deployment
- o `hlccInsStartup.py` - Instruments deployment

Alternatively, the sequence can be launched from the `seqtool gui` or from the `hlccTelSimui`.

### 6.3.4 Deployment shutdown

Deployment Shutdown sequences 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:

- o `hlccDevShutdown.py` - Development VM deployment
- o `hlccEcmShutdown.py` - ECM deployment
- o `hlccInstShutdown.py` - Instruments deployment

Alternatively the sequence can be launched from the `seqtool gui` or from the `hlccTelSimui`.



## 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]/telifsim/StdCmds`

Example:

```
$> msgsend --uri `consulGetUri -r ifuri -s telifsim -i 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: The `consulGetUri` utility is used to query Consul for the uri dynamically assigned to any of the HLCC processes when they are started up by Nomad.*

### 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
- OffsetSetSky
- SetObservingWavelength
- SetVelocityOffset
- GetConfig
- RousConfig, RousExecute and corresponding monitor points

The server URI is:



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

Example:

```
$> msgsend --uri `consulGetUri -r ifuri -s telifsim -i Commands`  
::ccsinsif::Commands::RequestControl '"CASCADE"'
```

*(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 table 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.

The table 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 three.

Name   Verif(impl)	Ref(impl)	Pub-Address (URI)   Defining file	Data Entity 	OLDB-DPs 	Freq(1CD) 	Freq(impl) 
telif:ccs:target_observed_altaz   Real		dds.ps:///TELIF_CCS_TRG_OBS_ALTAZ   ccsinsif/1cd/src/ccsinsif.xml	CPP -> Ccsinsif.hpp - ccsinsif::AltAz 	CCS_TARGET_OBSERVED_ALTAZ 		1   20
			PY -> ModCcsinsif.Ccsinsif.AltAz 			
telif:ccs:current_observed_altaz   Simulated		dds.ps:///TELIF_CCS_CUR_OBS_ALTAZ   ccsinsif/1cd/src/ccsinsif.xml	CPP -> Ccsinsif.hpp - ccsinsif::AltAz 	CCS_CURRENT_OBSERVED_ALTAZ 		1   20
			PY -> ModCcsinsif.Ccsinsif.AltAz 			
telif:ccs:deterministic   Simulated		mudpi.ps:///224.0.0.1:12783/TELIF_CCS_PK_POS   ccsinsif/1cd-determ/src/ccsinsif.xml	CPP -> Ccsinsif.hpp - ccsinsif::PointingKernelPositions 			20   20
			PY -> ModCcsinsif.Ccsinsif.PointingKernelPositions 			
telif:current_time   Real		dds.ps:///TELIF_CUR_TIME   ccsinsif/1cd/src/ccsinsif.xml	CPP -> Ccsinsif.hpp - ccsinsif::CurrentTime 	TIME_1ST 		1   20
			PY -> ModCcsinsif.Ccsinsif.CurrentTime 			
telif:tracking_data   Simulated		dds.ps:///TELIF_TRK_DATA   ccsinsif/1cd/src/ccsinsif.xml	CPP -> Ccsinsif.hpp - ccsinsif::TrackingData 	CCS_PARALLACTIC_ANGLE 		1   20
			PY -> ModCcsinsif.Ccsinsif.TrackingData 	CCS_NORTH_ANGLE 		
				CCS_PUPIL_ANGLE 		
				CCS_ELEVATION_DIRECTION_ANGLE 		
				CCS_RADEC_AT_XY_FROM_GUIDE_STARS 		
				CCS_OBSERVED_ALTAZ_AT_REQUESTED_XY 		
telif:tracking_data_extended   <none>		dds.ps:///TELIF_TRK_DATA_EXT   ccsinsif/1cd/src/ccsinsif.xml	CPP -> Ccsinsif.hpp - ccsinsif::TrackingDataExtended 	CCS_REMAINING_TRACKING_TIME_VALUE 		1   <none>
			PY -> ModCcsinsif.Ccsinsif.TrackingDataExtended 			



The command to extract the information in the table above (plus more) from the code is:

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

To subscribe to monitor data, see for example, the utility (6.13.4):

```
$ telif_subscriber --help
```

#### 6.4.4 MEDADAQ 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]/telifsim/MetaDaq
```

Example:

```
$> msgsend --uri $(consulGetUri -r ifuri -s telifsim -i 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]/telifsim/AppCmds
```

Example:

```
$> msgsend --uri $(consulGetUri -r ifuri -s telifsim -i 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](https://telifsimif.icd/src/telifsimif.xml) · master · ccs / hlcc · GitLab (eso.org)

The following commands are currently (partially) implemented:

- MoveToNamedPos
- MoveToAltAzPos
- UpdateRousTimer

- SetPresetSequenceMode

Set the mode to be used when handling a preset command:

- NO\_SEQUENCE – executes an hard-coded preset algorithm
- HEDLESS – 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]/telifsim/SimCmds`

Example:

```
$> msgsend --uri `consulGetUri -r ifuri -s telifsim -i SimCmds`  
::telifsimif::SimCmds::MoveToNamedPos "PARK"
```

### 6.4.7 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 eltpksim	"CONNECTION TO ELTPKSIM NOT ESTABLISHED!" (ccsinsif::GeneralException)
		Command rejected.	"COMMAND REJECTED FROM ELTPKSIM: " + 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)
	OffsetSetSky	Command accepted	"OK"
		Ready for handover = false or not successfully accepted by eltpksim	"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: UNKNOWN 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 eltpksim	"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 eltpksim	"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...		

## 6.5 hlcctelsimui documentation

This is the HLCC Simulator operator UI.

For the time being it provides only limited functionality:



- Connect/disconnect telifsim, telmon and eltpksim processes
- Monitor logs in the system
- Send standard commands to telifsim, telmon and eltpksim 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

To start the application:

```
$ hlcctelsimui
```

**Note:**

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

```
cii.olddb:///elt/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 6.4.

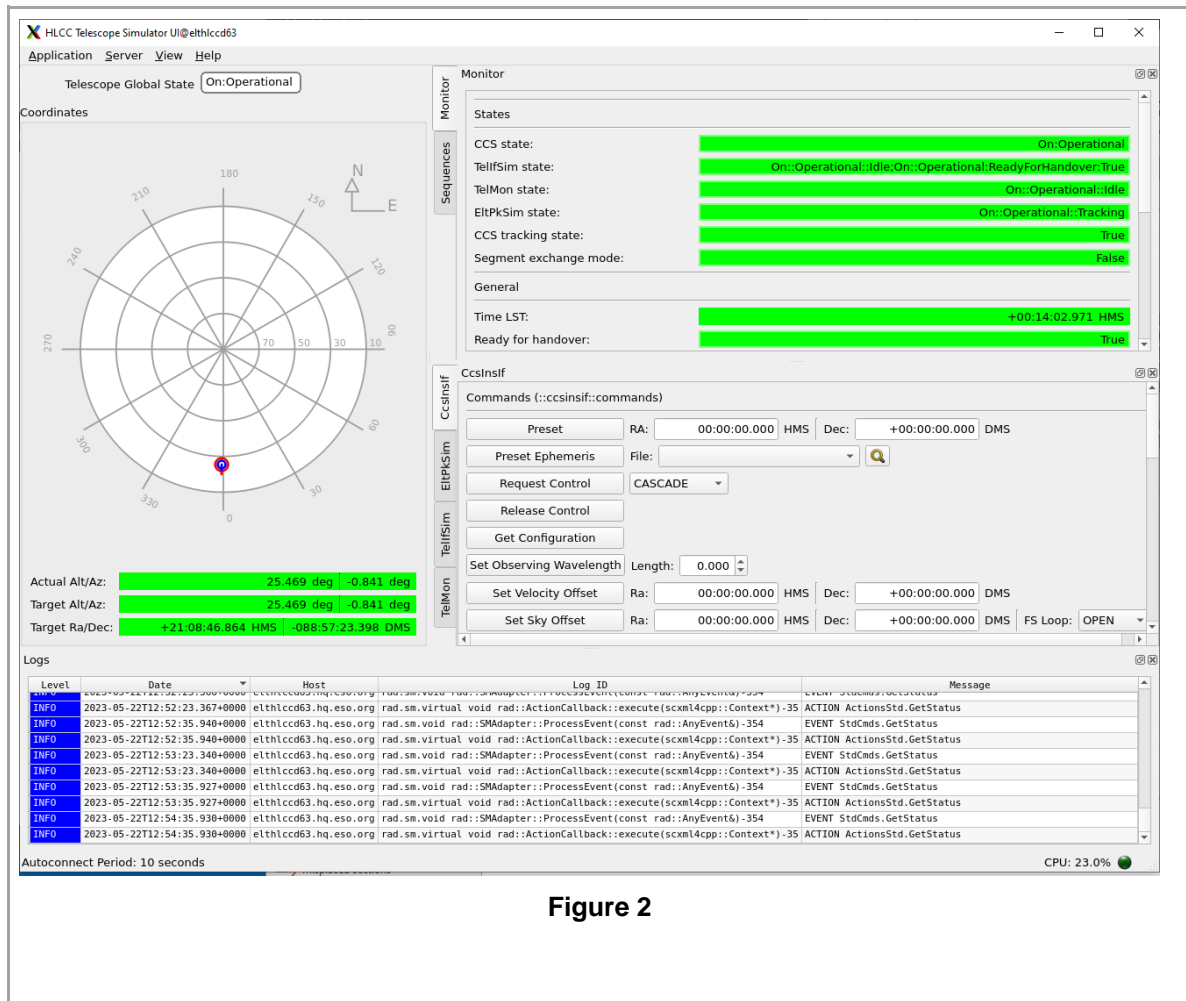


Figure 2

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

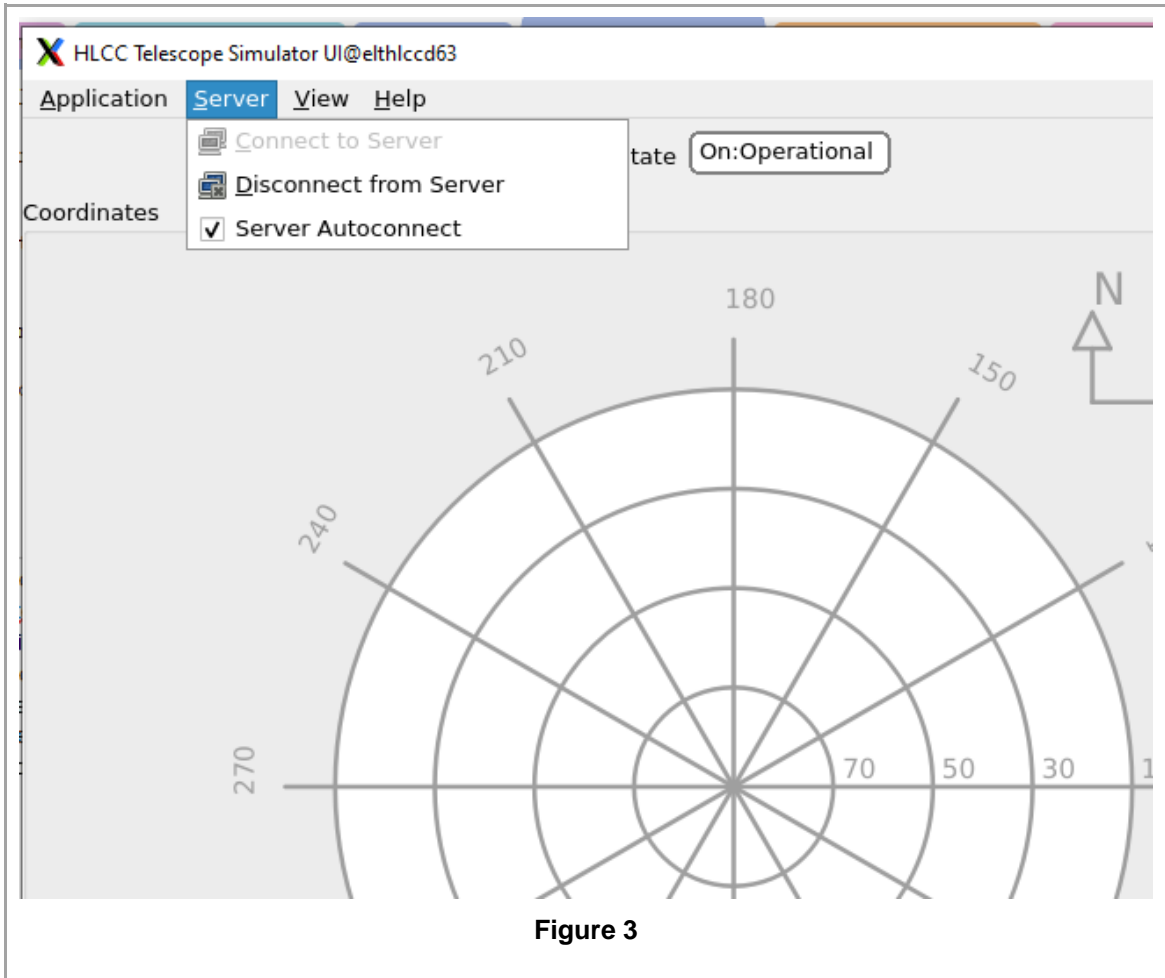
Override the default formatter

#### 6.5.1 Connection

From the Server menu (Figure 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.



### 6.5.2 Command tabs

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

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

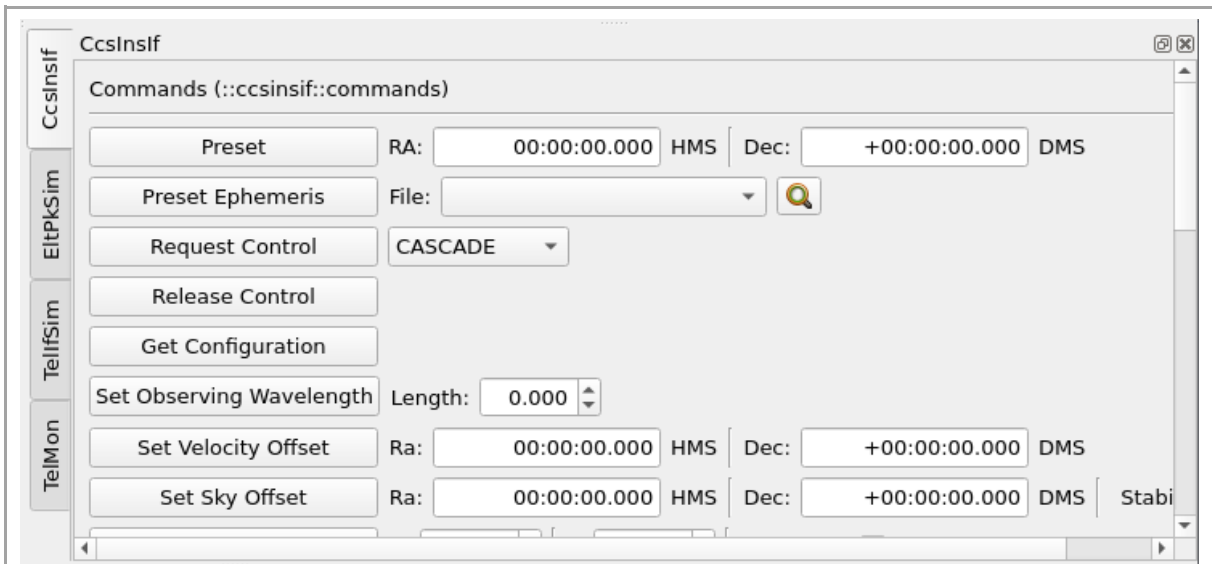


Figure 4

### 6.5.3 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 6.5.9.

### 6.5.4 Sequences panel

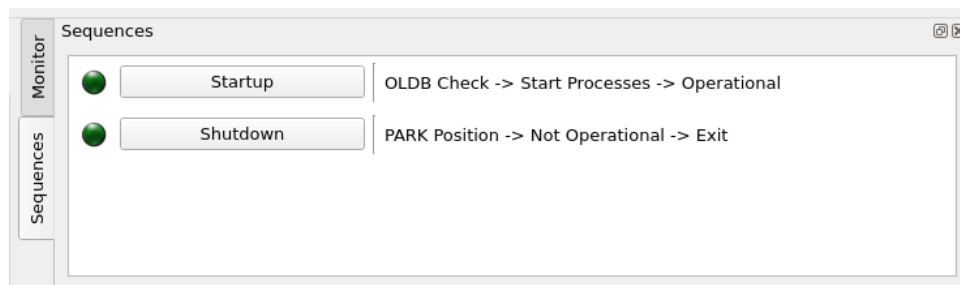
The Sequences panel from the UI dock widget helps interact with pre-implemented scripts (Sequencer scripts in particular) and to execute them **detached** from the main process.

The contents of this panel depends on the deployment.

As can be seen in Figure 5, 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 is selected, a Sequencer GUI is started with an own Sequencer server and Sequence is loaded in the server. The user/operator can then start the ssequence and interact with it using the Sequencer GUI.



**Figure 5**

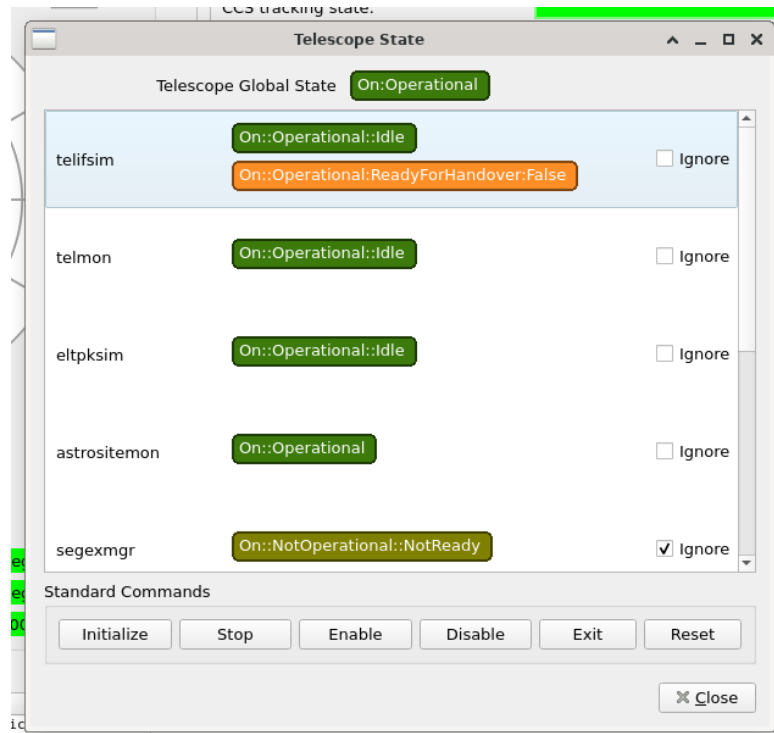
It's important to notice that only one script sequence of the same type can be executed at a given time, therefore if one is already running an information log will notify the user. There are instead no limitations in running multiple different sequences at the same time. It is responsibility of the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

For details on the configuration of this panel see section 6.5.9.3.

### 6.5.5 Telescope State panel

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





**Figure 6**

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 with request the process to exit, but if the process is managed by Nomad, the process will be typically restarted automatically*

### 6.5.6 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 6.5.9



### 6.5.7 Known Issues

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

- Too many open files errors  
If the application fails with "too many open files" errors, check the /var/log/elt folder. There might be too many files there, typically produced by the logging system. Cleanup the folder to solve the problem, using the `rm -f *` command as root.  
See: <https://jira.eso.org/browse/ECII-529>

### 6.5.8 ToDo

Missing/incomplete important functionality:

- None now

### 6.5.9 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`
- `states.json`

These files are located under `"telsimui/resouces/config/hlcc"` 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 `hlccetelsimui`, the application will find first your modified files and use them..

#### 6.5.9.1 Monitor panel configuration:

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

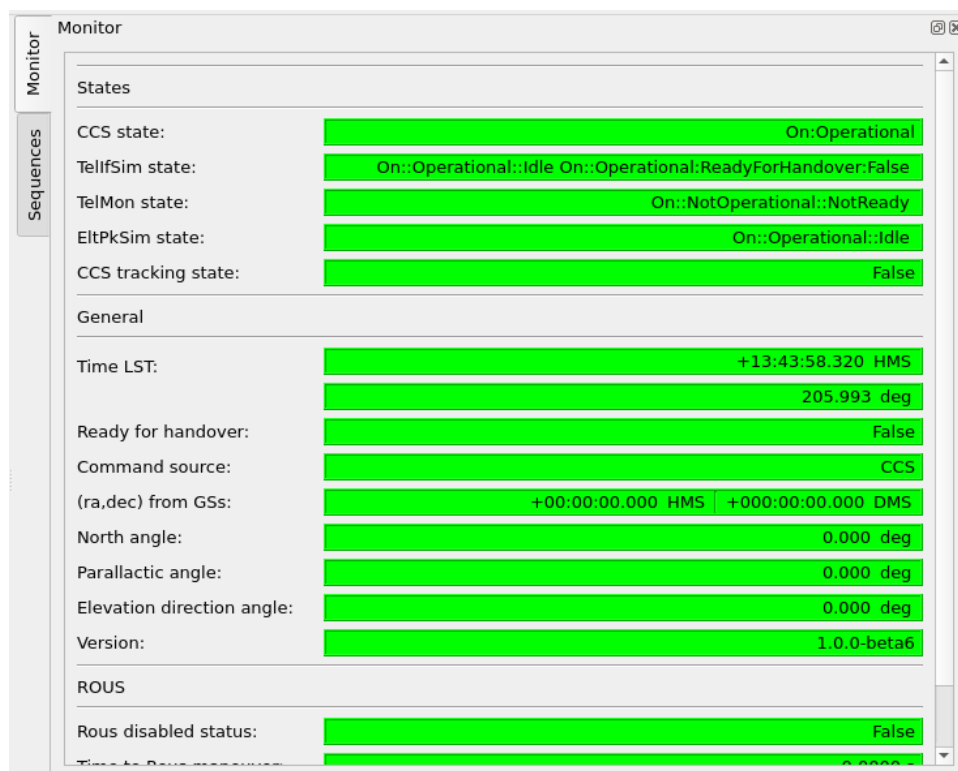


Figure 7

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** chosen to be displayed.

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

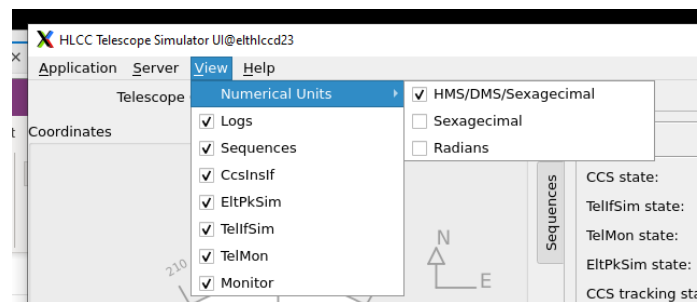


Figure 8

Since the types of widgets supported can change in the future it is advised to check 'monitorpanel\_item.py' for valid types.

```
{
  "version": 1,
  "groups": [
    {
      "name": "General",
      "monitors": [
        {
          "label": "Time LST",
          "uri": "cii.olddb:///elt/telif/time/lst",
          "widgets": ["TaurusRadiansLabel", "TaurusHmsLabel", "TaurusSexagecimalLabel"]
        },
        {
          "label": "Ready for handover",
          "uri": "cii.olddb:///elt/telif/ccs/ready_for_handover",
          "widgets": ["TaurusLabel"]
        },
        {
          "label": "Command source",
          "uri": "cii.olddb:///elt/telif/ccs/command_source",
          "widgets": ["TaurusLabel"]
        },
        {
          "label": "(ra,dec) from GSs",
          "uri": "cii.olddb:///elt/telif/ccs/radec_at_xy_from_guide_stars",
          "indexes": [0, 1],
          "widgets": ["TaurusRadiansLabel", "TaurusHmsDmsLabel", "TaurusSexagecimalLabel"]
        }
      ]
    }
  ]
}
```

Figure 9

#### 6.5.9.2 Processes configuration:

The processes required for the application to work on (TelfSim, EltPkSim, etc, ...) are dynamically retrieved from the telmon OLDB datapoint:

`cii.olddb:///elt/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.



### 6.5.9.3 Sequences Script UI List configuration:

In `"sequences.json"` you can configure the sequencer scripts of 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.

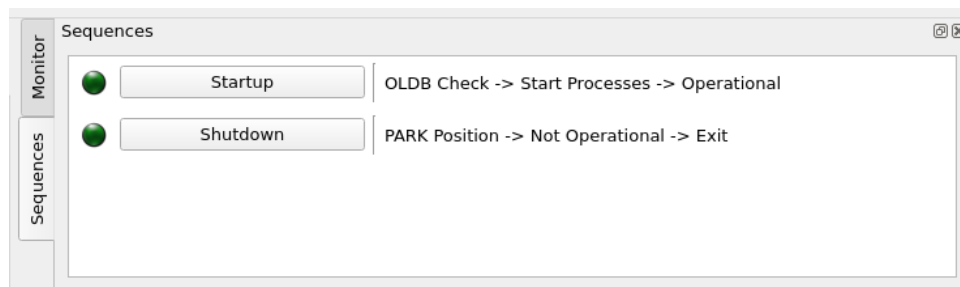


Figure 10

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 Manual Startup",  
    "command": "seqtool gui",  
    "sequence_file": "hlccseq/simStartup.py",  
    "description": "OLDB Check -> Start Processes -> Operational (fixed ports)"  
  },  
  {  
    "name": "DEV Manual Shutdown",  
    "command": "seqtool gui",  
    "sequence_file": "hlccseq/simShutdown.py",  
    "description": "PARK Position -> Not Operational -> Exit (fixed ports)"  
  },  
  {  
    "name": "DEV Manual ..."  
  }  
]
```

Figure 11

#### 6.5.9.4 Telescope Global State configuration:

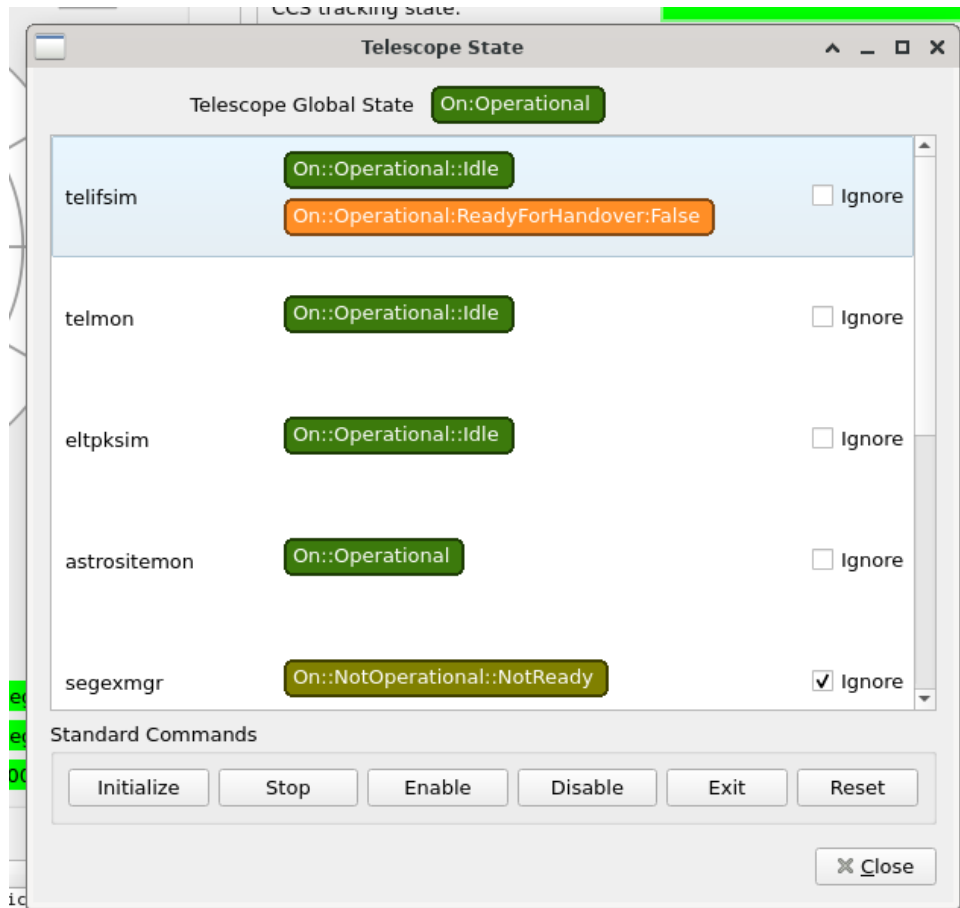


Figure 12

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

```
cii.olddb:///elt/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 store in the OLDB point.

## 6.6 hlcctelsimui\_mon documentation

This is an HLCC Simulator small monitoring UI, to be used as long as the main hlcctelsim\_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 telifsim, telmon and eltpksim rad applications
- Monitoring target and current positions as in the OLDB
- Monitoring a few flags

To start the application:

```
$ hlcctelsimui_mon
```

The screenshot shows a window titled 'Form@elthlccd23' with three main sections: General, States, and ROUS. Each section contains a list of parameters with their corresponding values displayed in green bars.

General	
Time LST:	3.2332 rad
	185.248 deg
Ready for handover:	True
Command source:	NONE
North angle:	-1.5021 rad
	-86.061 deg
Parallactic angle:	-2.0272 rad
	-116.150 deg
Elevation direction angle:	0.5251 rad
	30.088 deg
Version:	1.0.0-beta5

States	
TelIfSim state:	On::Operational::Idle On::Operational::ReadyForHandover:True
TelMon state:	On::Operational::Idle
EltpkSim state:	On::Operational::Tracking
CCS state:	On::Operational
CCS tracking state:	True

ROUS	
Rous disabled status:	False
Time to Rous maneuver:	299.0000 s
Rous maneuver in progress:	False

Figure 13

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.



## 6.7 telifsim documentation

This is the main entry point application to interact with the simulator.

All commands defined in the CCS-INS ICD are sent here.

### 6.7.1 Sending commands

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

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

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

```
$> msgsend --uri `consulGetUri -r ifuri -s telifsim -i Commands`  
::ccsinsif::Commands::RequestControl "CASCADE"
```

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

```
$> msgsend --uri `consulGetUri -r ifuri -s telifsim -i SimCmds`  
::telifsimif::SimCmds::MoveToNamedPos "PARK"  
  
$> msgsend --uri `consulGetUri -r ifuri -s telifsim -i SimCmds`  
::telifsimif::SimCmds::MoveToAltAzPos '{"alt": 0.5, "az": 0.2}'  
  
$> msgsend --uri `consulGetUri -r ifuri -s telifsim -i AppCmds`  
::appif::AppCmds::GetConfig "cfg/pub/dds/profile  
cfg/rous/timer_period_s"  
  
$> msgsend --uri `consulGetUri -r ifuri -s telifsim -i AppCmds`  
::appif::AppCmds::SetConfig "cfg/rous/timer_period_s: 123"
```

Because we are using Nomad/Consul to manage the deployment the uri to be used is retrieved with a query to consul using the `consulGetUri` command line utility, as seen in the examples above.

For example:

```
$> consulGetUri -r ifuri -s telifsim -i StdCmds
```

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 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 **telifsim\_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](#).

## 6.7.2 telifsim State Machine

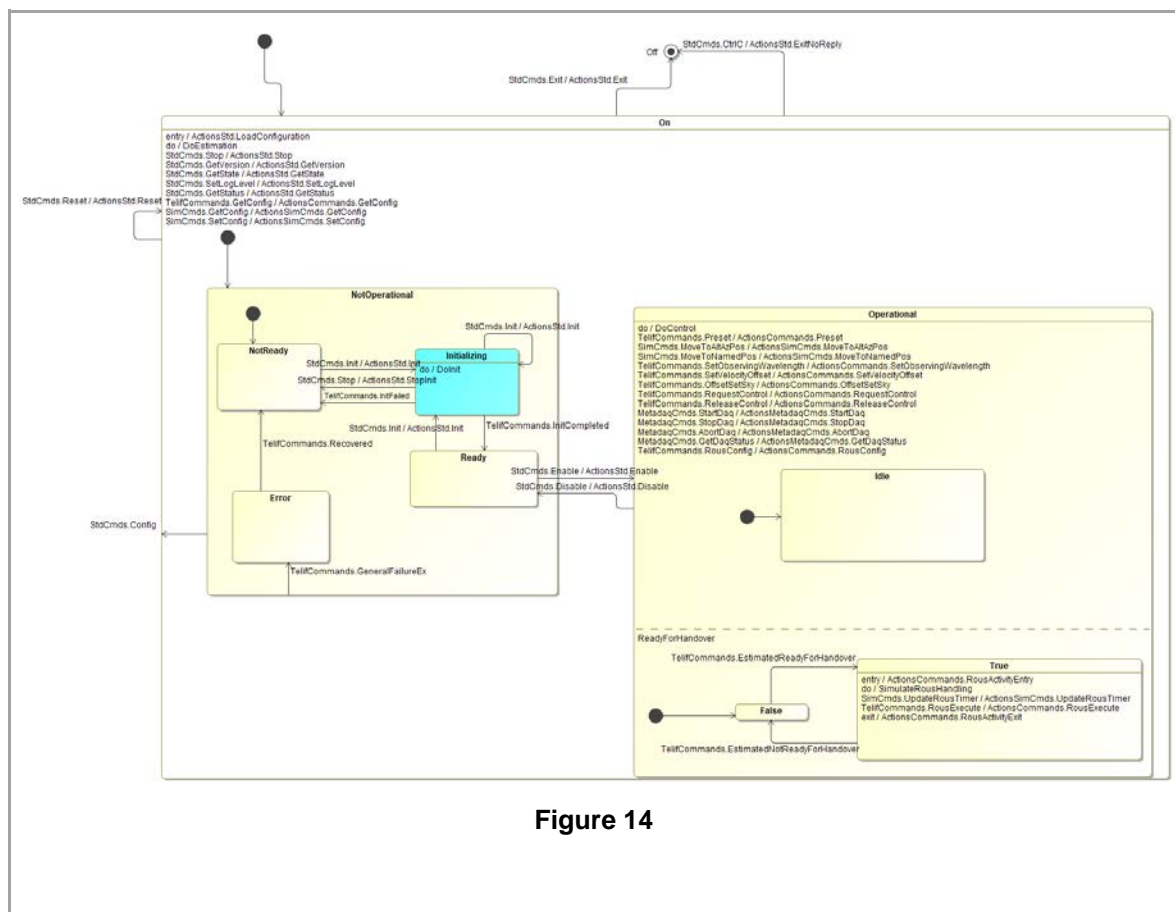


Figure 14



## 6.7.3 Rous Documentation

### 6.7.3.1 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 $(consulGetUri -r ifuri -s telifsim -i Commands)
::ccsinsif::Commands::RousConfig ' "ENABLE" '
```
  - `RousExecute`  
command that executes a ROUS maneuver immediately.  
==> This resets the counter  
For example:  

```
$ msgsend --uri $(consulGetUri -r ifuri -s telifsim -i 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

### 6.7.3.2 Rous Operation

By default Rous is enabled and as soon the telescope start tracking, Rous start 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 hlccsimui 'Ccsinsif' tab. See Figure 15 below:

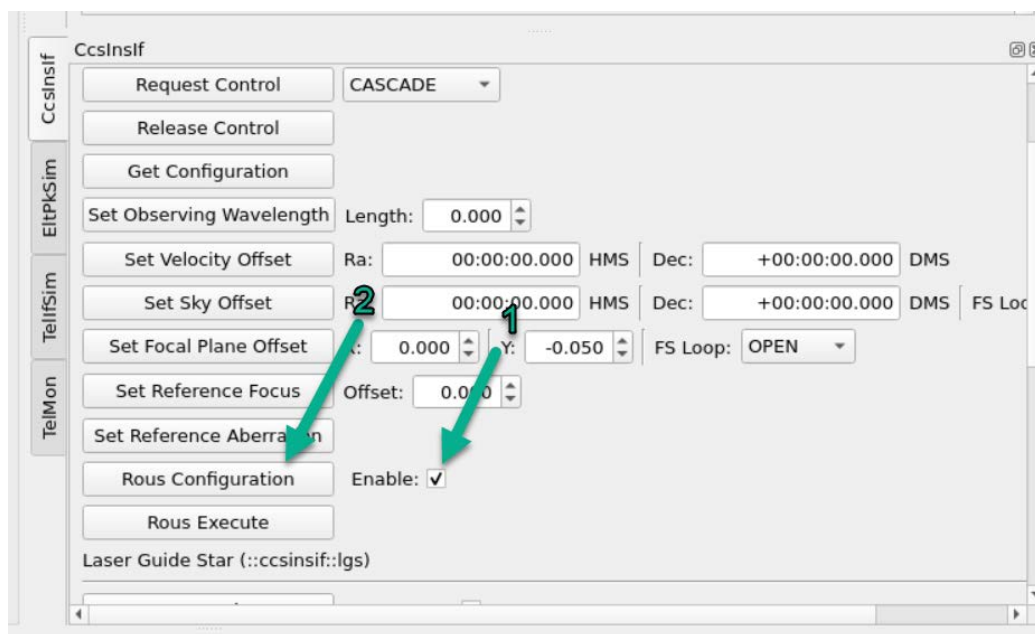


Figure 15

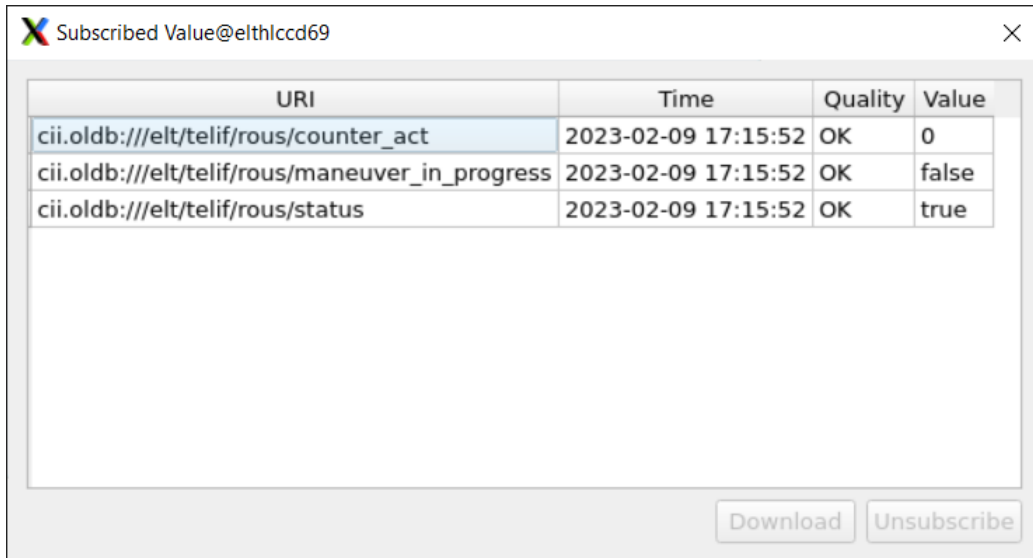
Set first the Rous status in the '**Enable**' box and then press '**Rous Configuration**', only when the button is pressed the command is sent to telifsim to change state.

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 16 below.



URI	Time	Quality	Value
cii.olddb:///elt/telif/rous/counter_act	2023-02-09 17:15:52	OK	0
cii.olddb:///elt/telif/rous/maneuver_in_progress	2023-02-09 17:15:52	OK	false
cii.olddb:///elt/telif/rous/status	2023-02-09 17:15:52	OK	true

Figure 16

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

#### 6.7.4 Preset Documentation

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

This is to explore the various possibilities and foster discussion with the users to converge toward an agreed strategy. Check below the available preset modes.

##### 6.7.4.1 NO\_SEQUENCE mode

The Preset sequence is hardcoded in a CPP preset function, very basic and not interactive. This will be phased out once agreed on all details and the procedures will be fully defined.

##### 6.7.4.2 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.



#### 6.7.4.3 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 telifsim:

- 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.

#### 6.7.4.4 Preset Sequences

Currently we have two preset sequences available as initial templates for discussion:

- *Hlccseq.hlccPreset* – This sequence is built to run automatically without any user intervention. It can be used in 'HEADLESS' or 'GUI' modes. This sequence is basically the same implementation of the 'NO\_SEQUENCE' mode, but done in a sequence instead of hardcoded in c++.
- *Hlccseq.hlccPresetInteractive* – This sequence should only be configured to run in 'GUI' mode because it will request user interaction and a Sequencer Gui shall be available otherwise the sequence will hang indefinitely. It implements the same behavior of *hlccPreset* but pops up a dialog at the beginning asking for user input.

#### 6.7.4.5 Sequence and mode configuration

To define how the Preset sequence 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 telifsim commands.

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



- HEADLESS
- GUI
- NO\_SEQUENCE
- Telifsim command:
  - `msgsend --uri `consulGetUri -r ifuri -s telifsim -i SimCmds`  
::telifsimif::SimCmds::SetPresetSequenceMode "GUI"`
- Sequence script
  - Oldb Address:
    - `cii.oldb:///elt/hlcc/telifsim/preset/sequence`
  - Default sequence
    - `hlccseq.hlccPreset`
  - Telifsim Command
    - `msgsend --uri `consulGetUri -r ifuri -s telifsim -i SimCmds`  
::telifsimif::SimCmds::SetPresetSequence "hlccseq.hlccPreset"`

## 6.8 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.

### 6.8.1 Sending commands

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

```
$> msgsend --uri `consulGetUri -r ifuri -s telmon -i StdCmds`  
::stdif::StdCmds::GetStatus
```

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

```
$> msgsend --uri `consulGetUri -r ifuri -s telmon -i AppCmds`  
::appif::AppCmds::GetConfig "cfg/estimation_period_ms"
```

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

```
$> msgsend --uri `consulGetUri -r ifuri -s telmon -i 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.

For example:

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



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 `consulGetUri -r ifuri -s telmon -i MonCmds`  
::telmonif::MonCmds::SetAppIgnore '{"app_name": "telifsim", "ignore":  
false}'
```

to retrieve a configuration parameter:

```
$> msgsend --uri `consulGetUri -r ifuri -s telmon -i AppCmds`  
::appif::AppCmds::GetConfig "cfg/estimation_period_ms"  
  
REPLY: "cfg: \n estimation_period_ms: 1000\n"
```

to set a configuration parameter:

```
$> msgsend --uri `consulGetUri -r ifuri -s telmon -i AppCmds`  
::appif::AppCmds::SetConfig "cfg/estimation_period_ms: 1500"  
  
REPLY: "OK, Updated Parameters: cfg/estimation_period_ms"
```

to load configuration from file:

```
$> msgsend --uri `consulGetUri -r ifuri -s telmon -i AppCmds`  
::appif::AppCmds::LoadConfig "config/telmon/config.yaml"
```

## 6.8.2 telmon State Machine

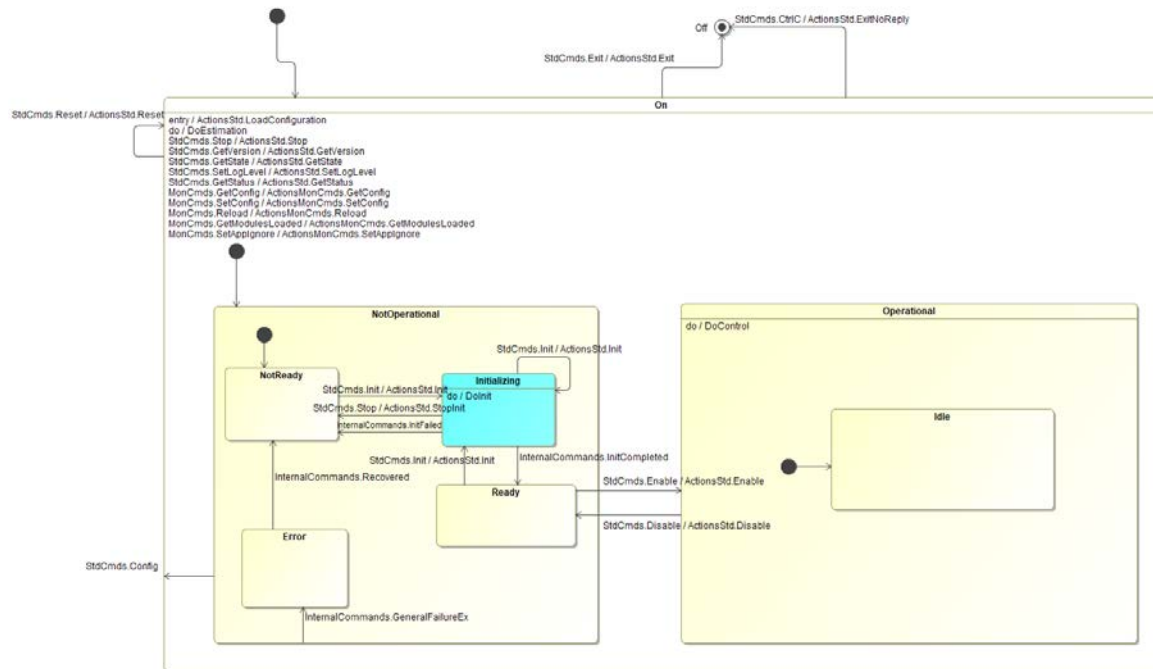


Figure 17

## 6.8.3 Adding new estimation scripts to Telmon

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

### 6.8.3.1 Telmon directory structure

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

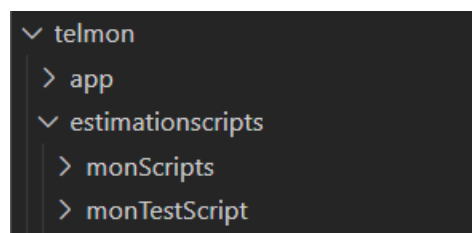


Figure 18

- 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.

#### 6.8.3.2 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.

#### 6.8.4 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 19

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



### 6.8.5 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:12084/telmon/MonCmds ::telmonif::MonCmds::Reload
```

### 6.8.6 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.

### 6.8.7 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.

#### 6.8.7.1 Ccs state estimation

This estimation script estimates the global ccs state by combing the state of all applications listed in the Oldb uri '*cii.ldb:///elt/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.ldb:///elt/telif/ccs/state*'. It is also being published in dds '*dds.ps:///TELIF\_STATUS*'.

It can be found in the telescope simulator monitor panel:

CCS state:

Off

#### 6.8.7.2 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.ldb:///elt/telif/ccs/ready\_for\_handover*'. It is also being published in dds '*dds.ps:///TELIF\_RDY\_FOR\_HANOVER*'.

It can be found in the telescope simulator monitor panel:



Ready for handover:

False

### 6.8.7.3 Segment exchange mode estimation

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

It can be found in the telescope simulator monitor panel:

Segment exchange mode:

False

### 6.8.7.4 Tracking estimation

Estimate the tracking state of the telescope and publishes the result in `'cii.olddb:///elt/hlcc/telmon/mon/tracking'`.

It can be found in the telescope simulator monitor panel:

CCS tracking state:

False

### 6.8.7.5 Night time estimation

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

It can be found in the telescope simulator monitor panel:

Night Time Operation:

False

## 6.9 eltpksim documentation

This is the pointing kernel simulator, emulating telescope tracking.

### 6.9.1 Sending commands

To send a standard command using the msgsend utility:

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

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

```
$> msgsend -uri `consulGetUri -r ifuri -s eltpksim -i Commands`  
::eltpkif::PointingKernelCommands::SetTargetAltPos `0.2`
```



```
$> msgsend -uri `consulGetUri -r ifuri -s eltpksim -i Commands`  
::eltpkif::PointingKernelCommands::SetTargetAzPos `0.3`
```

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

```
$> msgsend -uri `consulGetUri -r ifuri -s eltpksim -i AppCmds`  
::appif::AppCmds::GetConfig `"cfg/procname"`  
  
REPLY: "cfg: \n  procname: \"eltpksim\" \n"  
  
$> msgsend -uri `consulGetUri -r ifuri -s eltpksim -i AppCmds`  
::appif::AppCmds::SetConfig `"cfg/params/alt_speed_deg_per_s: 7.5"`  
  
REPLY: "OK, Updated Parameters: cfg/params/alt_speed_deg_per_s"
```

Because we are using Nomad/Consul to manage the deployment, the uri to be used is retrieved with a query to consul using the consulGetUri command line utility, as seen in the examples above.

For example:

```
$> consulGetUri -r ifuri -s eltpksim -i StdCmds
```

The standard commands supported are described in this ICD:

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

The specific commands supported are described in this ICD:

- [telif/eltpkif/icd/src/eltpkif.xml · master · ccs / hlcc · GitLab \(eso.org\)](#)

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

```
{\"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\": [] }}
```

Instead, for most uses we recommend to use the python utility `eltpksim_preset` (see section 6.13.2), 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](#)

## 6.9.2 eltpksim State Machine





- 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.ldb:///elt/hlcc/eltpksim/cfg/trs_health_enabled`
  - Actual health: `cii.ldb:///elt/hlcc/eltpksim/mon/trs/health`
  - Reason for health state: `cii.ldb:///elt/hlcc/eltpksim/mon/trs/reason`

## 6.10 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 Amazonas Site Monitor. The following examples will be based on this Lsvsim instance.

### 6.10.1 Sending Commands

**ATTENTION: At the moment the endpoint registration for Lsvsim applications is not consistent with the other HLCC applications, i.e. the endpoint does not contain the process name but only the interface. Therefore the `msgsend` command lines using `consulGetUri` are constructed in a different way.**

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 `consulGetUri -r uri -s astrositemon`/StdCmds  
::stdif::StdCmds::GetState
```

Lsvsim app implements the commands 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 `consulGetUri -r uri -s 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`:



#### 6.10.1.1.1 SetConfig

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

```
$> msgsend --uri `consulGetUri -r uri -s astrositemon`/AppCmds  
::appif::AppCmds::SetConfig "cfg/sim_activity_period_ms : 120"
```

#### 6.10.1.1.2 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 `consulGetUri -r uri -s astrositemon`/AppCmds  
::appif::AppCmds::GetConfig "cfg/sim_activity_period_ms"
```

#### 6.10.1.1.3 LoadConfig

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

```
$> msgsend --uri `consulGetUri -r uri -s astrositemon`/AppCmds  
::appif::AppCmds::LoadConfig "config/lsvsim/config_ml.yaml"
```

#### 6.10.1.1.4 LoadStateMachine

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

```
$> msgsend --uri `consulGetUri -r uri -s astrositemon`/AppCmds  
::appif::AppCmds::LoadStateMachine '{"main" :  
"config/lsvsim/sm.xml", "append" : "config/lsvsim/sm_append.xml"}'
```

#### 6.10.1.2 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:

##### 6.10.1.2.1 SetSim

Writes, in configuration parameter "cfg.sv\_specific\_conf", 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 `consulGetUri -r uri -s astrositemon`/SimCmds  
::lsvsimif::SimCmds::SetSim "svdoubles.sv_estimation_com: True"
```

#### 6.10.2 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\\_ml.yaml.in · master · ccs / hlcc · GitLab \(eso.org\)](#)





```
5  cfg:
6      modname      : "m1sv"
7      procname     : "m1sv_"
8      log_level    : "INFO"
9      log_properties : "config/lsvsim/log.properties"
10     sm_scxml      : "config/lsvsim/sm_radapp default.xml" # SCXML state machine model
11     sm_scxml_append : "config/lsvsim/sm.xml"
12     req_endpoint  : "zpb.rr://@ADDRESS_ZMQ@/" # IP address and port used to accept requests
13     oldb_uri_prefix : "cii.oldb:///elt/tel/m1/lsv/" # CII OLD prefix
14
15     pub:
16         dds:
17             profile : Localhost_Only
18             #profile_file : # ignored if profile not set. default: config/hlcc/dds/hlccDdsQosProfiles.xml
19     sv_list : lcfg.type:vector_SvDescription
20             - sv_name: svdoubles
21               sv_type: hlcc.SimScripts.sv_doubles_estimation
22             - sv_name: svstrings
23               sv_type: hlcc.SimScripts.sv_strings_estimation
24             - sv_name: svvectors
25               sv_type: hlcc.SimScripts.sv_vectors_estimation
26             - sv_name: svtmpctrlr
27               sv_type: hlcc.SimScripts.sv_temperature_controller
28
29     sim_activity_period_ms : 100
30     sv_specific_config : | # this is a string containing a 2nd level yaml specific for each simulation
31                         svdoubles:
32                             sv_entity      : ModLsvsimif.Lsvsimif.DoublesSv
33                             sv_pub_endpoint : dds.ps:///m1/doubles_sv
34                             sv_pub_update_period_ms : 100
35                             sv_oldb_update_period_ms : 1000
36                             sv_oldb_uris :
37                                 state : cii.oldb:///elt/tel/m1/svdoubles/state
38                                 quality : cii.oldb:///elt/tel/m1/svdoubles/quality
39                                 age : cii.oldb:///elt/tel/m1/svdoubles/age
40                                 sample1 : cii.oldb:///elt/tel/m1/svdoubles/sample1
41                                 sample2 : cii.oldb:///elt/tel/m1/svdoubles/sample2
42                                 sample3 : cii.oldb:///elt/tel/m1/svdoubles/sample3
43                                 sample4 : cii.oldb:///elt/tel/m1/svdoubles/sample4
44                                 sample5 : cii.oldb:///elt/tel/m1/svdoubles/sample5
45                                 sample6 : cii.oldb:///elt/tel/m1/svdoubles/sample6
46                                 sample7 : cii.oldb:///elt/tel/m1/svdoubles/sample7
47                                 sample8 : cii.oldb:///elt/tel/m1/svdoubles/sample8
48                                 sample9 : cii.oldb:///elt/tel/m1/svdoubles/sample9
49                                 sample10 : cii.oldb:///elt/tel/m1/svdoubles/sample10
50                             sv_estimation_com : False
51                             sv_estimation_remote : False
52                             sv_estimation_poweron : False
53                             sv_estimation_initialized : False
54                             sv_estimation_closedloop : False
55                             sv_estimation_moving : False
56                         svstrings:
57                             sv_entity      : ModLsvsimif.Lsvsimif.StringsSv
58                             sv_pub_endpoint : dds.ps:///m1/strings_sv
59                             sv_pub_update_period_ms : 100
60                             sv_oldb_update_period_ms : 1000
61                             sv_oldb_uris :
62                                 state : cii.oldb:///elt/tel/m1/svstrings/state
63                                 quality : cii.oldb:///elt/tel/m1/svstrings/quality
64                                 age : cii.oldb:///elt/tel/m1/svstrings/age
65                                 sample1 : cii.oldb:///elt/tel/m1/svstrings/sample1
66                                 sample2 : cii.oldb:///elt/tel/m1/svstrings/sample2
67                                 sample3 : cii.oldb:///elt/tel/m1/svstrings/sample3
68                                 sample4 : cii.oldb:///elt/tel/m1/svstrings/sample4
69                                 sample5 : cii.oldb:///elt/tel/m1/svstrings/sample5
70                                 sample6 : cii.oldb:///elt/tel/m1/svstrings/sample6
71                                 sample7 : cii.oldb:///elt/tel/m1/svstrings/sample7
72                                 sample8 : cii.oldb:///elt/tel/m1/svstrings/sample8
73                                 sample9 : cii.oldb:///elt/tel/m1/svstrings/sample9
74                                 sample10 : cii.oldb:///elt/tel/m1/svstrings/sample10
75                             sv_estimation_com : False
76                             sv_estimation_remote : False
77                             sv_estimation_poweron : False
78                             sv_estimation_initialized : False
79                             sv_estimation_closedloop : False
80                             sv_estimation_moving : False
81
```

Figure 21





**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 (2<sup>nd</sup> 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.

### 6.10.3 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](#).

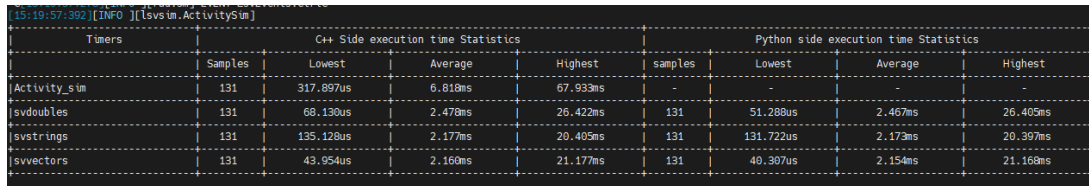
### 6.10.4 Code Execution Statistics

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:



Timers	C++ Side execution time Statistics				Python side execution time Statistics			
	Samples	Lowest	Average	Highest	samples	Lowest	Average	Highest
Activity_sim	131	317.897us	6.818ms	67.933ms	-	-	-	-
svdoubles	131	68.130us	2.478ms	26.422ms	131	51.288us	2.467ms	26.405ms
svstrings	131	135.128us	2.177ms	20.405ms	131	131.722us	2.173ms	20.397ms
svvectors	131	43.954us	2.160ms	21.177ms	131	40.307us	2.154ms	21.168ms

Figure 22

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.



```
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:

```
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() }
```

### 6.10.5 PID Temperature Controller example

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

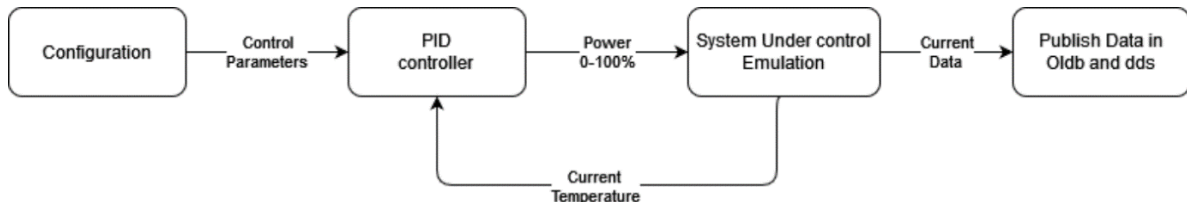
The idea is to have the PID algorithm to reach the predefined temperature on the (emulated) system under control.

Implemented as a consequence of this ticket: [\[ETCS-1146\] lvsim— implement PID based state variable example— ESO JIRA Projects](#)

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

[lvsim/simulationscripts/simScripts/src/hlcc/SimScripts/sv\\_temperature\\_controller.py · master · ccs / hlcc · GitLab \(eso.org\)](#)

Figure 23 below shows a simplified diagram of the main blocks for the process:

**Figure 23**

- 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 [lsvsim/simulationscripts/simScripts/src/hlcc/SimScripts/PID.py · master · ccs / hlcc · GitLab \(eso.org\)](#) 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.

#### 6.10.5.1 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_ml.yaml
```

#### 6.10.5.2 Sending commands

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

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

```
$ msgsend --uri zpb.rr://localhost:12090/SimCmds  
::lsvsimif::SimCmds::SetSim "svtmpctrlr.controller_poweron: True"
```

To tune the PID parameters use the following commands:



```
$ msgsend --uri zpb.rr://localhost:12090/SimCmds
::lsvsimif::SimCmds::SetSim '"svtmpctrlr.target_temperature_c: 50"'

$ msgsend --uri zpb.rr://localhost:12090/SimCmds
::lsvsimif::SimCmds::SetSim '"svtmpctrlr.pid_kp: 10"'

$ msgsend --uri zpb.rr://localhost:12090/SimCmds
::lsvsimif::SimCmds::SetSim '"svtmpctrlr.pid_ki: 3.5"'

$ msgsend --uri zpb.rr://localhost:12090/SimCmds
::lsvsimif::SimCmds::SetSim '"svtmpctrlr.pid_kd: 0.4"'
```

To subscribe to dds use `telifsim_subscriber`:

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

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

```
$ cii.olddb:///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.

## 6.11 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.

### 6.11.1 Sending commands

This process supports all the commands implemented in `lsvsim` application.

## 6.12 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 currently providing the following features:



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

### 6.12.1 Sending commands

This process supports all the commands implemented in lsvsim application.

### 6.12.2 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.

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

```
$> msgsend --uri zpb.rr://localhost:12091/SimCmds ::lsvsimif::SimCmds::SetSim  
'"astrositemon.site_dps.air_temperature.value: 120"'
```

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

```
$> msgsend --uri `consulGetUri -r uri -s 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

## 6.13 HLCC utilities

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

### 6.13.1 \$ 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  
Interface 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



### 6.13.2 \$ eltpksim\_preset --help

usage: eltpksim\_preset [-h] [-r RA] [-d DEC] [--init] [--test]

Sends a Preset command to eltpksim process (passing no arguments == --test)

optional arguments:

- h, --help show this help message and exit
- r RA, --ra RA RA of target (rad)
- d DEC, --dec DEC DEC of target (rad)
- init Initializes and Enable before sending the Preset command
- test Initialize, Enable and Preset to a predefined test set of coordinates
- n, --nomad Queries nomad for the current IP:PORT of applications

Example:

```
eltpksim_preset -r 5.54 -d -1.55
```

### 6.13.3 \$ telifsim\_preset --help

Sends a Preset command to telifsim process (passing no arguments == use defaults). Default values means that the telescope will always preset to a Ra/Dec that corresponds to an Alt/Az of 89.5deg/10deg at time of command execution.

usage: telifsim\_preset [-h] [--ra RA] [--dec DEC] [--gs1] [--gs1-ra GS1\_RA]  
                      [--gs1-dec GS1\_DEC] [--gs2] [--gs2-ra GS2\_RA]  
                      [--gs2-dec GS2\_DEC]

Sends a Preset command to the telifsim process

optional arguments:

- h, --help show this help message and exit
- r RA, --ra RA RA of target (rad)
- d DEC, --dec DEC DEC of target (rad)
- gs1 Use guidestar 1
- gs1-ra GS1\_RA RA of guidestar 1 (rad)





--gs1-dec GS1\_DEC DEC of guidestar 1 (rad)  
--gs2 Use guidestar 2  
--gs2-ra GS2\_RA RA of guidestar 2 (rad)  
--gs2-dec GS2\_DEC DEC of guidestar 2 (rad)  
--version show program's version number and exit  
--init Initializes and Enable before sending the Preset command  
--test Initialize, Enable and Preset to a predefined test set of coordinates  
-n, --nomad Queries nomad for the current IP:PORT of applications

Example:

```
telifsim_preset -r 0.9 -d 0.7
```

#### 6.13.4 \$ telif\_subscriber --help

usage: telif\_subscriber [-h] [--uri URI] [--entity ENTITY]  
                        [--endafter ENDAFTER] [--endifset ENDIFSET]  
                        [--verbose] [--version]

Subscribes to PubSub

optional arguments:

-h, --help show this help message and exit  
--uri URI publisher address (as URI), e.g.  
          "zpb.ps://localhost:12781/TELIF\_CCS\_TRG\_OBS\_ALTAZ"  
--entity ENTITY entity classname (as FQN), e.g.  
          "ModCcsinsif.Ccsinsif.AltAz"  
--endafter ENDAFTER max wait time (in seconds)  
--endifset ENDIFSET wait for content (as dict), e.g. '{"alt':1.0,'az':2.0}"  
--verbose print all received data (withuot that, received data would not be printed)  
--version show program's version number and exit

**Example 1: subscribing to the publisher example below (with dds and zpb)**



```
$ telif_subscriber \  
  --uri dds.ps:///TELIF_CCS_TRG_OBS_ALTAZ \  
  --entity ModCcsinsif.Ccsinsif.AltAz \  
  --endifset '{"alt":3.0, "az":9.0}' \  
  --endafter 10 \  
  --verbose  
recd {"alt": 1.0, "az": 9.0}  
recd {"alt": 2.0, "az": 9.0}  
recd {"alt": 3.0, "az": 9.0}
```

```
$ telif_subscriber \  
  --uri zpb.ps://localhost:12781/TELIF_CCS_TRG_OBS_ALTAZ \  
  --entity ModCcsinsif.Ccsinsif.AltAz \  
  --endifset '{"alt":3.0, "az":9.0}' \  
  --endafter 10 \  
  --verbose  
recd {"alt": 1.0, "az": 9.0}  
recd {"alt": 2.0, "az": 9.0}  
recd {"alt": 3.0, "az": 9.0}
```

Notice that with zpb the first sample is actually lost (this is a known zpb effect related with late joiner implementation)

```
$ echo $?  
0
```

## Example 2: subscribing to the PK-simulator's position stream

```
$ telif_subscriber \  
  --uri mudpi.ps://224.0.0.1:12783/TELIF_CCS_PK_POS \  
  --entity ModCcsinsdetif.Ccsinsif.PointingKernelPositions \  
  --verbose
```



Note: The entity data is too complex to be processed as JSON

Note: To boost the JSON processing, do: `pip install jsons`

Note: Falling back to plain attribute format

```
recd current_observed_altaz:SharedVectorConstDouble[0, 0] | elevation_direction_angle:0.0 |
north_angle:0.0 | observed_altaz_at_requested_xy:SharedVectorConstDouble[0, 0] |
parallactic_angle:0.0 | pupil_angle:0.0 |
radec_at_xy_from_guide_stars:SharedVectorConstDouble[0, 0] |
target_observed_altaz:SharedVectorConstDouble[0, 0] | time_lst:1.598577744817963 |
time_tai:1659448989.1 | time_utc:1659448952.1
```

### Example 3: subscribing to the PK-simulator's status message

```
$ telif_subscriber \
  --uri dds.ps:///TELIF_STATUS \
  --entity ModStdif.Stdif.Status \
  --endafter 120 \
  --verbose
recd {"status": "On::NotOperational::NotReady "}
```

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

#### 6.13.5 \$ telif\_publisher --help

```
usage: telif_publisher [-h] [--uri URI] [--entity ENTITY] [--set SET]
                        [--interval INTERVAL] [--iterations ITERATIONS]
                        [--verbose] [--version]
```

Publishes to PubSub

optional arguments:

```
-h, --help          show this help message and exit
--uri URI           publisher address, e.g.
                    "zpb.ps://localhost:12781/TELIF_CCS_TRG_OBS_ALTAZ".
                    The host name will be ignored.
--entity ENTITY     entity classname, e.g. "ModCcsinsif.Ccsinsif.AltAz"
```



--set SET            data content (as dict), e.g. '{"alt":1, "az":1}'. If  
                      specified multiple times, contents get accumulated.  
--interval INTERVAL wait time between publications (in seconds)  
--iterations ITERATIONS  
                      number of iterations  
--verbose            print all published data  
--version            show program's version number and exit

**Example (dds and zpb):**

```
$ telif_publisher \  
  --uri dds.ps:///TELIF_CCS_TRG_OBS_ALTAZ \  
  --entity ModCcsinsif.Ccsinsif.AltAz \  
  --set '{"alt":1.0, "az":9.0}' \  
  --set '{"alt":2.0}' \  
  --set '{"alt":3.0}' \  
  --verbose  
sent {"alt": 1.0, "az": 9.0}  
sent {"alt": 2.0, "az": 9.0}  
sent {"alt": 3.0, "az": 9.0}  
  
$ telif_publisher \  
  --uri zpb.ps://localhost:12781/TELIF_CCS_TRG_OBS_ALTAZ \  
  --entity ModCcsinsif.Ccsinsif.AltAz \  
  --set '{"alt":1.0, "az":9.0}' \  
  --set '{"alt":2.0}' \  
  --set '{"alt":3.0}' \  
  --verbose  
sent {"alt": 1.0, "az": 9.0}  
sent {"alt": 2.0, "az": 9.0}  
sent {"alt": 3.0, "az": 9.0}
```



### 6.13.6 \$ telifsim\_get\_config --help

usage: telifsim\_get\_config [-h] [--notest] [--version]

Sends an GetConfig command to telifsim

optional arguments:

- h, --help show this help message and exit
- notest Sends a Getconfig command and print response on console without testing
- version show program's version number and exit

### 6.13.7 \$ telifsim\_metadaqcmds --help

usage: telifsim\_metadaqcmds [-h] [-o {start,stop,abort,get\_status}] [-i ID] [--version] [--test]

Sends a Preset Daq commands to telifsim (passing no arguments == --test)

optional arguments:

- h, --help show this help message and exit
- o {start,stop,abort,get\_status}, --option {start,stop,abort,get\_status} Action to perform in the command
- i ID, --id ID Daq Id
- version show program's version number and exit
- test Runs the automated test routine
- n, --nomad Queries nomad for the current IP:PORT of applications

Meta data acquisition must happen while the telescope is tracking.

**Example - Starting/stopping the meta data acquisition with ID = 1 :**

```
telifsim_metadaqcmds -o start -i 1
```

```
Reply (DaqReply): Id '1'
```

```
telifsim_metadaqcmds -o stop -i 1
```

```
Reply (DaqStopReply): Id '1'
```

```
      : Files []
```

```
      : Keywords [{ "type": "esoKeyword", "name": "ESO TEL ALT", "value": 0.961 }, {
"type": "esoKeyword", "name": "ESO TEL AZ", "value": 0.348 }, { "type": "esoKeyword", "name": "ESO
TEL GEOELEV", "value": 3046.0 }, { "type": "esoKeyword", "name": "ESO TEL GEOLAT", "value": -
0.4292 }, { "type": "esoKeyword", "name": "ESO TEL GEOLON", "value": 1.2251 }, {
"type": "esoKeyword", "name": "ESO TEL TARG EPOCH", "value": 2000.000 }, { "type": "esoKeyword",
"name": "ESO TARG EPOCHSYSTEM", "value": "J" }, { "type": "esoKeyword", "name": "ESO TEL
TARG RADVEL", "value": 0.600 }, { "type": "esoKeyword", "name": "ESO TEL TARG PARALLAX",
"value": 0.500 }, { "type": "esoKeyword", "name": "ESO TEL TARG ALPHA", "value": 205930.590 }, {
"type": "esoKeyword", "name": "ESO TEL TARG DELTA", "value": -561702.970 }]
```

**6.13.8 \$ telifsim\_offset\_setsky --help**

```
usage: telifsim_offset_setsky [-h] [-r RA] [-d DEC] [-f {closed,open}]
                             [-g GUIDESTAR] [--version] [--test]
```

Sends an OffsetSetSky command to telifsim (passing no arguments == --test)

optional arguments:

- h, --help show this help message and exit
- r RA, --ra RA RA offset (rad)
- d DEC, --dec DEC DEC offset (rad)
- f {closed,open}, --field {closed,open} Field stabilization loop
- g GUIDESTAR, --guidestar GUIDESTAR guide star parameter
- version show program's version number and exit
- test Sends a pre-defined offset to telifsim.
- n, --nomad Queries nomad for the current IP:PORT of applications



### 6.13.9 \$ telifsim\_request\_release\_control --help

usage: telifsim\_request\_release\_control [-h] [-o {request,release}]  
[-m {cascade,sequential}] [--version]  
[--test]

Sends Request/Release command to telifsim (passing no arguments == --test)

optional arguments:

- h, --help show this help message and exit
- o {request,release}, --option {request,release}  
Action to perform in the command
- m {cascade,sequential}, --mode {cascade,sequential}  
Control Mode
- version show program's version number and exit
- test Sends a request / release control command
- n, --nomad Queries nomad for the current IP:PORT of applications

### 6.13.10 \$ telifsim\_rous --help

usage: telifsim\_rous [-h] [-o {config,execute,update\_timer}]  
[-m {enable,disable}] [-t TIME] [--version] [--test]

Sends Request/Release command to telifsim (passing no arguments == --test)

optional arguments:

- h, --help show this help message and exit
- o {config,execute,update\_timer}, --option {config,execute,update\_timer}  
Rous command to be executed
- m {enable,disable}, --mode {enable,disable}  
Rous config Mode
- t TIME, --time TIME Time in seconds to update rous timer
- version show program's version number and exit



--test                Executes the default Rous testroutine  
-n, --nomad           Queries nomad for the current IP:PORT of applications

## 6.14 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 yet available in DevEnv 4.x/ Fedora, See: [\[ELTDEV-1090\] Package jupyterlab - ESO JIRA Projects](#))

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
```

### 6.14.1 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:



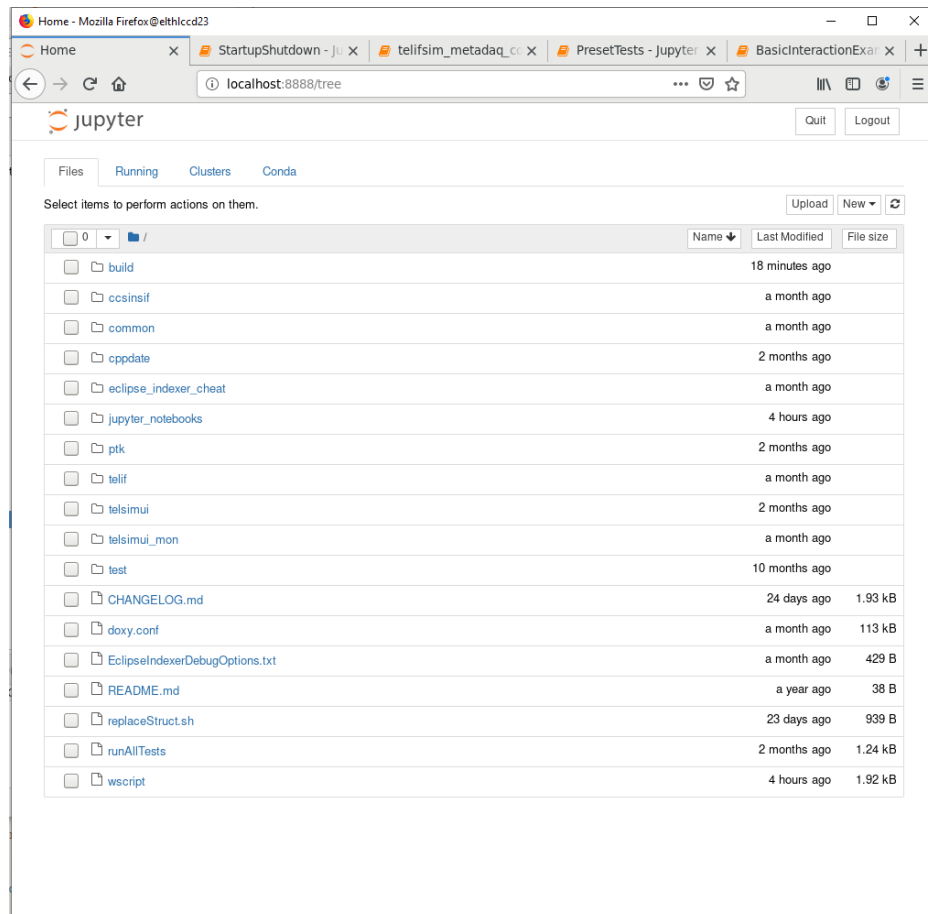


Figure 24

### 6.14.2 Start jupyter server and browser client on the 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 starts 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 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.

### 6.14.3 Executing Jupyter Notebooks

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

For example:

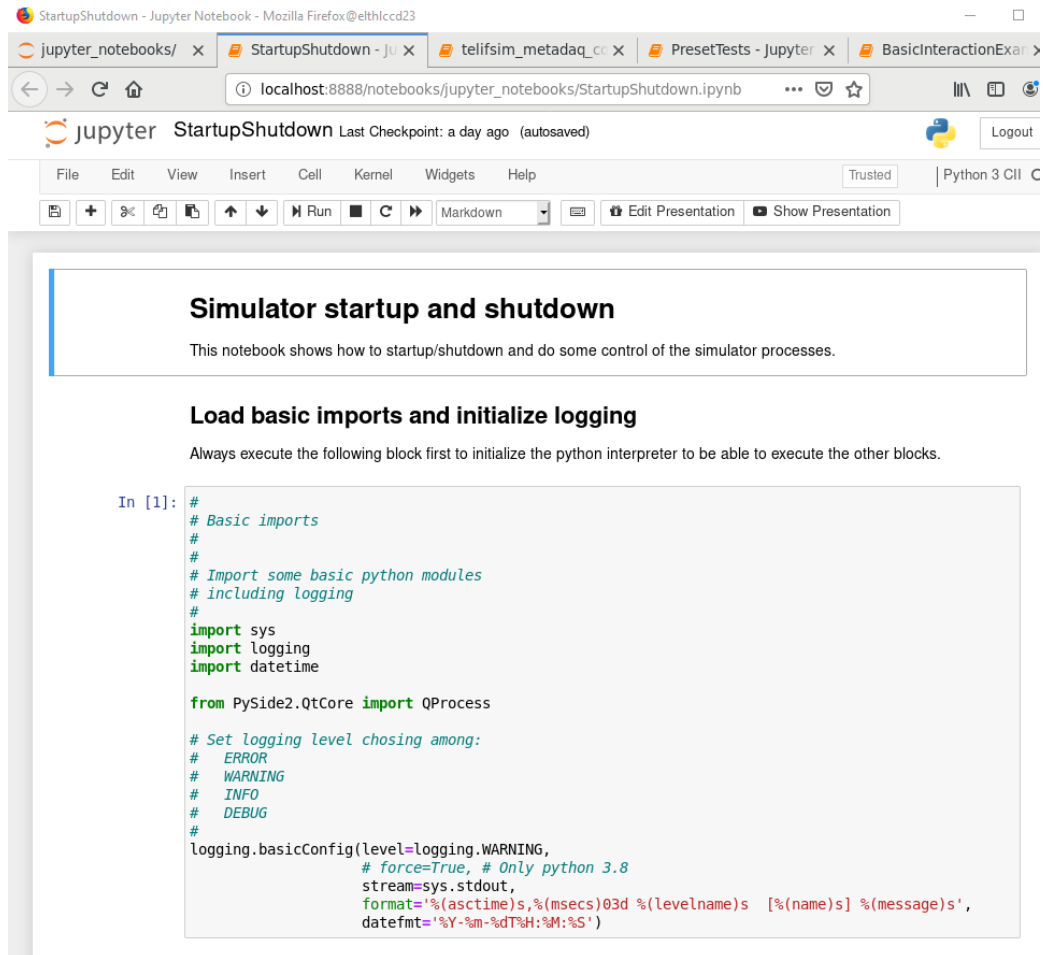


Figure 25

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.



## 6.15 Sequencer scripts

The code repository includes a set of [ELT Sequencer scripts](#):

The Sequencer scripts are in this folder:

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

and are installed as the `hlccseq` python module.

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

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

### 6.15.1 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
```

### 6.15.2 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`
```

Note: the configuration file

```
seqgui_config.yaml
```

should exist in the home directory, otherwise the sequencer will fail with an error message.

The typical content of this file is:

```
seq_server:
```



```
url: localhost:8000
loglevel: info
logs:
  url: localhost:8113
otto:
  url: http://127.0.0.1:5000/
  insid: FORS2
  mode: VM
```

If the file does not exist you can create it with the contents above or you can take it from the seq source tree in:

```
seq/gui/src/seq/gui/seqgui_config.yaml
```

The sequencer uses a default if that does not exist.

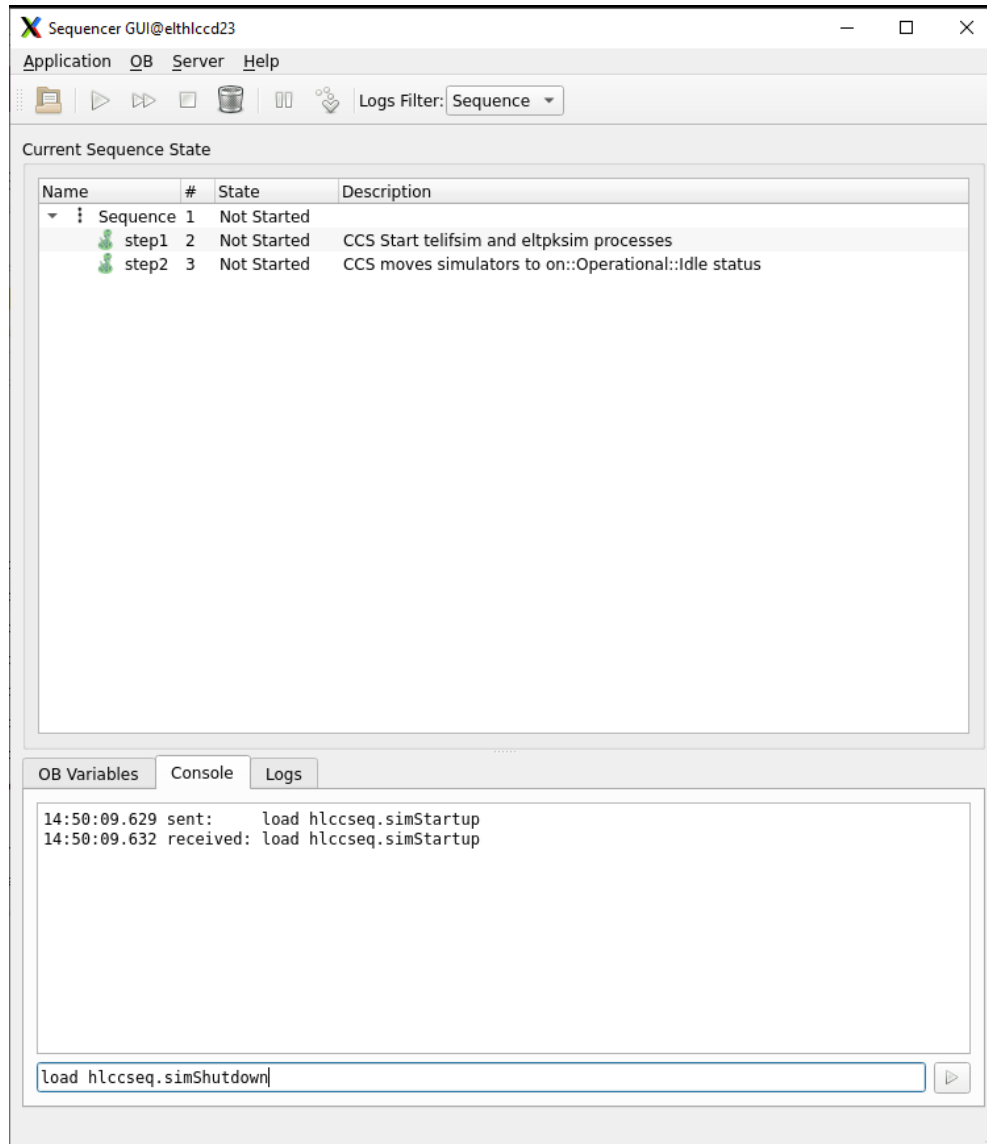
Once the gui is started, there are two ways to load and run sequences from the sequencer gui:

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:
  - o activate the debug mode from the Application->Debug mode menu
  - o issue in the Console tab the commands to load specific sequences, like

```
load hlcseq.simStartup
load hlcseq.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 26**

For more details look at the Sequencer documentation in [ELT Sequencer scripts](#)

### 6.15.3 Known Issues

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

None listed here at the moment



#### 6.15.4 ToDo

None listed here at the moment

### 6.16 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 eltpksim 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.oidb:///elt/telif/site/info` (public CCS-INS interface)

`cii.oidb:///elt/hlcc/eltpksim/cfg/site/info/elevation` (eltpksim private OLDB)

#### 6.16.1 Change telescope location in configuration files

- The configuration is stored in the file [telif/eltpksim/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

#### 6.16.2 Change telescope location with configuration commands

- The procedure is documented in details in the Jupyter notebook [jupyter notebooks/ChangeTelescopeSite.ipynb](#)



- In summary:
  - The site location configuration keywords can be changed only when the `eltpksim` 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

## 6.17 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.



### 6.17.1 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.





### 6.17.2 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.

### 6.17.3 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.

## 7. 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.



## 8. 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.

### 8.1 telif documentation

#### 8.1.1 Start the application:

```
$> telif -c config/telif/config.yaml -lTRACE
```

where:

-c argument is the path to the configuration file, as a relative path that will be searched using the \$CFGPATH environment variable

-l [ --log-level ] arg Log level: ERROR, WARNING, STATE, EVENT, ACTION, INFO, DEBUG, TRACE

#### 8.1.2 Sending commands

To send a standard command using the msgsend utility:

```
$> msgsend --icd ../ecs-interfaces/std/if/src/stdif.xml --uri  
zpb.rr://localhost:12081/telif/StdCmds ::stdif::StdCmds::Exit
```

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

```
$> msgsend --icd ./ccsinsif/icd/src/ccsinsif.xml --uri  
zpb.rr://localhost:12081/telif/Commands  
::ccsinsif::Commands::GetConfig
```

The standard commands supported are described in this ICD:

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

The specific commands supported are described in this ICD:



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

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

The python utility `telifsim 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](#).

## 9. 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 two Jupyter Notebooks that demonstrate such integration and in the future it is foreseen to implement a UI, some scripts or even an application implementing a telescope mount LSV interface simulator integrated with Stellarium

### 9.1 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
```

### 9.2 Configuration

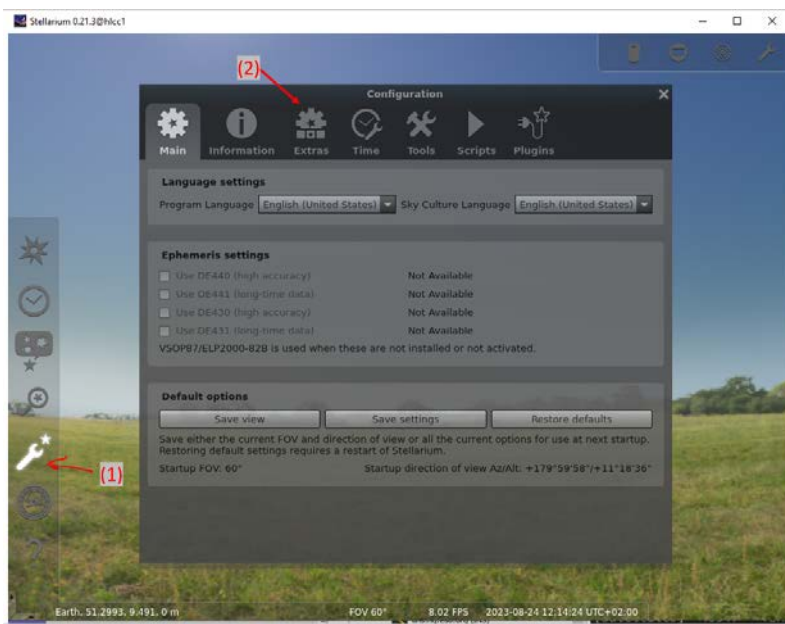
In order to optimally use Stellarium with HLCC it is necessary to configure it.

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).

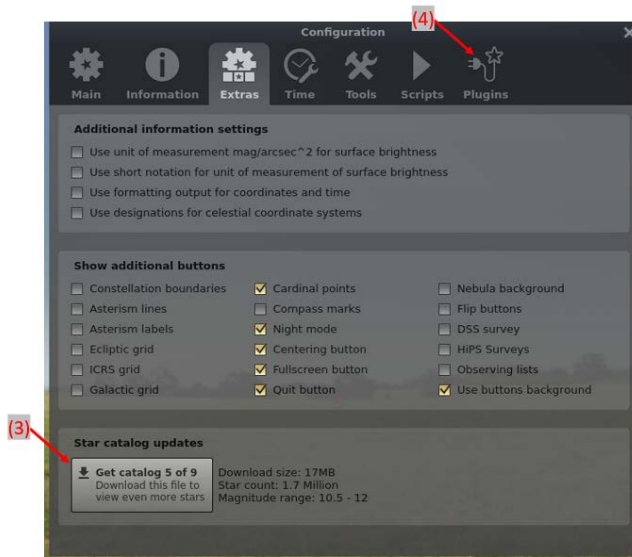
- Start Stellarium



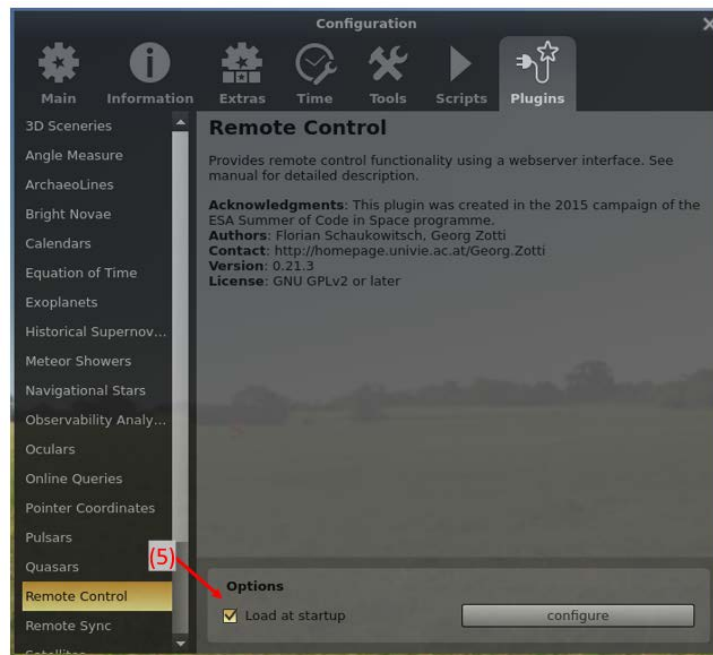
- To exit from the default full-screen view press F11
- Open the cascade command bar on the left, by getting near to the left frame border
- Select Configuration (1) to open the configuration panel
- Select Extras (2)

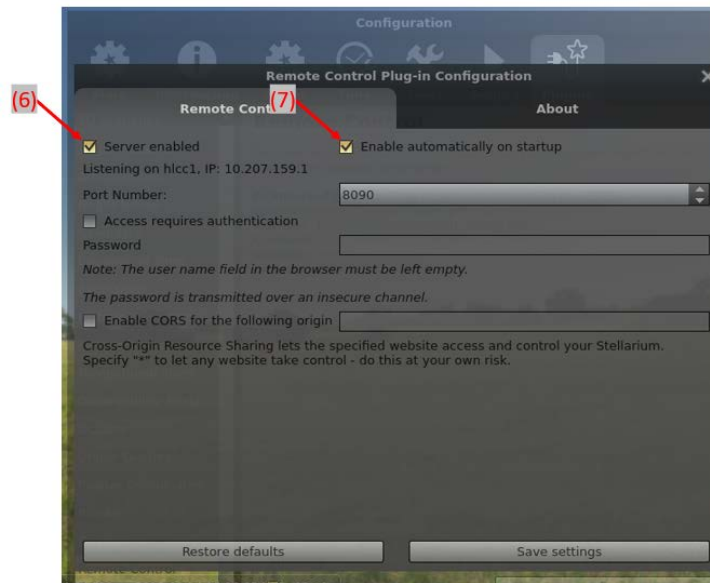


- Load all star catalogues (3)  
to get a more realistic view of what would be visible from the ELT.  
With all catalogues it is possible to see stars up to magnitude 17.

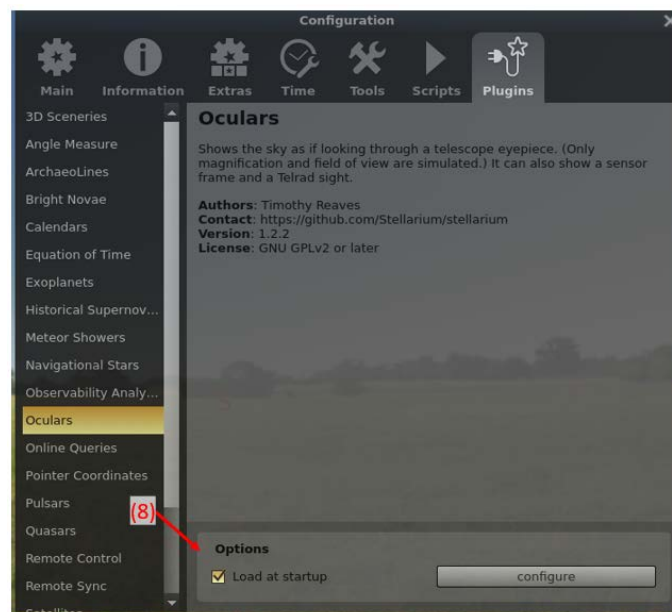


- Select Plugins (4)
  - Remote control plugin (plugin managing the web server with rest API)



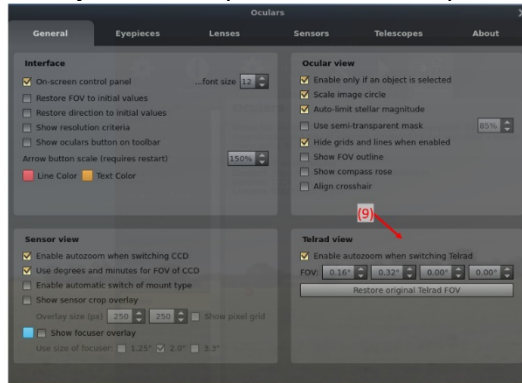


- Load at startup (5)
- restart Stellarium if not already selected
- Configure -> Server enabled (6), enable automatically at startup (7)
- Oculars

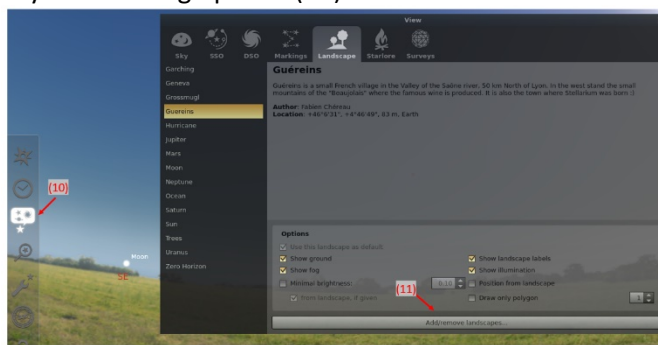


- Load at statup (8)
- restart Stellarium if not already selected

- Configure -> Telrad View (0.16, 0.32, 0.0, 0.0) to set the graphics to identify the telescope instrument and probe patrol FOVs (9)



- 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)





- Sky
  - To see the sky also during day time, switch off "atmosphere visualization" (13)



## 9.3 Running the notebooks

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

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.

**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.

## 9.4 Learning and debugging

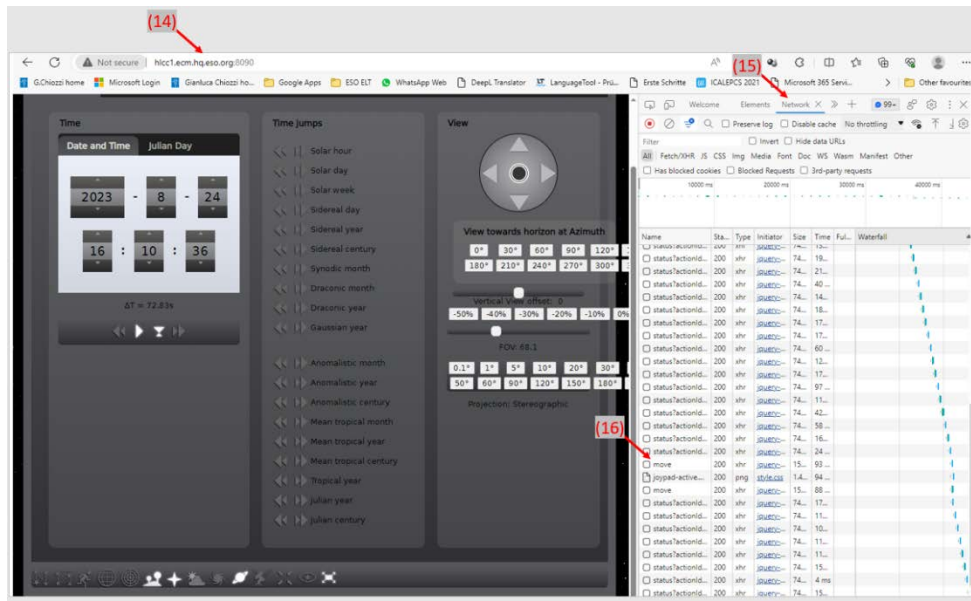
A very good way to learn the Stellarium Remote Control API and to debug is, it

- open the Remote Control 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)



## 9.5 To know more

Here some links 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](#)