European Organisation for Astronomical Research in the Southern Hemisphere

Programme: ELT

Project/WP: Instrumentation Framework

ELT ICS Framework - Transfer Document

Document Number: ESO-363356

Document Version: 4

Document Type: Manual (MAN)

Released on: 2024-12-11

Document Classification: Public

Owner:	Kiekebusch, Mario
Validated by PM:	Kornweibel, Nick
Validated by SE:	González Herrera, Juan Carlos
Validated by PE:	Biancat Marchet, Fabio
Approved by PGM:	Tamai, Roberto
	Name

Release

This document corresponds to ifw-doc[1] v2024.0.1.

Authors

Name	Affiliation
Kiekebusch, Mario	ESO/DOE/CSE

Change Record from Previous Version

Affected Section(s)	Changes / Reason / Remarks
	See CRE ET-1517
All	All sections updated for IFW24

[1] https://gitlab.eso.org/ifw/ifw-doc

Table of Contents

1 Introduction

This document serves as a general guide for the ICS Framework (IFW) version 2024, aiming to provide an updated ELT software stack for instrument developers. The framework is intended as a toolkit to assist instrument developers in implementing their control systems. The IFW has been validated with the following ELT packages:

- ELT Development Environment version 5.2.0 (Fedora Distribution)
- CII MAL 4.2.0
- CII Services 4.3.1

Note: From version 2024 onward, IFW changed the versioning scheme from a correlative number to a version based on the year.

1.1 Scope

This document is the transfer document for the ELT ICS Framework version 2024. The intended audience are ELT users, consortia developers or software quality assurance engineers.

1.2 Acronyms

DB	Database
CCF	Camera Control Framework
CCS	Central Control System
DAQ	Data Acquisition
DDT	Data Display Tool
ELT	Extremely Large Telescope
ETR	Extensible Test Runner
FCF	Function Control Framework
FCS	Function Control System
GUI	Graphical User Interface
HLCC	High Level Coordination and Control
ICS	Instrument Control System
IFW	ICS Framework
ODP	Online Data Processing
OLDB	Online DB
OTTO	The new observing tool: Otto Tackles Telescope Observations
PLC	Programming Logical Controller
RAD	Rapid Application Development
SEQ	Sequencer
SUP	Supervisor

1.3 Overview

The framework components included in version 2024 are listed in the release notes *Summary*.

Our software is divided in several independent WAF projects, see the picture below.

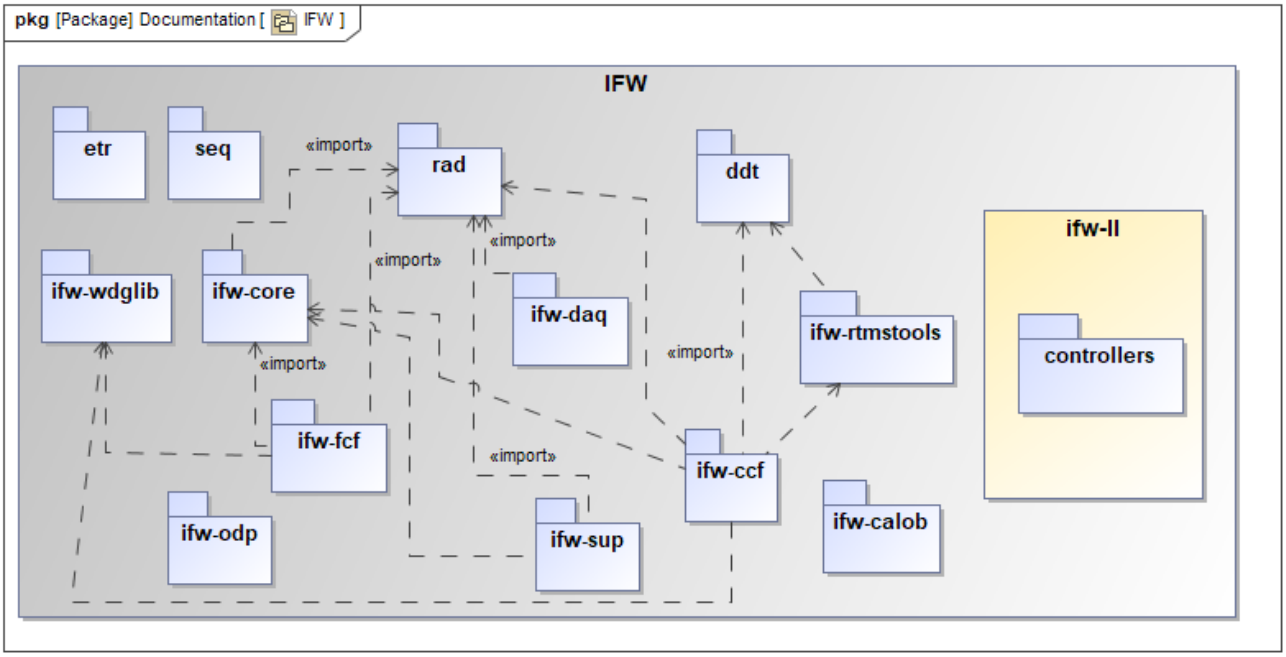


Fig. 1: IFW GIT packages and their main dependencies.

Note: The ifw-ll stores mainly PLC Visual Studio projects.

Warning: *Disclaimer:*

ESO does not warrant that the functions contained in version 2024 of the ICS Framework will meet all requirements or that the operation of their components and libraries will be flawless.

ESO does not ensure that solutions included in version 2024 are not subject to changes in future releases.

While every precaution has been taken in the development of the ICS Framework software and in the preparation of the documentation, ESO assumes no responsibility for errors or omissions, or for damage resulting from the use of the software or of the information contained in the documentation.

Note: The ICS Framework is distributed outside ESO for the development of applications related

2 Release Notes

2.1 Summary

The official release 2024 of the IFW includes the following products:

ICS Framework Product	Repository/Waf project	Product version	RPM version
Application Framework	rad	5.5.0	elt-rad-5.5.0
Test Runner Framework	etr	3.1.3	elt-etr-3.1.3
Sequencer	seq	4.2.0	elt-seq-4.2.0
Data Display Tool	ddt	1.1.0	elt-ddt-1.1.0
Camera Control Framework	ifw-ccf	4.0.0	elt-ifw-ccf-4.0.0
Observation Coordination Framework (daq)	ifw-daq	3.1.0	elt-ifw-daq-3.1.0
Observation Coordination Framework (sup)	ifw-sup	4.0.0	elt-ifw-sup-4.0.0
Widget Library (wdglib)	ifw-wdglib	0.2.0	elt-ifw-wdglib-0.2.0
RTMS Tools	ifw-rtmstools	2.0.0	elt-ifw-rtmstools-2.0.0
Calibration Framework (calob)	ifw-calob	0.1.3	elt-ifw-calob-0.1.3
Function Control Framework	ifw-fcf	6.0.0	elt-ifw-fcf-6.0.0
Online Data Processing	ifw-odp	4.0.1	elt-ifw-odp-4.0.1
Miscellaneous Core Libraries	ifw-core	5.0.1	elt-ifw-core-5.0.1

The links to the components release in Gitlab are :

- ifw-core release[1]
- ifw-rtmstools release[2]
- ifw-wdglib release[3]
- ifw-calob release[4]
- ifw-fcf release[5]
- ifw-sup release[6]

[1] https://gitlab.eso.org/ifw/ifw-core/-/releases
[2] https://gitlab.eso.org/ifw/ifw-rtmstools/-/releases
[3] https://gitlab.eso.org/ifw/ifw-wdglib/-/releases
[4] https://gitlab.eso.org/ifw/calob/-/releases
[5] https://gitlab.eso.org/ifw/ifw-fcf/-/releases
[6] https://gitlab.eso.org/ifw/ifw-sup/-/releases

- ifw-daq release[7]
- ifw-odp release[8]
- ifw-ccf release[9]
- ifw-calob release[10]
- ifw-ll release (fcf controllers)[11]
- rad release[12]
- etr release[13]
- seq release[14]

Note: *Repository for Binaries:*

We have moved our binaries to GIT Large File Storage (LFS). This includes but is not limited to PLC compiled libraries, PLC modules and other utilities (binaries for Windows).

What is the scope of this version?

Version 2024 is the official release of the IFW under Fedora distribution (ELT DevEnv 5.2.0). This release tries to keep updated developers with the latest developments the ELT software stack.

What is new in this release?

- IFW adapted to the latest ELT Software Stack (ELT DevEnv + ECOS)
- New versioning scheme based on the year rather than on a correlative number.
- Separation of MAL interfaces in dedicated Git repositories, e.g. ifw-fcfif for FCF interfaces.
- A new version of RTMS tools based on llnetio library is provided. The old version is still available but it will be removed in the future.
- A first version of the Cryogenic HMI libraries is provided.
- **Some changes in the Template Instrument, such as:**
  - Nomad files extension from .nomad to .hcl to follow new convention in the ELT project.

[7] https://gitlab.eso.org/ifw/ifw-daq/-/releases
[8] https://gitlab.eso.org/ifw/ifw-odp/-/releases
[9] https://gitlab.eso.org/ifw/ifw-ccf/-/releases
[10] https://gitlab.eso.org/ifw/ifw-calob/-/releases
[11] https://gitlab.eso.org/ifw/ifw-ll/-/releases
[12] https://gitlab.eso.org/ifw/rad/-/releases
[13] https://gitlab.eso.org/ifw/etr/-/releases
[14] https://gitlab.eso.org/ifw/seq/-/releases

  - Nomad files modified to support parameters (nomad variables). This allowed to remove hardcoded paths in template files.
  - Nomad files modified to limit reesheduling of jobs.
  - Simplified a bit the data acquisition sequence for templates.
- **Some improvements in DDT, such as:**
  - Final delivery from CGI merged into trunk and updated to Fedora 38
  - Implemented prototype of a multi-viewer display in Python.
  - Implemented prototype of signal visualization panel based on the transmission of 1D arrays (Python).
  - Converted DDT User Manual from Word to Sphinx/RsT
  - Implemented a new dialog to monitor performance of the broker. The dialog shows: Latency, sample count, number of subscriber, number of publishers, last sample receiver among other attributes.
- **Some improvements in the Sequencer, such as:**
  - seqtool gui modified to accept a sequence as command line parameter.
  - Added widget to report server execution status.
  - Added initial support for Instrument Package (IP)
  - Minor improvements in the GUI to display template parameters.
- **Some improvements in FCF such as:**
  - Implemented generic multi-axis device.
  - Externalized JSON schema files for client interfaces.
  - Implemented Cabinet Cooling Controller (CCC) PLC library.
  - Implemented a sensor widget with plotting capabilities.
  - Extended device simulators to support different PLC instances in the same project.
- **Some improvements in CCF, such as:**
  - Implementation of telemetry.
  - Implementation of reconnection capabilities for ENVision protocol. This should improve reliability under poor network conditions or unexpected disconnections.
  - Added report of skipped frames.

For a more detailed list of changes, please refer to the individual component release notes.

Some of the third party products and middleware solutions used by the ICS Framework are:

- Qt Framework[15] for developing GUIs.
- ZeroMQ[16] for implementing request/reply and publish/subscribe message patterns.
- Google Proto Buffers[17] for serialization/deserialization.
- Open62541 OPC-UA toolkit[18] for communication to the PLCs
- Google Tests and Mockup libraries[19] for unit tests.
- Robot Framework[20] for integration tests.
- Nomad/Consul[21] for software deployment.
- JSON format[22] for setup serialization. The Development Environment includes libraries in C++ (nlohmann) and Python (json) to work with JSON format.

2.2 Development Environment

- The Development Environment has been upgraded to Fedora 38 v5.2.0

For a detail release notes of the Development Environment, please visit here[23]

2.3 Instrument Specifics Packages

To facilitate the development and integration, the IFW is now delivered as a set of additional RPMs to be installed on top of the Development Environment. This saves developers from retrieving, building and installing the IFW by their own. The additional RPMs are not part of the Development Environment since this is specific for instruments and therefore its installation shall be done on demand.

2.4 Components

Rapid Application Development (RAD)

The is an application framework that enables the development of event-driven applications for the ELT, based on call-backs or state machines.

For details, see the *rad* user manual.

[15] https://www.qt.io/
[16] http://zeromq.org/
[17] https://developers.google.com/protocol-buffers/
[18] https://www.open62541.org/
[19] https://github.com/abseil/googletest
[20] http://robotframework.org/
[21] https://www.nomadproject.io/
[22] https://www.json.org/json-en.html
[23] https://ftp.eso.org/pub/elt/repos/docs/RELEASE_NOTES

Extensible Test Runner (ETR)

The component *etr* is included in the release to support running integration tests. The following features are available:

- Run Robot Framework tests with the *robot* plugin.
- Request test resources with the *resources* plugin.
- Modify template files with Jinja2 template engine and the *jinja2* plugin.
- Deploy software with Nomad with the *nomad* plugin (experimental support).

For details see the *etr* user manual.

Sequencer

The component *seq* is included in the release to support the implementation of engineering scripts. The following features are available:

- Sequencer Engine
- Sequencer API
- Sequencer CLI
- Sequencer GUI (experimental)

For details see the *seq* user manual.

Function Control Framework (FCF)

Device Manager

Supported Devices

- *Shutters*
  - Configuration parameters: initial state (open, closed), timeout for HW operations, signal logic, and more.
  - Close/Open control.
- *Lamps*
  - Configuration parameters: initial state (On, Off), timeout for HW operations, signal logic, and more.
  - On/Off control.
  - Intensity control.

  - Automatic switch off after a timeout.
- *Motors*
  - Configuration parameters: maximum velocity, axis type, initialisation sequence, timeouts, SW limits, backlash compensation, usage of brakes, named positions, and more.
  - Move in absolute encoders and user units.
  - Move using named positions.
  - Move in relative encoders.
  - Move in speed.
- *IOdevs*
  - Monitor a list of engineering variables.
  - Values are not continuously read from the LCS but updated on data change.
  - Write digital and analog signals.
- *Derotators*
  - Five operation modes: engineering, stationary, sky, elevation and user defined.
  - Tracking computation is based on slalib running in the PLC.
- *ADCs*
  - Two operation modes: off and automatic.
  - Multi-axis support.
  - Tracking computation is based on slalib running in the PLC.
- *Piezos*
  - Two operation modes: position and automatic.
  - Supports up to three axes.
  - Move in bits and userunits.
- *Actuators*
  - Generic On/Off control.

Device Simulators

FCF provides a set of simulators to all supported devices. The simulators have been implemented in Python based on the FreeOpcUa toolkit[24] and the "rad/scxml4py" engine.

Engineering Graphical Interfaces

The Device Manager includes two graphical applications, see fcf_gui_ref.

Warning: These two applications shall be considered just as a prototype implementations. Their design (color scheme, layout and in general their look&feel) may change in the future according to the development of ELT widget libraries and standards for graphical interfaces.

- FCF GUI (fcfGui): Engineering interface for the control and monitoring of *Device Manager* and devices under its control.
- Motor GUI (fcfMotGui): Engineering interface for the control and monitoring of a single motor device, implemented in Python.

PLC Libraries

PLC Libraries are now available in GIT and can be retrieved from the ifw-resource repository[25]. Note that this repository uses the LFS (Large File Storage) Git extension. All libraries are located in the same directory. This simplifies the installation of the libraries in the TwinCAT IDE.

The PLC source code can be retrieved from: https://gitlab.eso.org/ifw/ifw-ll/-/releases/v6.0.0

All libraries have been created with **TwinCAT 3.1.4024.35.**

Required TwinCAT 3.1 build is 4024.35 or higher.

Required OPC UA Server version is 3.3.16.0 or higher.

2.5 Known Problems

1. Errors running GUIs for the first time after installation:

Errors may occur when running the GUIs for the first time after installation. These errors are due to missing OLDB attributes dynamically created by the applications.

[24] https://github.com/FreeOpcUa/python-opcua
[25] https://gitlab.eso.org/ifw/ifw-resource

3 Installation

3.1 Machine Preparation

Install a real or virtual machine with Fedora, ELT Dev packages and ELT Common Software (ECOS, including CII) according to the Linux Installation Document[26]

ESO recommends setting the option ELT_STABLE_STREAM=no as the default configuration before installing DevEnv. This setting is intended to disable the upgrade of packages, ensuring better configuration control. The default configuration can be changed at any time if needed.

3.2 Requirements

For this version, the IFW requires a WS with at least 8Gb of RAM memory to compile.

3.3 ELT Development Environment (DevEnv)

Before installing the IFW it is needed to setup the environment. A very good starting point is this document[27]

Note: The version **5.2.0** of ELT Development Environment (Fedora) shall be used with version 2024 of ICS Framework (IFW).

3.4 ELT CII Services

The IFW requires the proper installation and configuration of CII services on the machine where it is intended to be deployed. You can just execute the following script **as root** (see below) or follow the post installation procedure documented here[28]

The IFW team recommends to use the "ownserver" role for the development machine where the IFW is going to be installed.

```
$ /elt/ciisrv/postinstall/cii-postinstall role_ownserver
$ cii-services start all
```

For more information about CII post installation and other CII related topics, you can visit the ESO knowledge base page[29]

[26] https://ftp.eso.org/pub/elt/repos/docs/devenv5/install/html/index.html
[27] https://ftp.eso.org/pub/elt/repos/docs/devenv5/developing-software-guide/html/index.html
[28] https://www.eso.org/~eltmgr/CII/v2024-03/4.3.0/manuals/html/docs/services.html#cii-postinstall
[29] https://gitlab.eso.org/ecos/eltsw-docs/-/wikis/KnowledgeBase/CII

3.5 Operational User

It is recommended to install and run the instrument software under the eltdev account to simplify integration with Nomad/Consul that comes configured to run under eltdev in DevEnv version 5.2.0.

Note: Since version **5.2.0** of ELT Development Environment (Fedora), nomad runs as root.

It is possible to start/shutdown the software from another user but it requires setting properly the environment variables for the account where you want to run your software.

3.6 Environment Variables

IFW is relying on the following environment variables:

- **DATAROOT:** Directory on the host machine in which Output Data Products will be generated.
- **CFGPATH:** The "CFGPATH" environment variable, is a colon separated list of paths, pointing to possible Resource Directories in which resource data of different kinds are located.
- **INTROOT:** In this release, the example configuration, libraries, header files and binaries, are installed into the location pointed by "INTROOT". It is used as "PREFIX" for waf installation location.

3.7 Getting Started

1. Checking CII services

   Check that CII services are properly running:

```
$ cii-services info
```

The output of the above command shall be similar to this:

```
CII Services Tool (20240220)
...
# function .....
Log       |functional:yes
OLDB DP   |functional:yes
OLDB CE   |functional:yes
IntCfg    |functional:yes
```

If they are not properly running you can execute (as **root**):

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

2. Install IFW components

Starting from IFW version 4, all IFW components are available as RPMs and can be installed using dnf install as **root**.

```
$ dnf -y install elt-ifw
```

This command will install all ifw RPMs and the components seq, ddt RPMs :

- elt-ifw-2024.07
- elt-ifw-if-devel-2024.07
- elt-ifw-core-5.0.1
- elt-ifw-supif-1.0.0
- elt-ifw-wdglib-0.2.0
- elt-ifw-rtmstools-2.0.0
- elt-ifw-odp-4.0.1
- elt-ifw-daqif-1.0.0
- elt-ifw-fcfif-1.0.0
- elt-ifw-fcf-6.0.0
- elt-ifw-fcf-devel-6.0.0
- elt-ifw-fcfif-devel-1.0.0
- elt-ifw-daq-3.1.0
- elt-ifw-daq-devel-3.1.0
- elt-ifw-daqif-devel-1.0.0
- elt-ifw-ccf-4.0.0
- elt-ifw-ccf-devel-4.0.0
- elt-ifw-odp-devel-4.0.1
- elt-ifw-rtmstools-devel-2.0.0
- elt-ifw-wdglib-devel-0.2.0
- elt-ifw-sup-4.0.0
- elt-ifw-sup-devel-4.0.0
- elt-ifw-supif-devel-1.0.0
- elt-ifw-calob-0.1.3
- elt-ifw-calob-devel-0.1.3
- elt-ifw-core-devel-5.0.1

- elt-ifw-wdglib-doc-0.2.0
- elt-ifw-supif-doc-1.0.0
- elt-ifw-sup-doc-4.0.0
- elt-ifw-rtmstools-doc-2.0.0
- elt-ifw-odp-doc-4.0.1
- elt-ifw-fcfif-doc-1.0.0
- elt-ifw-fcf-doc-6.0.0
- elt-ifw-daq-doc-3.1.0
- elt-ifw-core-doc-5.0.1
- elt-ifw-ccf-doc-4.0.0
- elt-ifw-calob-doc-0.1.3

Note: Since IFW version 4, the elt-etr RPM is installed together with ELT Development Environment.

Note: The RPM elt-ifw-2024 is a mega RPM package to simplify the installation of all RPMs needed by an Instrument.

3. Install HLCC Components (optional)

Since IFW version 5, the HLCC components are available as RPMs and can be installed using yum install as **root**. The HLCC includes a telescope simulator that can be used to validate the non-deterministic interface between the instrument and the ELT CCS. For more information about HLCC refer to the User Manual[30]

Using the HLCC with the template instrument is optional. In case you want to use it, please install it in a separate machine, and follow the installation procedure documented in the HLCC User Manual.

4. Create the directories for the installation areas:

```
$ cd <the location for introot>
$ getTemplate -d introot INTROOT
$ cd <the location for dataroot>
$ getTemplate -d dataroot DATAROOT
```

The environment shall contain the definitions of the relevant environment variables such as INTROOT, DATAROOT, LD_LIBRARY_PATH, PYTHONPATH, etc. These environment variables will be automatically defined by means of the file *private.lua*, defined here below, which in turn uses the system modulefile definitions in */elt/System/modulefiles/introot.lua*. (In the following steps, we assume that

[30] https://ftp.eso.org/pub/elt/repos/docs/HLCC/webpages/hlcc-main/html/html/

INTROOT and DATAROOT directories are created in the home directory of the user.)

5. Under eltdev home directory:

```
$ mkdir modulefiles
$ cd modulefiles
```

6. Create and edit the file private.lua under modulefiles directory. Use the example file below:

```
local home = os.getenv("HOME")

local introot = pathJoin(home, "INTROOT")
setenv ("INTROOT", introot)
setenv ("PREFIX", introot)

local dataroot = pathJoin(home, "DATAROOT")
setenv ("DATAROOT", dataroot)

load ("introot")
setenv ("NOMAD_ADDR", "http://<node ip>:4646")
setenv ("CONSUL_ADDR", "http://<node ip>:8500")

prepend_path("CFGPATH","<path_to_ins>/<instrument>/resource")
```

Note: IFW RPMs (elt-ifw, elt-seq, elt-ddt) provide a lua file to define minimum set of default environment variables. The lua files are installed under /elt/common/modulefiles/default and loaded by default.

Warning: Log out and then in again so that modulefiles directory becomes known to the environment and the newly created private.lua is loaded. This is needed only when the directory modulefiles and the private.lua are created for the first time.

File private.lua is loaded by default upon login. In case more .lua files (with different names) will be added to $HOME/modulefiles, they can be made known to the environment just with:

```
$ module load <lua file>
```

You can check which LMOD modules are available after login with:

```
$ module avail
```

The output should look like as follows: (the available/loaded modules might change with the software versions, but *private* and *introot* should be loaded)

```
------------------------------------------------------------------------------ /home_local/eltdev/modulefiles
   ---------------------------------------------------------------------------
   private (L)

------------------------------------------------------------------------------ /elt/common/modulefiles/core -----
----------------------------------------------------------------------------
   eltdev (L)    introot (L)

------------------------------------------------------------------------------ /elt/common/modulefiles/default
   ---------------------------------------------------------------------------
   ciisrv (L)    cut (L)    dcsif (L)    ddt (L)    elt-trs (L)    ifw (L)    mal (L)    metadaqif (L)
mudpi (L)    ra
d (L)    recif (L)    roadrunner (L)    rtms (L)    seq (L)    stdif (L)

------------------------------------------------------------------------------ /usr/share/lmod/lmod/modulefiles/
Core ---------------------------------------------------------------------------
   lmod    settarg

Where:
   L:  Module is loaded
```

Note: For more information, read this document[31]

3.8 Start nomad and consul services

Following ELT standards, IFW uses Nomad (see here[32]) to manage the life cycle of the ICS SW components (see next chapter Getting Started).

Warning: Unlike in the previous releases of the DevEnv, users have to provide a custom configuration for Nomad/Consul services. The default one cannot be used.

The IFW template provides an example configuration for Nomad/Consul that projects could use as starting point. These files are located under:

```
$ <instrument>/<prefix>-resource/nomad/etc
```

Replace the <node ip> with the correct IP for your IWS machine and place these files under /etc/nomad.d and /etc/consul.d.

[31] https://ftp.eso.org/pub/elt/repos/docs/devenv5/developing-software-guide/html/docs/devtools.html#environmental-modules-system-lmod

[32] https://www.nomadproject.io/

Nomad and Consul services has to be started as the eltdev user with the command systemctl.

- Start nomad and consul services

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

- Check status of nomad and consul services

```
$ systemctl status nomad
$ systemctl status consul
```

- Stop nomad and consul services

```
$ systemctl stop nomad
$ systemctl stop consul
```

3.9 Manual installation

As in previous versions, the IFW components can also be installed from Gitlab using the release tar file. This step is optional since IFW software is already included in the RPMs mentioned above.

Note: The components to be installed will depend on the usage required by developers. Below are the instructions of the packages to be installed that are needed to follow the examples provided throughout the manual. Other IFW components can be installed following the same procedure.

Warning: Before building IFW components (rtmstools, ccf), the DDT RPM need to be installed using yum install as **root**.

```
$ dnf -y install elt-ddt-devel
```

- Download, unpack and build IFW components

Go to the ESO Gitlab site and download the release tar file for each component (ifw-core, ifw-daq, ifw-daq-if, ifw-rtmstools, ifw-odp, ifw-wdglib, ifw-fcf, ifw-fcf-if, ifw-ccf, ifw-sup, ifw-sup-if, ifw-calob).

After unpacking the ifw packages downloaded from ESO GitLab, execute the steps below to build and install the software for each component.

```
$ cd ifw-core
$ waf configure
$ waf build install
```

(continues on next page)

(continued from previous page)

```
$ cd ifw-daq-if
$ waf configure
$ waf build install

$ cd ifw-daq
$ waf configure
$ waf build install

$ cd ifw-rtmstools
$ waf configure
$ waf build install

$ cd ifw-wdglib
$ waf configure
$ waf build install

$ cd ifw-sup-if
$ waf configure
$ waf build install

$ cd ifw-sup
$ waf configure
$ waf build install

$ cd ifw-fcf-if
$ waf configure
$ waf build install

$ cd ifw-fcf
$ waf configure
$ waf build install

$ cd ifw-odp
$ waf configure
$ waf build install

$ cd ifw-ccf
$ waf configure
$ waf build install

$ cd ifw-calob
$ waf configure
$ waf build install
```

3.10 Project Template

This template will provide a sample configuration for an instrument project. Developers can adapt it to their own instruments. Examples in the documentation will refer to this template so it is recommended to download it from GitLab.

Go to the ESO Gitlab site and download the tar file of the template: Template release[33]

After unpacking the template, you should see the following:

```
$ tree ifw-templates
ifw-templates
└── project
    ├── cookiecutter.json
    ├── {{cookiecutter.project_name}}
    └── hooks
```

The instructions how to use this template can be found in next chapter (Getting Started).

[33] https://gitlab.eso.org/ifw/ifw-templates/-/releases

4 Getting Started

This section will guide the ICS software developer in creating a working project using a provided coockiecutter template. For getting the template from GitLab, go to the ESO Gitlab site and download the tar file of the template: Template release[34]

The user will be able to start the software components with the specific configuration prepared as a showcase for instrument developers. More details for each of the components will be given in the respective user manuals.

4.1 Updating an existing Project Configuration

If a version of your instrument project already exists, it is recommended to start from scratch following the instructions below and update your specific configuration in the new template afterwords. This approach might be better because of the possible changes in the template configuration and in the IFW components. If you have problems doing this, please contact ESO to get help doing this migration. Find here some guidelines to do the porting from previous version: ifw-porting.

4.2 Creating a Project Configuration

The IFW includes a project template that can be used to generate the initial package of an instrument. The generated project can be considered as a mini template instrument that could be used as starting point for the development of the control software. It is still basic but the idea is to develop it further in future versions according to the progress of the framework components.

The generated directory contains a fully working waf project with the instrument directory structure, some configuration files and some custom subsystem samples, e.g. an FCS including a special device. In this example we will use “micado” as an example instrument. After executing the *cookiecutter* command with the provided template, the system will request the user input to enter the information for the generation of the configuration and customized code. This template also generates the code for a special FCF device that in this case we will name as “mirror”. The ‘component_name’ is referring to an instance of FCS.

```
> cookiecutter ifw-template/project
[1/10] project_name (myproject): micado
[2/10] project_description (my project description): MICADO project
[3/10] project_prefix (xxx): mic
[4/10] nomad_user (eltdev):
[5/10] component_name (mycomponent): fcs
[6/10] device_name (mydevice): mirror
[7/10] olas_directory (): <path to OLAS directory>
[8/10] hlcc_available (No):
```

(continues on next page)

[34] https://gitlab.eso.org/ifw/ifw-templates/-/releases

(continued from previous page)

```
[9/10] hlcc_uri (zpb.rr://127.0.0.1:000):
[10/10] ngc2o_available (No): No
```

Note: The OLAS directory is where the FITS files will be stored after they are completed to be sent to the archive system.

Note: You can skip the usage of HLCC by setting the hlcc_available to No.

The generated directory structure including the first two levels is shown below. In this case, the directory *mic-ics* is a waf project that can be built. The resource directory is meant for storing the instrument resources like configuration files.

```
micado                  # Instrument repository
├── mic-ics             # Valid waf project
│   ├── build
│   ├── fcs             # Custom FCF instance
│   ├── micstoo         # Startup/Shutdown sequencer scripts
│   ├── seq             # Sample template implementation.
│   └── wscript
└── mic-resource        # Instrument resource directory
    ├── config          # Configuration files
    ├── nomad           # Nomad job files
    └── seq             # Sample OB
```

After the new directory is created, one could build and install the generated software.

```
cd micado/mic-ics
waf configure
waf build install
```

4.3 Update CFGPATH environment variable

If not already done, the CFGPATH environment variable shall be updated in the LMOD configuration to include the template resource directory. Add the following line to the modulefiles/private.lua file.

```
prepend_path("CFGPATH","<path_to_template>/micado/mic-resource")
```

Note: The above setting is needed for the proper functioning of tge IFW components. Make sure the setting is correct before starting nomad.

4.4 Starting/Stopping the ICS Software

The IFW uses Nomad (see here[35]) to manage the life cycle of the ICS SW components following the recommendation from the ELT Control project. We are also using Consul, a complementary package providing service discovery that allow us to use names instead of using hostname/IPs and port numbers.

The project template includes the Nomad job configuration to start-up/shutdown the ICS components that are generated by the coockiecutter template. We are also proving a Startup/Shutdown Sequencer script that uses the Nomad jobs to start/stop the complete ICS SW resembling the *osfStartup* tool in the VLT.

4.5 Startup/Shutdown Contents

The project template comes with a predefined startup/shutdown script to start/stop a representative sample of ICS software processes. The list of processes is here:

- DDT broker
- CCF instance with Simulator and DDT publisher
- FCF Simulators (shutter, lamp and motor)
- Subsystem Simulators (subsim1, subsim2 and subsim3)
- Custom FCF server instance with custom device (mirror).
- Custom FCF simulator (mirror)
- HLCC processes (optional)
- NGCII Optical (optional)
- OCM instance
- DPM instance
- System Supervisor

These components are obviously using simulators and not real hardware. The script shall be executed by the Sequencer.

The script contains three main parts:

1. Stop all processes
2. Start all processes
3. Move all processes to Operational state.

[35] https://www.nomadproject.io/

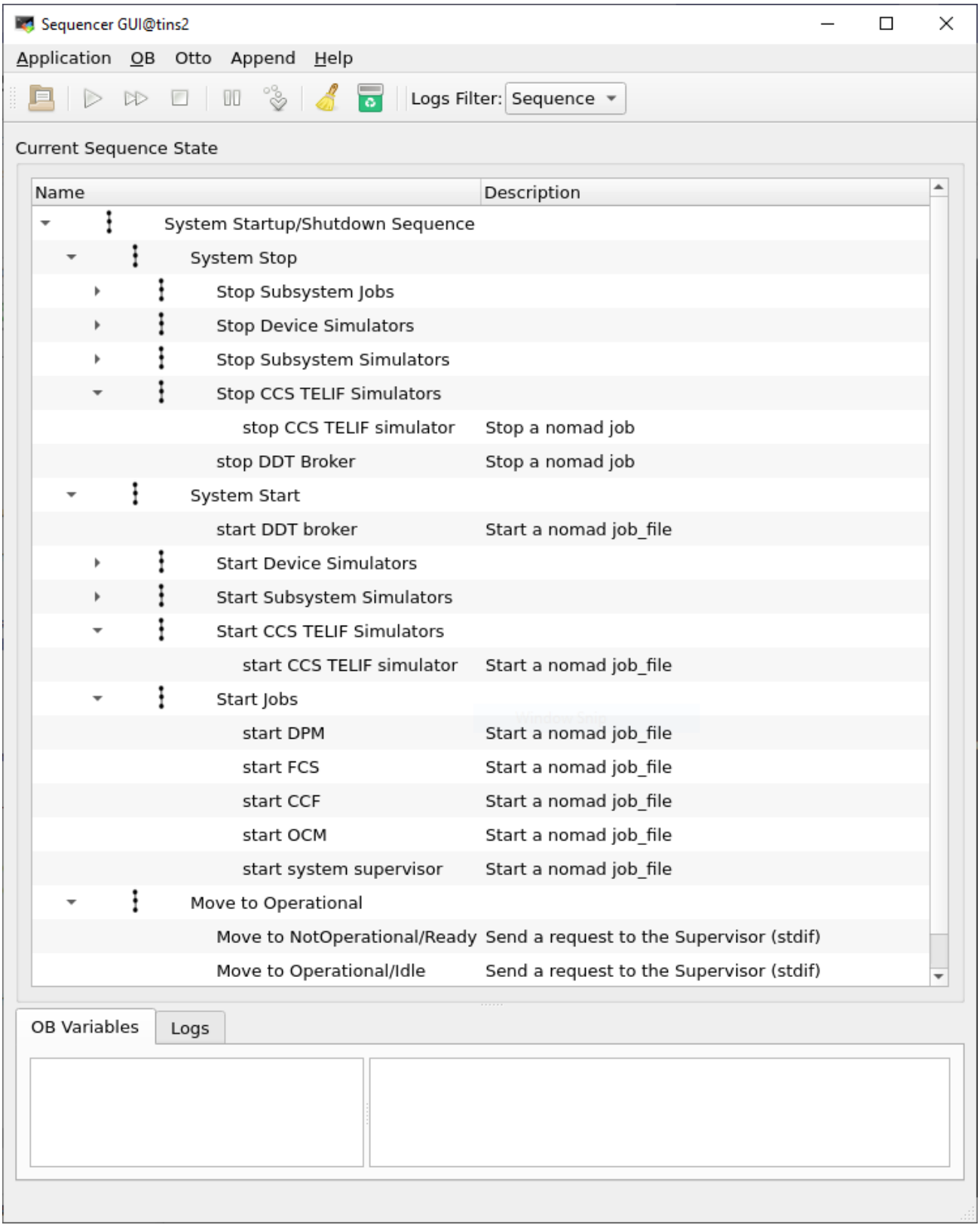


Fig. 1: Startup/Shutdown script in the Sequencer.

4.6 Executing Startup/Shutdown Script

Starting Sequencer GUI

In a terminal, type the following command to start the sequencer GUI.

```
> seqtool gui --config config/seqgui_config.yaml
```

Note: Since IFW version 2024, you need to specify the GUI configuration to enable the support of Instrument Package (IP).

Running the Startup Script

Once the Sequencer GUI is running. Load the startup script (micado/micics/micstoo/src/micstoo/startup.py) by selecting the Load Script option as shown in the following figure. It is assumed that the software has been already built and installed.

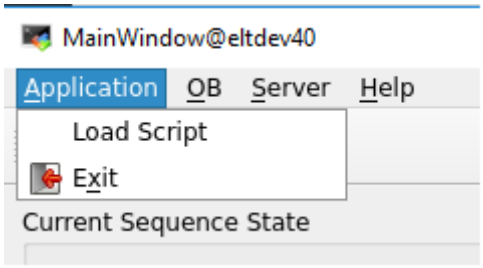


Fig. 2: Load script option from Sequencer File menu.

To execute the script, just press the play icon at the top of the Sequencer GUI as it is shown in the next figure. At the end of the execution, all instrument jobs shall be running and the system should be in Operational state.

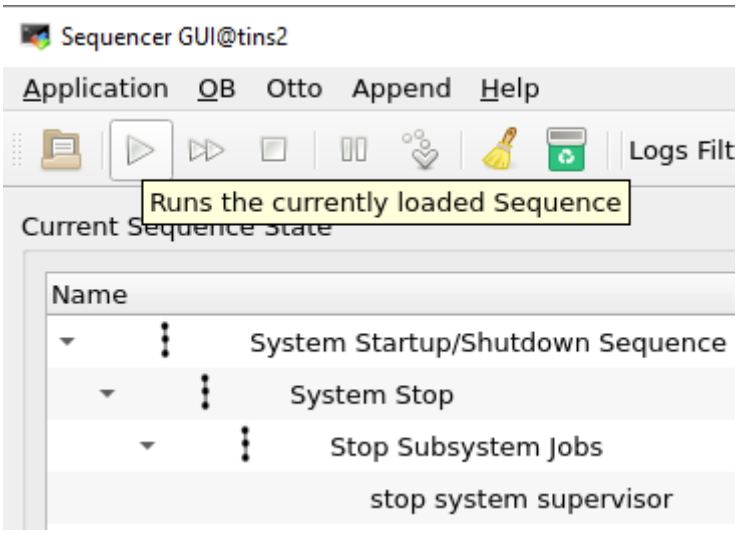


Fig. 3: Load script option from Sequencer File menu.

A quick way to verify is to check the status of the Supervisor.

```
> supClient `geturi syssup-req` GetStatus
Operational;Idle
```

After a successful execution of the startup script, the nomad web UI can be used to verify the status the nomad jobs. A total of 15 jobs shall be running.

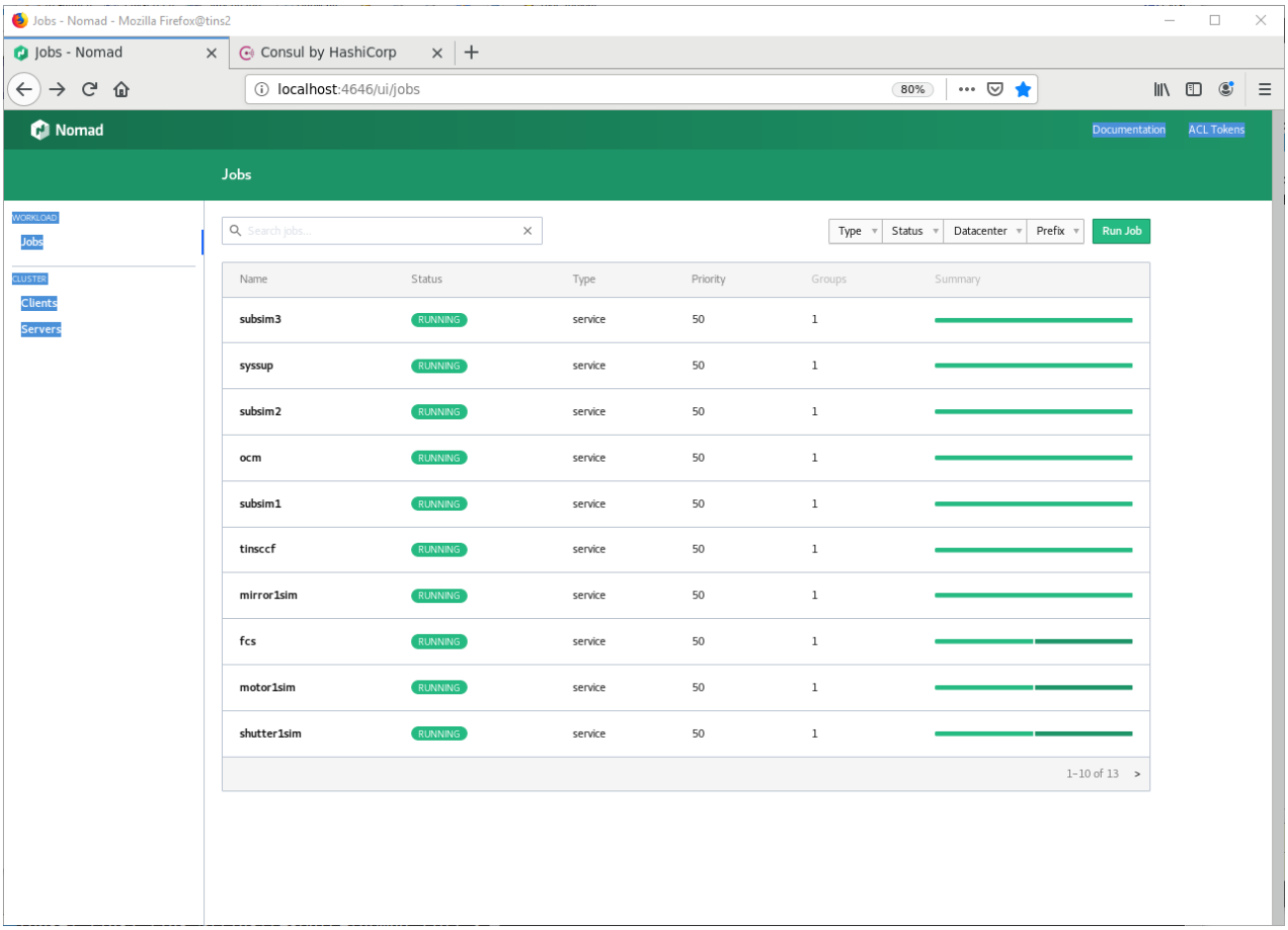


Fig. 4: List of TINS Nomad Jobs.

The Consul UI can be used to verify the services registered. In this case the number of services is greater because in some cases there are two services defined per each Job.

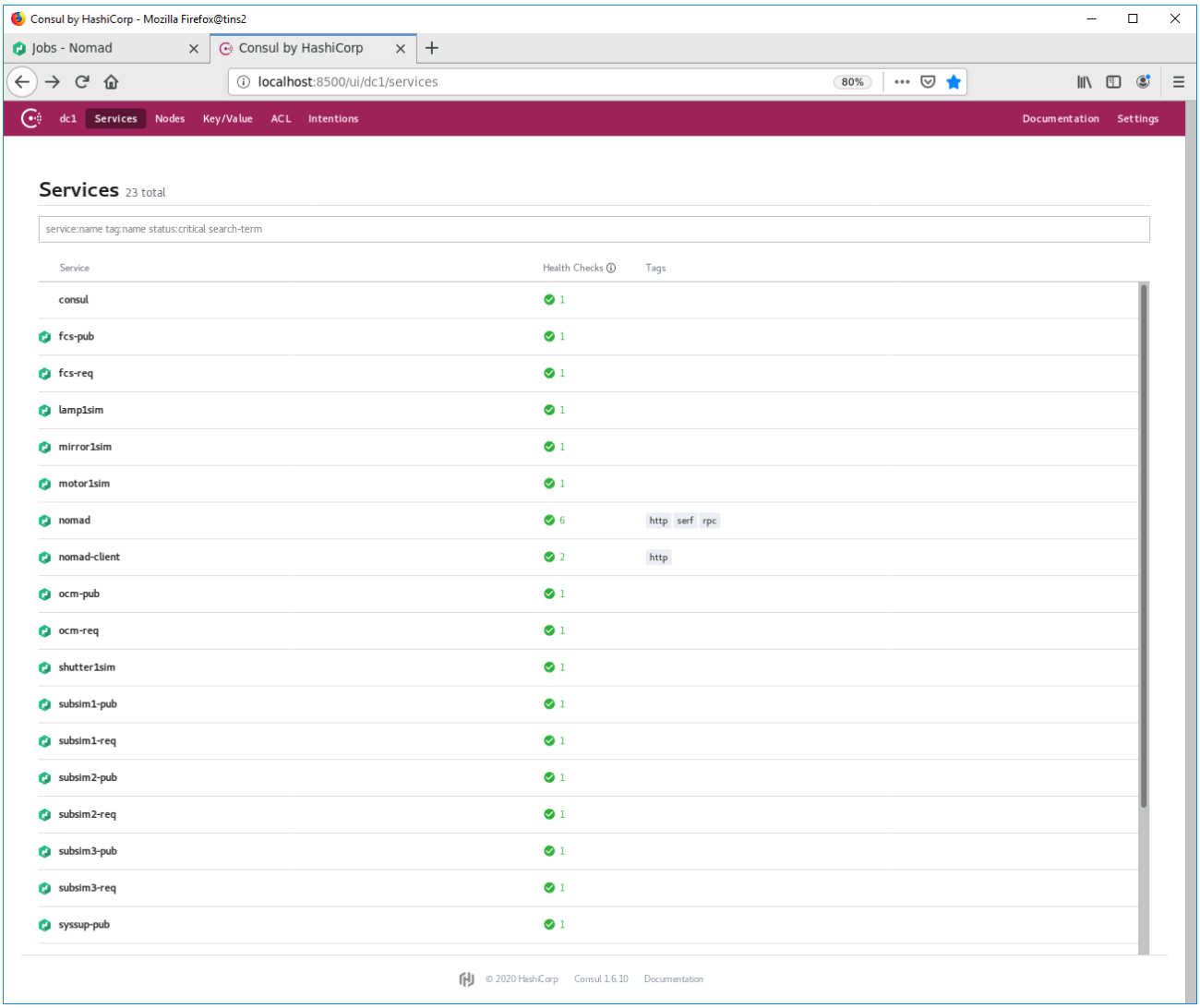


Fig. 5: List of TINS Consul Services.

Note: As convention, we use <service>-req as the registered name in consul for the service request/reply port, e.g. fcs-req

As convention, we use <service>-pub as the registered name in consul for the service publish/subscribe port, e.g. fcs-pub

Troubleshooting

If all or some processes do not start, make sure of the following:

1. Check that Nomad/Consul have been started and are running correctly. Try using the Systemd commands to get status of the service, see below.

```
> systemctl status nomad
* nomad.service - Nomad
Loaded: loaded (/usr/lib/systemd/system/nomad.service; disabled; vendor preset: disabled)
Active: active (running) since Tue 2021-04-20 07:14:11 UTC; 3 weeks 0 days ago
 Docs: https://nomadproject.io/docs/
Main PID: 413992 (nomad)
    Tasks: 541
Memory: 969.4M
CGroup: /system.slice/nomad.service
        ├─ 413992 /opt/nomad/bin/nomad agent -config /opt/nomad/etc/nomad.d
        ├─2864103 /opt/nomad/bin/nomad logmon
       ...
```

2. Make sure eltdev user has properly defined its environment. All environment variables shall be defined under eltdev since it is at the end the user that runs the processes through Nomad.

3. Check the status information of Nomad/Consul services with journalctl.

```
> journalctl -u nomad
```

4. Stop Nomad and Consul and run them manually outside the Systemd to get all logs and see the possible cause of the issues.

Validating the Software with a sample OB

We have prepared a very basic OB with an acquisition and observation template. The acquisition template prepares FCS, the camera system and the telescope simulator for the upcoming observation template that takes an image with the camera control system.

Note: The interaction between the Sequencer and the components is through the python client libraries provided by each component.

Before to start, the current script loaded in the Sequencer GUI must be cleared by pressing the reset button (trash icon).

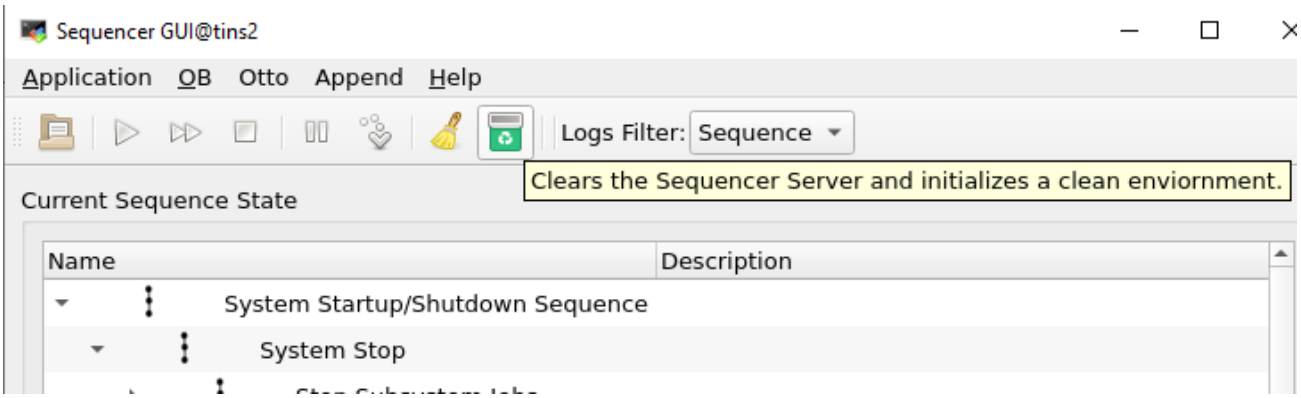


Fig. 6: Clear the current script and reset the server.

Then, the OB shall be loaded by pressing the open button as shown in the next figure. The path of the sample OB is: micado/mic-resource/seq/tec/MICADO_OB_sample.json.

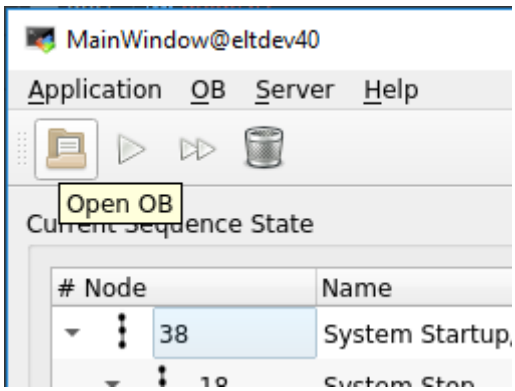


Fig. 7: Load an OB.

To run the template, just press the play icon at the top of the Sequencer GUI.

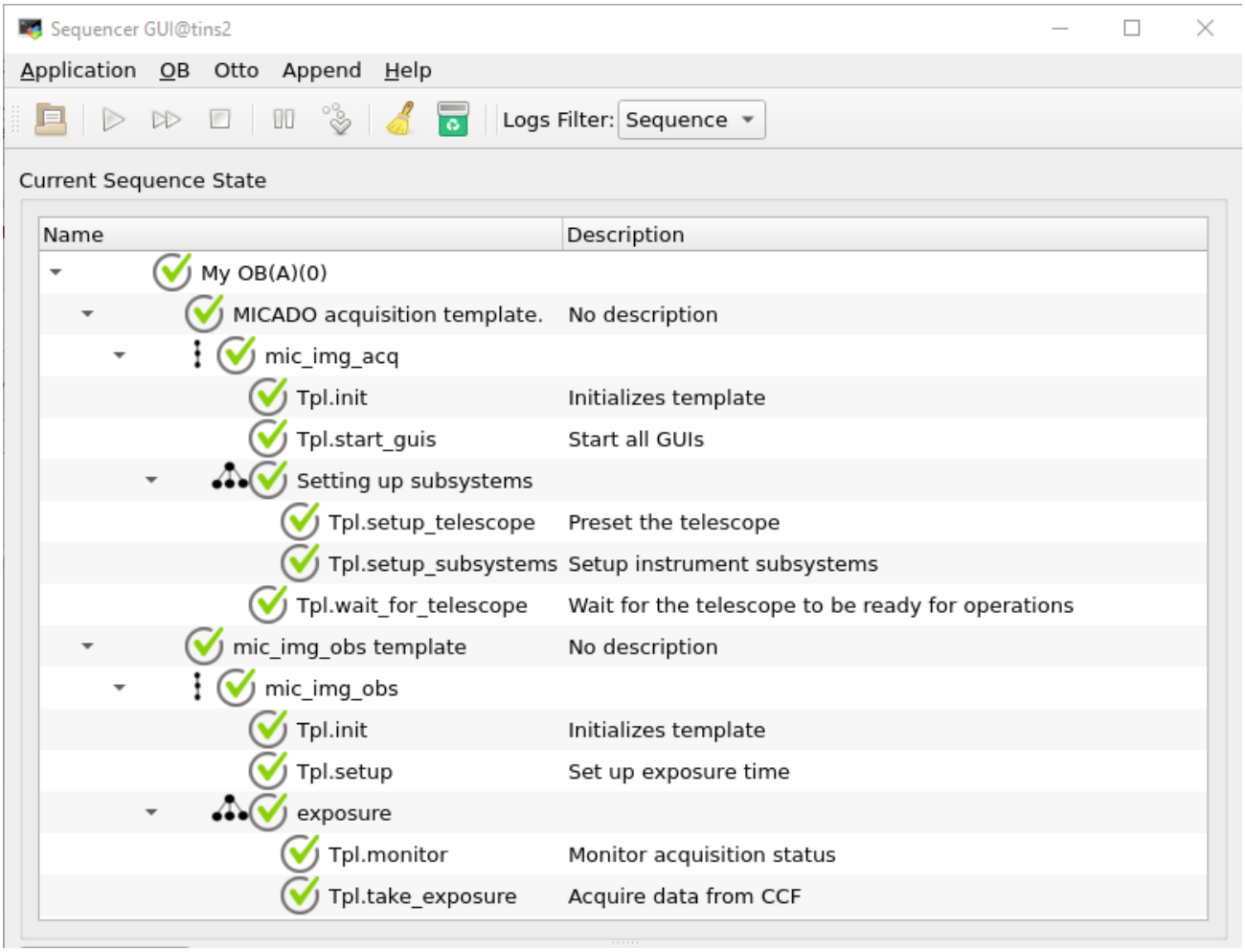


Fig. 8: Sample OB.

At the end of the execution, the image acquired by CCF shall be displayed in the DDT Viewer that it started by the template.

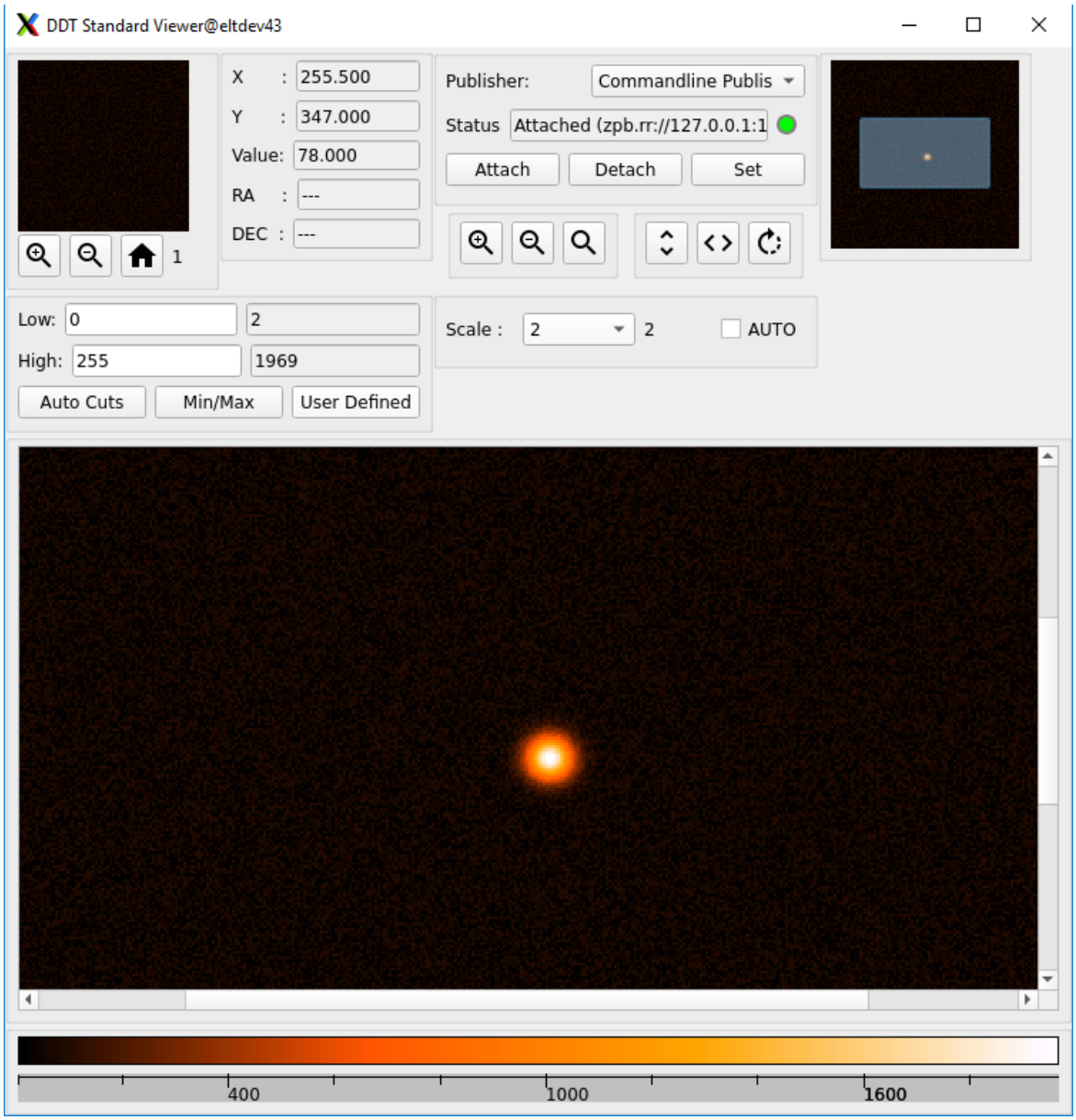
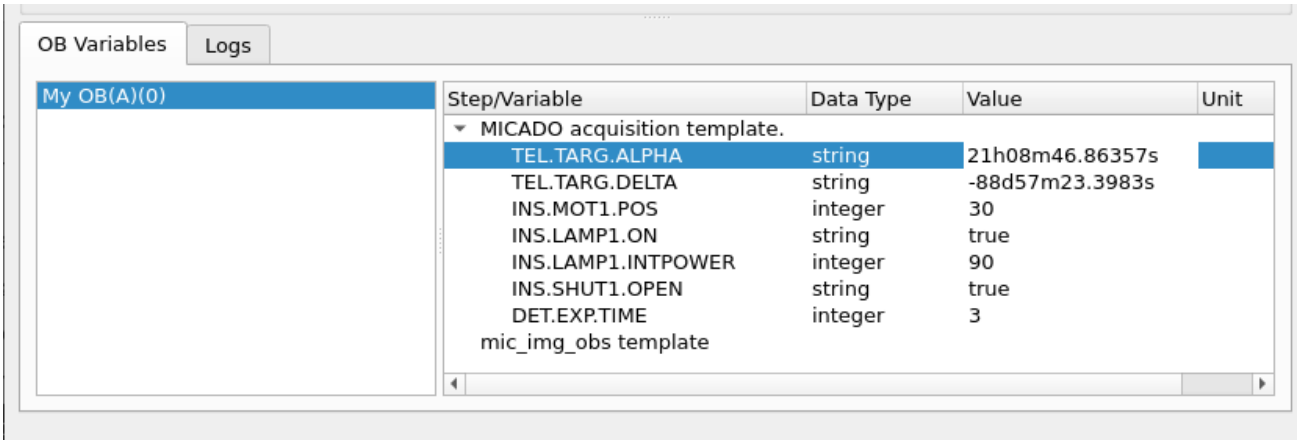


Fig. 9: DDT Viewer with image received from CCF.

When HLCC is used, you can verify using the HLCC GUI that the telescope simulator is pointing to the right coordinates. You could change the OB parameters from the sequencer GUI and execute the OB again validating the new values are correctly received by the telescope simulator.

Note: For simplicity, we are currently only sending alpha and delta parameters.



The resulting FITS file generated by DPM is located under $DATAROOT/dpm/result.

Congratulations that you reached the end of the general Getting Started section. Further instructions you may find in the specific documentation of the components.

Updating Sample Configuration

To update the default configuration of the template, developers can modify the configuration files that are located under the resource directory, e.g. under resource/nomad.