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: 2

Document Type: Manual (MAN)

Released on: 2021-05-31

Document Classification: Public

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

Release

This document corresponds to `ifw-hl`[1] v3.0.0.

Authors

Name	Affiliation
Mario Kiekebusch	ESO/DOE/CSE

Change Record from previous Version

Affected Section(s)	Changes / Reason / Remarks
	See CRE ET-1084
Overall	- Updated IFW version
Overview	- Removed list of components
	- Updated package diagram.
Release Notes	- Added summary table.
	- Updated package diagram.
	- Added links to the GitLab releases.
	- Update what's new.
	- Update Known Problems.
	- Updated PLC Library Versions
Installation	- Updated installation procedures.
	- Added section Manual Installation.
	- Added section Project Template.
Getting Started	- Added new section.

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

Table of Contents

1 Introduction

ICS Framework (IFW) version 3.0 is the third version of the framework in the pre-release phase. The framework is being developed by ESO and intended as toolkit to help instrument developers to implement their control systems. This version (3.0) includes the components according to the incremental delivery plan of the IFW.

1.1 Scope

This document is the transfer document for the ELT ICS Framework version 3. The intended audience are ELT users, consortia developers or software quality assurance engineers.

1.2 Acronyms

DB	Database
CCS	Central Control System
ELT	Extremely Large Telescope
FCF	Function Control Framework
FCS	Function Control System
GUI	Graphical User Interface
ICS	Instrument Control System
IFW	ICS Framework
PLC	Programming Logical Controller

1.3 Overview

The framework components included in version 3 are listed in the release notes *Summary*.

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

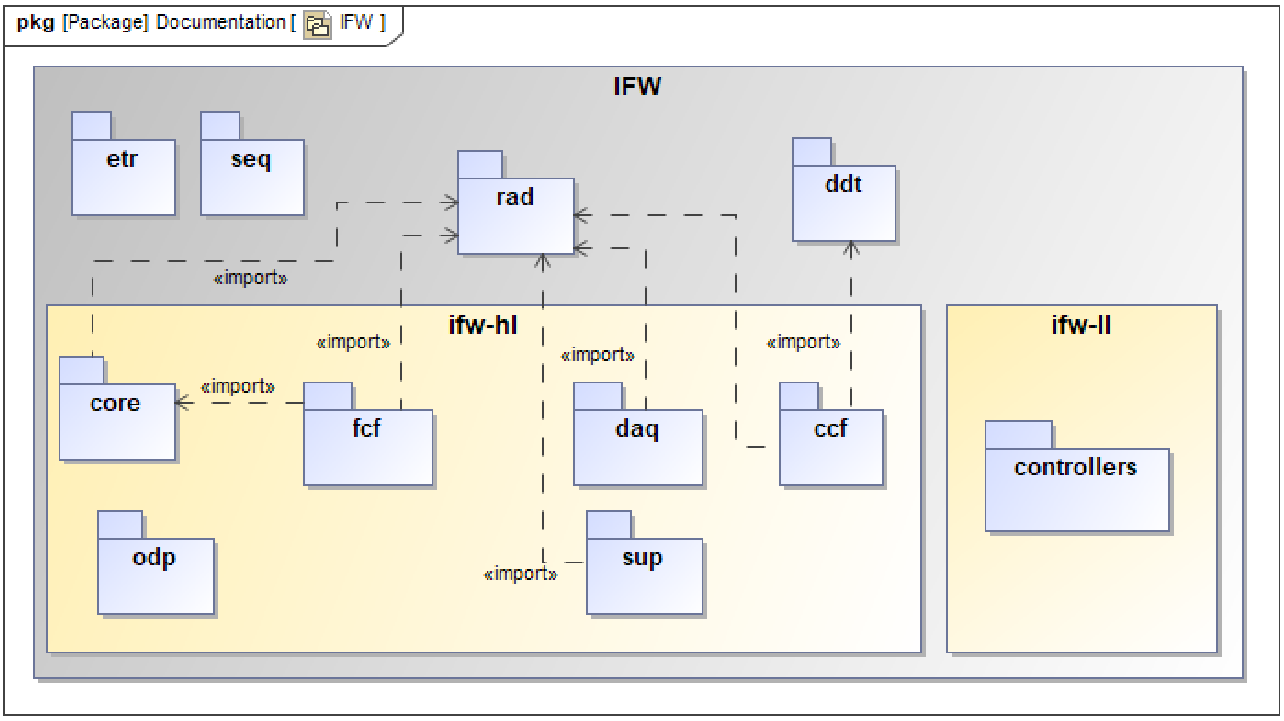


Fig. 1.1: IFW GIT packages and their main dependencies.

Note: The ifw-hl contains most of the IFW code and configuration files while the ifw-ll stores mainly PLC Visual Studio projects.

Warning: *Disclaimer:*

ESO does not warrant that the functions contained in version 3 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 3 are not subject to changes in future releases. The future upgrade of ICS Framework to the Core Integration Infrastructure (CII) may introduce significant modifications to the actual interfaces and services.

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 to the ELT Project and ruled by the "General Conditions of ESO Contracts". Any other use is not

2 Release Notes

2.1 Summary

This release includes the following products:

ICS Framework Product	Repository/Waf project	Product version	RPM version
Application Framework	rad	3.0.0	elt-rad-3.0.0
Test Runner Framework	etr	3.0.0	elt-etr-3.0.0
Sequencer	seq	2.0.0	elt-seq-2.0.0
Data Display Tool	ddt	0.1.0 (beta)	elt-ddt-0.1.0
Camera Control Framework	ifw-hl/ccf	1.0.0	elt-ifw-3.0.0
Observation Coordination Framework	ifw-hl/daq ifw-hl/sup	1.0.0	elt-ifw-3.0.0
Function Control Framework	ifw-hl/fcf	3.0.0	elt-ifw-3.0.0
Online Data Processing	ifw-hl/odp	2.0.0	elt-ifw-3.0.0
Miscellaneous Core Libraries	ifw-hl/core	3.0.0	elt-ifw-3.0.0

The links to the components release in Gitlab are :

- ifw-hl release (fcf, odp, ccf, sup, daq and core)[1]
- ifw-ll release (fcf controllers)[2]
- rad release[3]
- etr release[4]
- seq release[5]

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).

[1] https://gitlab.eso.org/ifw/ifw-hl/-/releases
[2] https://gitlab.eso.org/ifw/ifw-ll/-/releases
[3] https://gitlab.eso.org/ifw/rad/-/releases
[4] https://gitlab.eso.org/ifw/etr/-/releases
[5] https://gitlab.eso.org/ifw/seq/-/releases

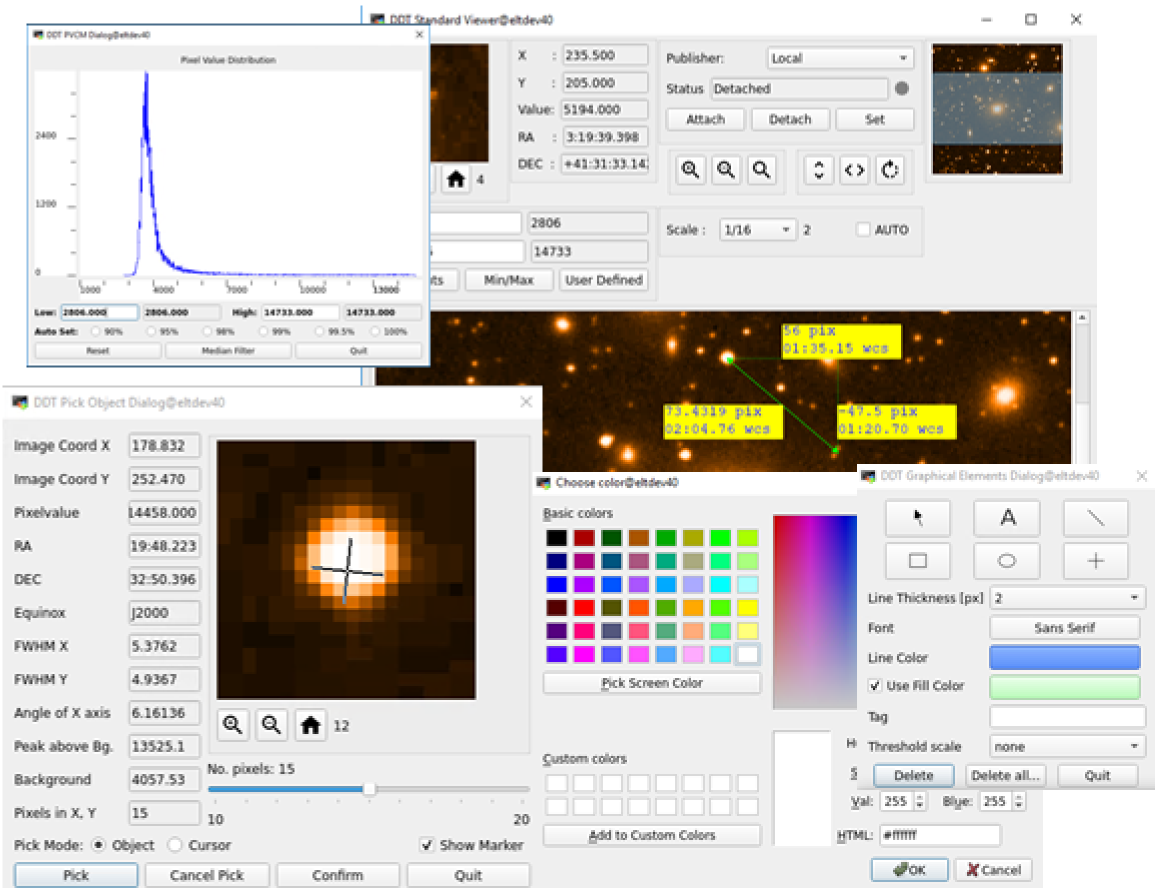
What is the scope of this version?

Version 3 responds to the ICS Framework release strategy where the framework is delivered incrementally. This release add important components to the Framework although not all requirements are necessarily met. However, this version tries to cover the implementation of the most important design choices and what is pending will be delivered in the coming versions (major or minor) of the Framework. This integration will continue in the future versions of the ICS Framework according to the availability of the CII services.

What is new in this release?

This release includes important new components.

- Beta version of the Data Display Tool (DDT)
- First version of the Camera Control Framework (CCF).
- First version of some components of the Observation Coordination Framework (OCF) like the System Supervisor and the Data Acquisition.



X : 235.500
Y : 205.000
Value: 5194.000
RA : 3:19:39.398
DEC : +41:31:33.14
Publisher: Local
Status: Detached
Scale : 1/16 2 AUTO
2806
14733
Low: 2806.000 2806.000 High: 14733.000 14733.000
Auto Set: 90% 95% 98% 99% 99.5% 100%

Image Coord X 178.832
Image Coord Y 252.470
Pixelvalue 14458.000
RA 19:48.223
DEC 32:50.396
Equinox J2000
FWHM X 5.3762
FWHM Y 4.9367
Angle of X axis 6.16136
Peak above Bg. 13525.1
Background 4057.53
Pixels in X, Y 15
No. pixels: 15
Pick Mode: (●) Object ( ) Cursor ✓ Show Marker

Line Thickness [px] 2
Font Sans Serif
Line Color
✓ Use Fill Color
Tag
Threshold scale none
Val: 255 Blue: 255
HTML: #ffffff

Fig. 2.1: DDT Viewer overview.

Besides the above new components, additional changes are present in this version:

- IFW has been ported to CentOS 8 following the change in the Development Environment.
- IFW has been modified to support multiples CII MAL interfaces. All our components implement the ELT standard interface (stdif).
- IFW has adopted experimentally Nomad/Consul for deploying software components.
- IFW is providing a project template which includes startup/shutdown scripts.
- FCF has adopted a simplified schema for providing extendability based on a JSON serialisation. This removed the actual limitation to deploy special devices in separate Device Managers.
- FCF has included a Command Line Interface (CLI) to simplify the interaction with the system for users.
- FCF PLC libraries moved from SVN to GIT.

What is the purpose of the IFW release?

- Provide the updated ELT Development Environment to consortia software developers, to allow them to stay aligned with the latest developments and gain hands-on experience with the new technologies adopted by the ELT.
- Facilitate an early feedback from developers in order to correct issues and modify the design/implementation as early as possible.
- Give consortia the tools for the control of hardware functions that will allow them to carry out prototyping activities.

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

- Qt Framework[6] for developing GUIs.
- ZeroMQ[7] for implementing request/reply and publish/subscribe message patterns.
- Google Proto Buffers[8] for serialization/deserialization.
- Softing OPC-UA toolkit[9] for communication to the PLCs
- FreeOpcUa toolkit[10] for implementing OPCUA servers in Python.
- Google Tests and Mockup libraries[11] for unit tests.
- Robot Framework[12] for integration tests.

[6] https://www.qt.io/
[7] http://zeromq.org/
[8] https://developers.google.com/protocol-buffers/
[9] https://industrial.softing.com/products/opc-opc-ua-software-platform/opc-server-middleware.html
[10] https://github.com/FreeOpcUa/python-opcua
[11] https://github.com/abseil/googletest
[12] http://robotframework.org/

- Nomad/Consul[13] for software deployment.
- JSON format[14] 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 CentOS 8.
- The Development Environment includes the standard MAL interface modules.

For a detail release notes of the Development Environment, please visit here[15]

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.

At present, the Rapid Application Development uses the following libraries:

Service	Description
Error Handling	Exceptions
Configuration	Based on files using YAML
Messaging	Req/Rep and Pub/Sub using CII MAL ZPB
Logging	Based on Log4cplus
Online-DB	Redis in-memory key/value DB

For details, see the *rad* user manual.

[13] https://www.nomadproject.io/
[14] https://www.json.org/json-en.html
[15] https://www.eso.org/~eltmgr/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)

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.
- *Sensors*
  - Monitor a list of engineering variables.
  - Values are not continuously read from the LCS but updated on data change.
- *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[16] and the “rad/scxml4py” engine.

[16] https://github.com/FreeOpcUa/python-opcua

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 (`pymotgui`): 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 retreived from the ifw-resource repository[17]. 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 retreived from: https://gitlab.eso.org/ifw/ifw-ll/-/releases/v3.0.0

The following table lists the versions of the PLC libraries:

Library	Version	What is new in Release 3.0
ioDev.library	1.1.1.0	
lamp.library	1.1.1.1	
motor.library	4.2.2.6	Addition of control parameter for MOVE REL. Updated Control GUI.
piezo.library	0.6.6.2	
rsComm*.library (4 libraries)	0.2.0.0	Restructuring of the module. Addition of simulation for serial and USB devices. Addition of read function 4 for MODBUS (in addition to function 3 that is default).
shutter.library	1.1.0.1	
timer.library	1.0.2.1	
actuator.library	1.0.0.2	

All libraries have been created with **TwinCAT 3.1.4024.7.**

Required TwinCAT 3.1 build is 4024.7 or higher.

Required OPC UA Server version is 3.3.16.0 or higher.

[17] https://gitlab.eso.org/ifw/ifw-resource

PLC Related Incompatibility Issues and Improvements

- With the introduction of serial and USB simulators in module rsComm, it is now required to explicitly pass command and reply suffix strings to the controller, so they could be further passed to the incorporated simulator. In other words, there are no more default suffix strings in rsComm Function Blocks for controllers with ASCII serial interfaces.

```
PROGRAM MAIN
VAR
    // Lakeshore 340 (RS-232 interface)
    {attribute 'OPC.UA.DA' := '1'}
    LS340:    FB_LAKESHORE_RS;

END_VAR
```

```
// Old Release 2.0 format - DO NOT USE!
LS340(
    in_sName                := 'Lake 340',
    in_nModel               := 340,
    in_bAutoMonitor         := TRUE,
    in_nPeriod              := 1700);

// Release 3.0 format
LS340(
    in_bSimulation          := FALSE,
    in_sName                := 'Lake 340',
    in_nModel               := 340,
    in_bAutoMonitor         := TRUE,
    in_sCmdSuffix             := '$0D$0A',
    in_sReplySuffix           := '$0D$0A',
    in_nPeriod              := 1700);
```

- The list and the order of the Lakeshore commands to be periodically executed has changed. For the list of commands, see the declaration part of method M_Configure() of FB_LAKESHORE_RS, FB_LAKESHORE_TCP and FB_LAKESHORE_TCP_RT. In addition, the readings are also extracted into dedicated status variables, see T_LAKESHORE_STAT.

2.5 Known Problems

- CII spurious messages: Latest version of CII MAL produces what it seems to be a harmless message. So far we have not identified any impact due to this in our applications.

```
Could not inform listener about ZpbMalEvent::UNREGISTERED since lock on its weak␣
↪pointer failed
tcp://*:5100   dealer ID: de 13 b3 f3 cc 4b 3b 38
```

- CII logger error: Latest version of CII MAL complains about log4cplus initialization. This does not have a known effect in the applications.

```
log4cplus:ERROR No appenders could be found for logger (malZpbServer).
log4cplus:ERROR Please initialize the log4cplus system properly.
```

- CII URI Parsing: An URI which contains an extra '/' is not handled properly and clients may hang.
- Sequecer Server: The sequencer server might detect some not recognized JSON stataments from templates scripts and issue an error in its standard output. This should not affect the normal execution of the templates.

```
KeyError: 'key'
ERROR:(seqsh.py.proc_reader:99) - Reading subproc error
Traceback (most recent call last):
  File "/home/psivera/INTROOT/lib/python3.7/site-packages/seq/cli/seqsh.py",␣
↪line 75, in proc_reader
    if not d["key"] in self.keywords:
```

3 Installation

3.1 Machine Preparation

Install a real or virtual machine according to the Linux Installation Document[18]

Note: use as ELT_ROLE the default: ELT_ROLE=BASE

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[19]

Note: The version **3.1.8** of ELT Development Environment shall be used with version 3 of ICS Framework (IFW).

Warning: The ELT Development Environment has been upgraded to Centos8 so the WS needs to be reinstalled from scratch.

3.4 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 3.1.8.

It is possible to start/shutdown the software from another user but it requires setting properly the environment variables and the permissions for shared folders like INTROOT and DATAROOT.

[18] http://www.eso.org/~eltmgr/ESO-287339_5_2%20ELT%20Linux%20Installation%20Guide.pdf
[19] http://www.eso.org/~eltmgr/ESO-288431_4%20Guide%20to%20Developing%20Software%20for%20the%20EELT.pdf

3.5 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.

3.6 Getting Started

1. Install IFW components

Starting from IFW version 3, all IFW components are available as RPMs and can be installed using yum install as **root**.

```
$ yum -y install elt-ifw
```

This command will install the ifw RPM but also the other components rad, seq, ddt, etr RPMs :

- elt-ifw-3.0.0
- elt-ddt-0.1.0
- elt-etr-3.0.0
- elt-rad-3.0.0
- elt-seq-2.0.0

2. Login as eltdev user.
3. 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
$ cd DATAROOT
$ mkdir image
$ mkdir data
```

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

4. Under eltdev home directory:

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

5. 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 ("ifw")
load ("introot")
```

Note: Each IFW components RPMs provide a lua file to define the minimum set of default environment variables. Loading ifw.lua will also load all IFW components lua files (seq, rad, etr, ddt).

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/.../
↪modulefiles ---------------------------------------------------------------------
   private (L)

-------------------------------------------------------------------- /elt/System/
↪modulefiles ---------------------------------------------------------------------
```

(continues on next page)

(continued from previous page)

```
   ciisrv/ciisrv  (L)    cpl  (L)     doxygen  (L)     etr  (L)      introot  (L)    ␣
↪       msgpack  (L)    opentracing  (L)     seq  (L)
   clang                  czmq  (L)     ecsif    (L)     gcc/9  (L)     jdk/java-openjdk␣
↪(L)     nix/2.3  (L)    python/3.7   (L)     shiboken2  (L)
   consul           (L)    ddt  (L)     eltdev   (L)     ifw  (L)      mal/mal         ␣
↪(L)     nomad    (L)    rad  (L)            slalib_c   (L)

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

  Where:
   L:  Module is loaded
```

Note: For more information, read this document[20]

3.7 Start nomad and consul services

The IFW adopted experimentally Nomad (see here[21]) to manage the life cycle of the ICS SW components (see next chapter Getting Started). The Nomad and Consul RPMs installed within the ELT Development Environment, provide a default configuration to start Nomad and Consul in one node cluster using a loopback interface. Nomad and Consul services has to be started as the eltdev user with the command systemctl.

Warning: For Nomad to work properly, the environment under eltdev user shall be defined with the same LMOD configuration as the running user (see *elmo_installation_ref*).

- 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

[20] http://www.eso.org/~eltmgr/ESO-288431_4%20Guide%20to%20Developing%20Software%20for%20the%20EELT.pdf
[21] https://www.nomadproject.io/

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

3.8 Manual installation

As before the 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 from 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.

- Download, unpack and build RAD component

Go to the ESO Gitlab site and download the tar file of the component: RAD release[22]

After unpacking the rad package downloaded from ESO GitLab, execute the steps below to build and install the software.

```
$ cd rad
$ waf configure
$ waf build install
```

- Download, unpack and build IFW-HL component

Go to the ESO Gitlab site and download the tar file of the component: IFW-HL release[23]

Note: This tar contains core, ccf, ocm, fcf and odp WAF projects.

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

```
$ cd ifw-hl/core
$ waf configure
$ waf build install
$ cd ifw-hl/fcf
$ waf configure
$ waf build install
....
```

[22] https://gitlab.eso.org/ifw/rad/-/releases
[23] https://gitlab.eso.org/ifw/ifw-hl/-/releases

3.9 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[24]

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).

[24] 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, see the Installation)

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 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-hl/templates/project
project_name [myproject]: micado
project_description [this is my project description]:
project_prefix: [xxx]: mic
component_name [mycomponent]: fcs
device_name [mydevice]: mirror
```

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
└── 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
```

4.2 Starting/Stopping the ICS Software

The IFW adopted experimentally Nomad (see here[25]) 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. A good introduction to these packages can be found here:

- nomad introduction[26]
- consul introduction[27]

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.

Before describing the startup/shutdown sequence, a quick introduction to Nomad/Consul will be given to familiarize the reader with these new tools introduced in this version.

4.3 Introduction to Nomad/Consul

Basic Configuration and Assumptions

We are still learning how to use in the most efficient way Nomad/Consul. For this version, we have provided a experimental configuration based on some assumptions that are documented hereafter. More complex configurations and deployments is of course possible but that it is left to each system. The idea of the template is to show the concept by providing a simple and representative example that can be easily extended.

1. One Nomad data center is foreseen for the complete instrument. Here we use the default one 'dc1'
2. The Nomad configuration foresees running one Nomad Server and one Nomad Client in one node, e.g. the IWS.
3. Jobs are all deployed locally in the IWS.
4. Nomad and Consul shall be running as systemd services in the machine under the user eltdev.
5. We use raw_exec as the driver to run each of the jobs.

[25] https://www.nomadproject.io/
[26] https://www.youtube.com/watch?v=s_Fm9UtL4YU
[27] https://www.youtube.com/watch?v=mxeMdl0KvBI

6. We have not used Nomad for GUIs
7. Ports are assigned randomly by Nomad.
8. We are using only a python library to query the consul service discovery.
9. We have used the Template Stanza for handling the automatic discovery of services in configuration files.
10. We are using signals to communicate changes of port numbers at run-time.

Nomad Jobs

The job configuration files uses the HashiCorp Configuration Language (HCL) to specify the details of each job. For more details about HCL can be found here[28].

A Nomad job configuration file is shown in the example below. This is the configuration for the DDT broker job included in the coockiecutter template. This file also contains the coockiecutter tags that will be used to customize it for each instrument.

```
job "{{cookiecutter.project_prefix}}broker" {
    datacenters = ["dc1"]

    group "{{cookiecutter.project_prefix}}broker_group" {
        network {
            port "broker" {} // use random port for broker
        }

        // register broker service in Consul
        service {
            name = "{{cookiecutter.project_prefix}}broker"
            port = "broker"
        }

        task "{{cookiecutter.project_prefix}}broker" {
            driver = "raw_exec"

            config {
                        command = "/bin/bash"
                args = ["-c", " ddtBroker --uri zpb.rr://*:${NOMAD_PORT_broker}/
↪broker"]
                }
         }
     }
 }
```

[28] https://www.nomadproject.io/docs/job-specification

Nomad GUI

The Nomad package includes a Web UI where to display the jobs and the Nomad Configuration. The Web UI runs on a browser under a http://localhost:4646. You can start firefox and connect to this page. For more details visit the following page:[29].

Consul Service Discovery

Consul is a tool for discovering and configuration services. We have used it to provide service discovery for our applications.

For servers, we are using the template Stanza[30] to automatically retrieve the service information from configuration files. These templates files are rendered using the information provided by Consul when starting each Nomad job. The rendered files are saved in local directories managed by Nomad.

For scripts and GUIs, we use the Consul python client library (Python) to query the address and port of each service. For the command line, we provide a very simple utility (*geturi*) that will format a valid CII MAL URI to be used by other applications. This utility is using the consul python library internally to retrieve the address and the port from a given service name, see an example below.

```
> geturi fcs-req
zpb.rr://134.171.3.48:30043
> fcfClient `geturi fcs-req` GetStatus
Operational;Idle
```

In case the service is not available, you get an exception.

```
> geturi pippo
...
ValueError: ERROR: service not found in consul db: pippo
```

For python scripts, we provide a very simple utility class using the REST API of consul. See the example below. This class provides already the formatted URI that is used by the CII MAL clients.

```
import stooUtils.consul as consul_utils
cons = consul_utils.ConsulClient()
""" Get formatted URI of Supervisor service """
uri = cons.get_uri("syssup-req")
""" Create instance of the Supervisor client """
supif = sup.SysSupCommands(uri, timeout=2000)
if not supif.is_operational():
    raise Exception('SW is not operational !')
```

[29] https://learn.hashicorp.com/tutorials/nomad/get-started-ui?in=nomad/get-started
[30] https://www.nomadproject.io/docs/job-specification/template

Consul GUI

As for Nomad, Consul also provides a web based UI. You can use UI to display the active services that will be registered automatically when starting the Nomad jobs. The Web UI runs on a browser under a http://localhost:8500/ui. For more details visit the following page:[31].

Nomad Templates

In order to use the service discovery from Consul, we needed to adapt the configuration files to include the look-up of the service information. Nomad defines a way to do that in the job definition that it integrates with the consul template. When the job is started, the template is rendered and saved under the Nomad allocation directory. Then the processes uses the rendered files just like they normally do without Nomad. In this context, rendering means that we get the information from Consul to compose the URI, see the example below. This is a part of the content of a typical configuration.

```
req_endpoint     : "zpb.rr://{{ range service "fcs-req" }}{{ .Address }}:{{ .Port␣
↪}}{{ end }}/"
pub_endpoint     : "zpb.ps://{{ range service "fcs-pub" }}{{ .Address }}:{{ .Port␣
↪}}{{ end }}/"
```

Here the template does a query for a particular service in Consul and from the answer it gets the address and port. After the rendering process, the files are generated under <alloc>/<task>/local and the above contents are replaced by something like the following.

```
req_endpoint     : "zpb.rr://134.171.3.48:30043/"
pub_endpoint     : "zpb.ps://134.171.3.48:20054/"
```

Note: The ports numbers are unique and generated by Nomad.

How to start/stop a job from the command line

The sequencer can be used to start/stop all jobs. You could also select individual ones to stop/start if needed. There are dependencies between the jobs that need to be consider when starting them. The sequencer script defines the right sequence. However you can also use the Nomad CLI to start/stop each job from the command line.

As an example, you can see how to start/stop a given job from the command line:

Warning: Do not execute this command yet otherwise you will get an error.

[31] https://learn.hashicorp.com/tutorials/consul/get-started-explore-the-ui

```
> cd micado/resource/nomad
> nomad job run fcs.nomad
==> Monitoring evaluation "f5d1514b"
    Evaluation triggered by job "fcs"
    Evaluation within deployment: "6aea79eb"
    Allocation "93945d7d" created: node "885603af", group "fcs_group"
    Evaluation status changed: "pending" -> "complete"
==> Evaluation "f5d1514b" finished with status "complete"
```

One could check the status of the job with the Web UI or with the following command:

```
> nomad status fcs
ID            = fcs
Name          = fcs
Submit Date   = 2021-03-25T13:31:15Z
Type          = service
Priority      = 50
Datacenters   = dc1
Namespace     = default
Status        = running
Periodic      = false
Parameterized = false

Summary
Task Group  Queued  Starting  Running  Failed  Complete  Lost
fcs_group   0       0         1        0       1         0

Latest Deployment
ID          = 6aea79eb
Status      = successful
Description = Deployment completed successfully

Deployed
Task Group  Desired  Placed  Healthy  Unhealthy  Progress Deadline
fcs_group   1        1       1        0          2021-03-25T13:41:25Z

Allocations
ID        Node ID   Task Group  Version  Desired  Status    Created    Modified
93945d7d  885603af  fcs_group   2        run      running   3m26s ago  3m16s ago
f7826c80  885603af  fcs_group   0        stop     complete  23h12m ago  3m35s ago
```

Stopping the FCS from the command line:

```
> nomad job stop fcs
```

Note: You can stop any job that is currently running with its name registered in Nomad.

How to get the Job standard output

Navigating through the Nomad UI you can access the logs of each process (stderr, and stdout). However an easier way can be done using the command line (Nomad CLI). More details can be found here[32].

```
> nomad alloc logs -f -job <job>
```

4.4 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:

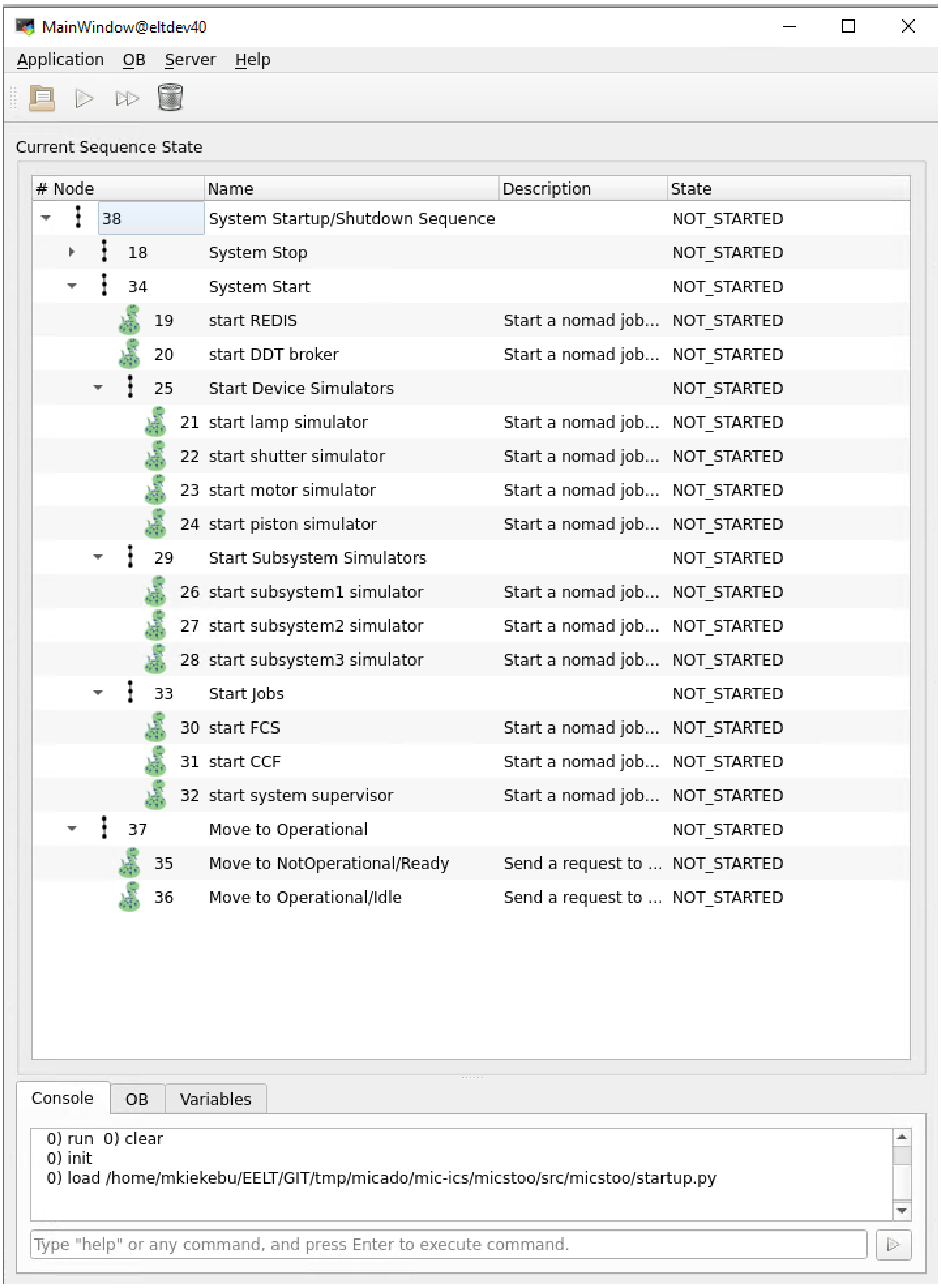
- Redis instance
- DDT broker
- CCF instance with Simulator and DDT publisher
- FCF Simulators (shutter, lamp and motor)
- Subsystem Simulators (subsim1, subsim2 and subsim3)
- Custom FCF instance with custom device.
- Custom FCF simulator (mirror)
- OCM instance
- System Supervisor

These components are obviously using simulators and not real hardware. The script shall be executed by the Sequencer. The Sequencer requires to have an instance of Redis running.

The script contains three main parts:

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

[32] https://www.nomadproject.io/docs/commands/alloc/logs



# Node	Name	Description	State
38	System Startup/Shutdown Sequence		NOT_STARTED
18	System Stop		NOT_STARTED
34	System Start		NOT_STARTED
19	start REDIS	Start a nomad job...	NOT_STARTED
20	start DDT broker	Start a nomad job...	NOT_STARTED
25	Start Device Simulators		NOT_STARTED
21	start lamp simulator	Start a nomad job...	NOT_STARTED
22	start shutter simulator	Start a nomad job...	NOT_STARTED
23	start motor simulator	Start a nomad job...	NOT_STARTED
24	start piston simulator	Start a nomad job...	NOT_STARTED
29	Start Subsystem Simulators		NOT_STARTED
26	start subsystem1 simulator	Start a nomad job...	NOT_STARTED
27	start subsystem2 simulator	Start a nomad job...	NOT_STARTED
28	start subsystem3 simulator	Start a nomad job...	NOT_STARTED
33	Start Jobs		NOT_STARTED
30	start FCS	Start a nomad job...	NOT_STARTED
31	start CCF	Start a nomad job...	NOT_STARTED
32	start system supervisor	Start a nomad job...	NOT_STARTED
37	Move to Operational		NOT_STARTED
35	Move to NotOperational/Ready	Send a request to ...	NOT_STARTED
36	Move to Operational/Idle	Send a request to ...	NOT_STARTED

```
0) run  0) clear
0) init
0) load /home/mkiekebu/EELT/GIT/tmp/micado/mic-ics/micstoo/src/micstoo/startup.py
```

Type "help" or any command, and press Enter to execute command.

Fig. 4.1: Startup/Shutdown script in the Sequencer.

4.5 Executing Startup/Shutdown Script

This requires to start Redis Server first because the Sequencer uses Redis to exchange information between the server and the GUI.

Starting Redis Server

```
redis-server &
```

Note: When no port is specified, the Redis DB server uses the port 6379.

Starting Sequencer Server

In one terminal type the following command to start the sequencer server.

```
> seq server
```

Starting Sequencer GUI

In another terminal type the following command to start the sequencer GUI.

```
> seq gui
```

Running the Startup Script

Once Redis, Sequencer Server and Sequencer GUI are running. Load the startup script (micado/mic-ics/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.

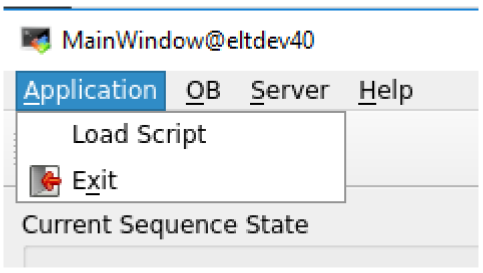
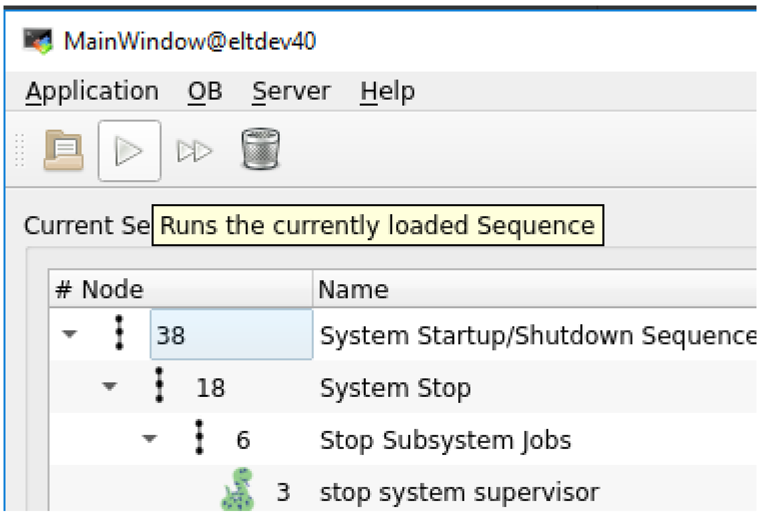


Fig. 4.2: Load script option from Sequencer File menu.

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.



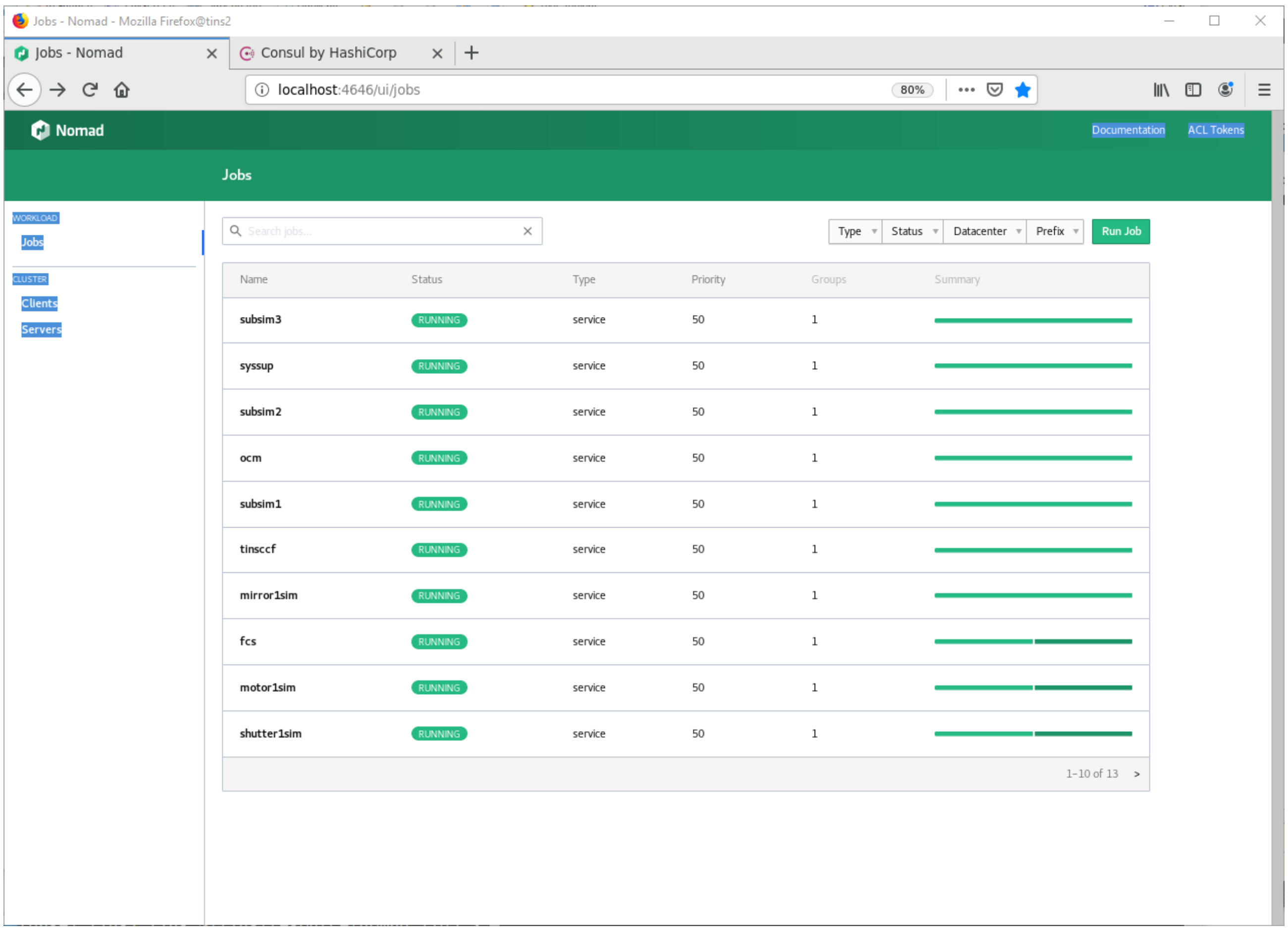
# Node	Name
38	System Startup/Shutdown Sequence
18	System Stop
6	Stop Subsystem Jobs
3	stop system supervisor

Fig. 4.3: 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 13 jobs shall be running.



Jobs - Nomad - Mozilla Firefox@tins2

Jobs - Nomad | Consul by HashiCorp

localhost:4646/ui/jobs

Nomad — Documentation — ACL Tokens

Jobs

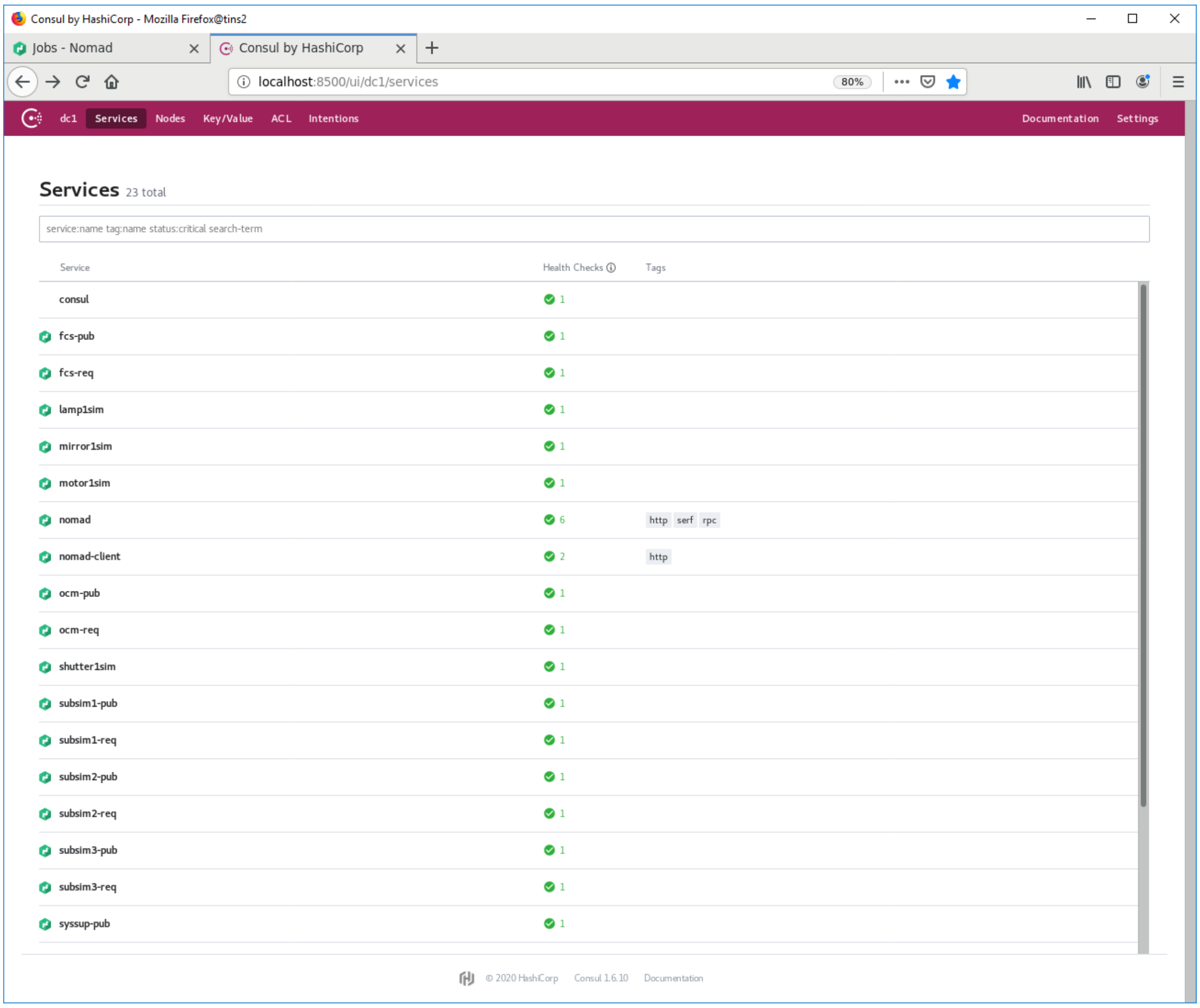
WORKLOAD: Jobs
CLUSTER: Clients, Servers

Name	Status	Type	Priority	Groups	Summary
subsim3	RUNNING	service	50	1	
syssup	RUNNING	service	50	1	
subsim2	RUNNING	service	50	1	
ocm	RUNNING	service	50	1	
subsim1	RUNNING	service	50	1	
tinsccf	RUNNING	service	50	1	
mirror1sim	RUNNING	service	50	1	
fcs	RUNNING	service	50	1	
motor1sim	RUNNING	service	50	1	
shutter1sim	RUNNING	service	50	1	

1–10 of 13

Fig. 4.4: 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.



Consul by HashiCorp - Mozilla Firefox@tins2

localhost:8500/ui/dc1/services

Services 23 total

service:name tag:name status:critical search-term

Service	Health Checks	Tags
consul	✓ 1	
fcs-pub	✓ 1	
fcs-req	✓ 1	
lamp1sim	✓ 1	
mirror1sim	✓ 1	
motor1sim	✓ 1	
nomad	✓ 6	http serf rpc
nomad-client	✓ 2	http
ocm-pub	✓ 1	
ocm-req	✓ 1	
shutter1sim	✓ 1	
subsim1-pub	✓ 1	
subsim1-req	✓ 1	
subsim2-pub	✓ 1	
subsim2-req	✓ 1	
subsim3-pub	✓ 1	
subsim3-req	✓ 1	
syssup-pub	✓ 1	

© 2020 HashiCorp Consul 1.6.10 Documentation

Fig. 4.5: List of TINS Consul Services.

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
```

(continues on next page)

(continued from previous page)

```
 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. If you run the startup/shutdown script under a different user than eltdev, make sure that Nomad (running under eltdev) can read the template files. These files are read by Nomad during the rendering process. If your problems persists try to run the startup/shutdown script under the eltdev account.

3. 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.

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 setup FCF and CCF for the upcoming observation template that takes an image with the simulator.

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).

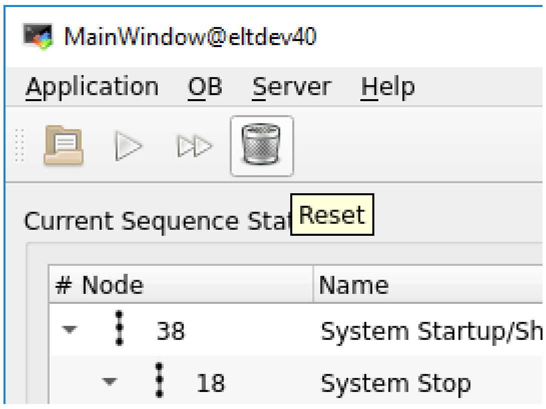


Fig. 4.6: Clear the current script and reset the server.

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/resource/seq/tec/MICADO_OB_sample.json.

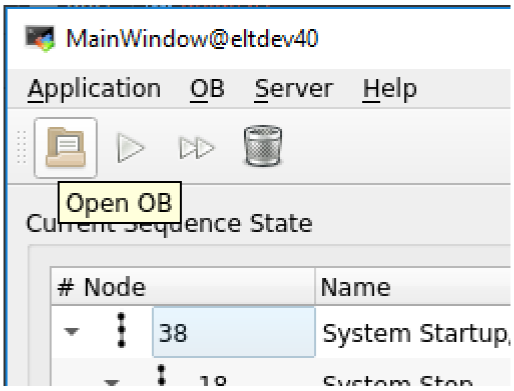
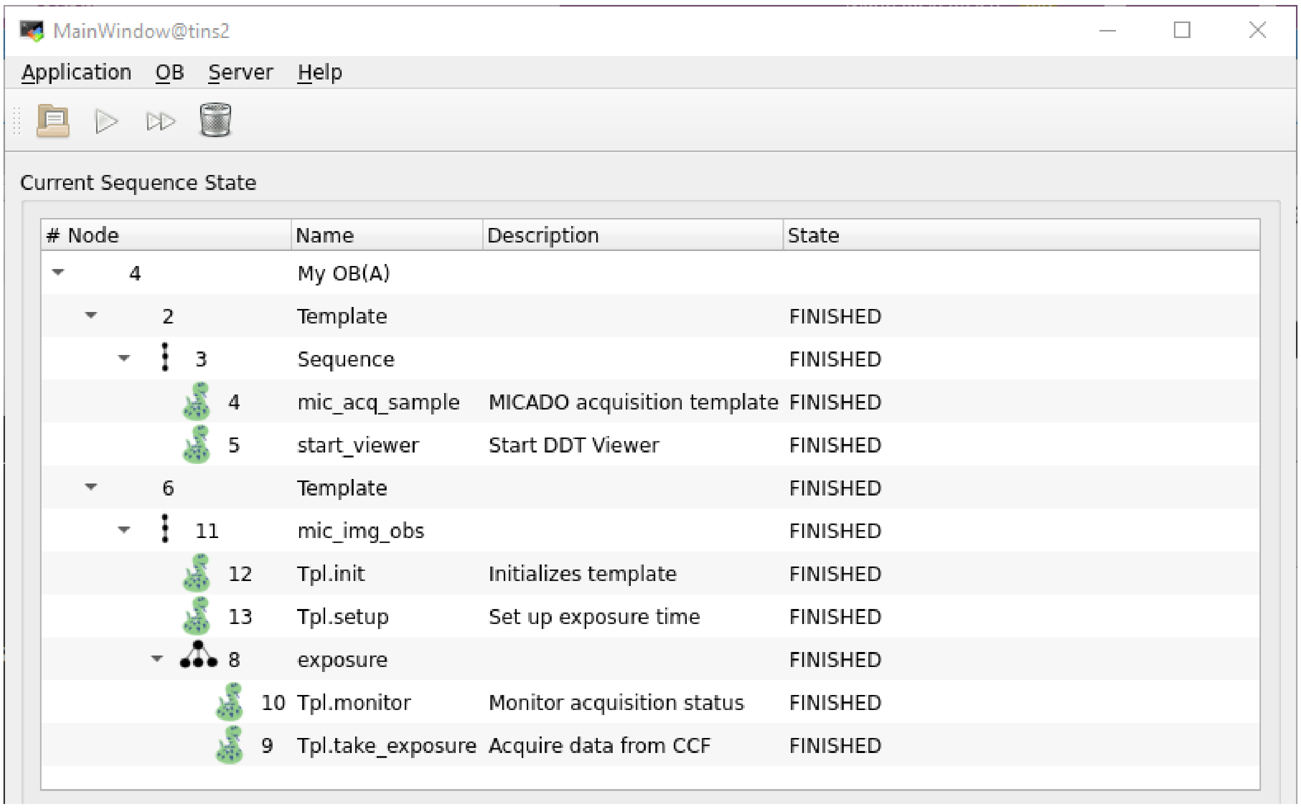


Fig. 4.7: Load an OB.

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



# Node	Name	Description	State
4	My OB(A)		
2	Template		FINISHED
3	Sequence		FINISHED
4	mic_acq_sample	MICADO acquisition template	FINISHED
5	start_viewer	Start DDT Viewer	FINISHED
6	Template		FINISHED
11	mic_img_obs		FINISHED
12	Tpl.init	Initializes template	FINISHED
13	Tpl.setup	Set up exposure time	FINISHED
8	exposure		FINISHED
10	Tpl.monitor	Monitor acquisition status	FINISHED
9	Tpl.take_exposure	Acquire data from CCF	FINISHED

Fig. 4.8: 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.

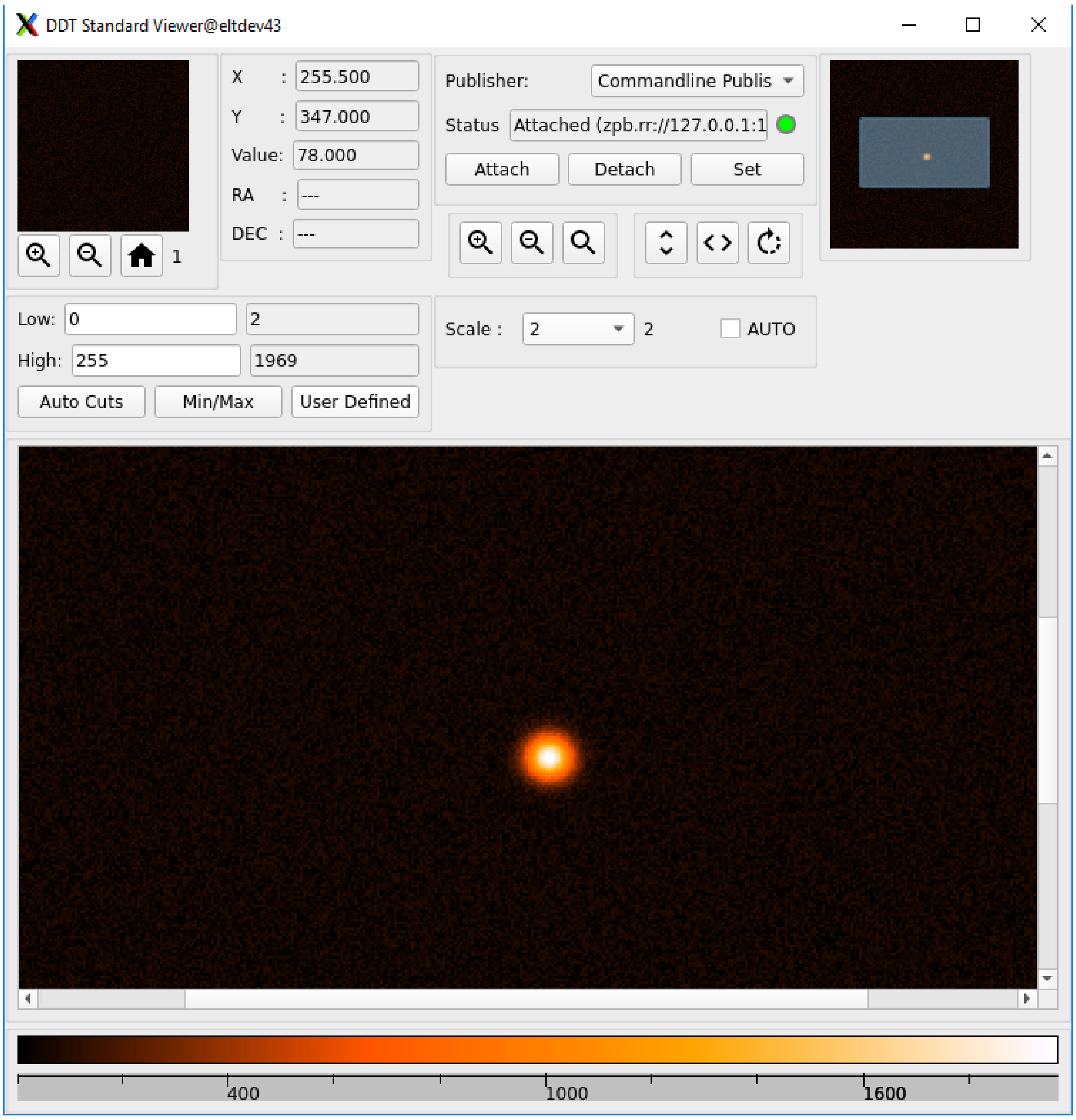


Fig. 4.9: DDT Viewer with image received from CCF.

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.