European Organisation for Astronomical Research in the Southern Hemisphere

Programme: ELT

Project/WP: Instrumentation Framework

ELT ICS Framework - Calibration Framework - User Manual

Document Number: ESO-547361

Document Version: 1

Document Type: Manual (MAN)

Released on: 2024-12-11

Document Classification: Public

Owner:	Reinero, Claudio
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-calob[1] v0.1.3.

Authors

Name	Affiliation
Reinero, Claudio	ESO/DOE/CSE
Knudstrup, Jens	ESO/DOE/CSE

Change Record from Previous Version

Affected Section(s)	Changes / Reason / Remarks
All	First version

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

Contents

1 Introduction

The main purpose of the ICS Calibration Framework (CalOb) is to provide an SDK and a tool, based on former, used for integrating the CalOb Configuration for the operations at the various ESO telescopes. Each instrument has a CalOb Configuration.

The tool is used to generate Observation Blocks, which are executed, normally at day-time, to generate the Calibration Data needed to process the Science Data and to monitor the health of the instrumnents.

The CalOb SDK can be used to implement other tools for scheduling calibrations.

The present version of the CalOb, is mostly a re-implementation of the existing VLT CalOb tool in terms of features and functionality. The existing CalOb Tool has been in use for many years at Paranal and is serving the operations well.

The new version of CalOb is implemented in Python and supports the new data formats defined e.g. for configuration files and OBs. Another major difference between the old and the new tool is that the configuration, defining the rules for scheduling the calibrations to be executed, is a Python script, which is interpreted/executed, rather than a static/text-based definition (for VLT, “PAF”).

1.1 Deliverables

The CalOb project provides six main deliverables:

- **CalOb SDK (Common Library):** Library of classes implementing the business logic of the CalOb (*The CalOb SDK*).
- **CalOb Generator Tool:** Command line utility, using the CalOb SDK to process/generate the Daily Calibration OB (*CalOb Generator - Execution - Example*).
- **Real example of a CalOb Configuration:** The CalOb Configuration for an instrument. This is used for demo purposes and for the internal verification of the CalOb code (*Example*).
- **Cookiecutter Template:** Cookiecutter template which is used to initiate the development of a new Calibration Configuration for an instrument (*Initiating a Calibration Config Set for an Instrument*).
- **User Manual:** This document.
- **Doxygen Documentation:** The standard Doxygen documentation, as provided for all IFW software components. Note, no specific references are made into the CalOb Doxygen online documentation. The reader is requested to navigate in this to find the appropriate information.

1.2 Scope

The scope of this manual is developers, integrating CalOb solutions for the instruments, based on the CalOb package. Furthermore for the operations personnel, using the CalOb tool on a daily basis to understand the functionality.

1.3 Disclaimer

The current version mostly a re-implementation of the existing CalOb (Tcl) tool, in use at Paranal. An additional feature may be added at a later stage, whereby QCFlow provides a service to retrieve accurate information about which calibration data is missing. This will allow CalOb to schedule *exactly* Calibrations needed.

The test data set provided is a hybrid between FORS2 and the upcoming FORSUP data. As soon as FORSUP starts producing real data, the data set will be updated to match the actual FORSUP configuration.

1.4 Acronyms

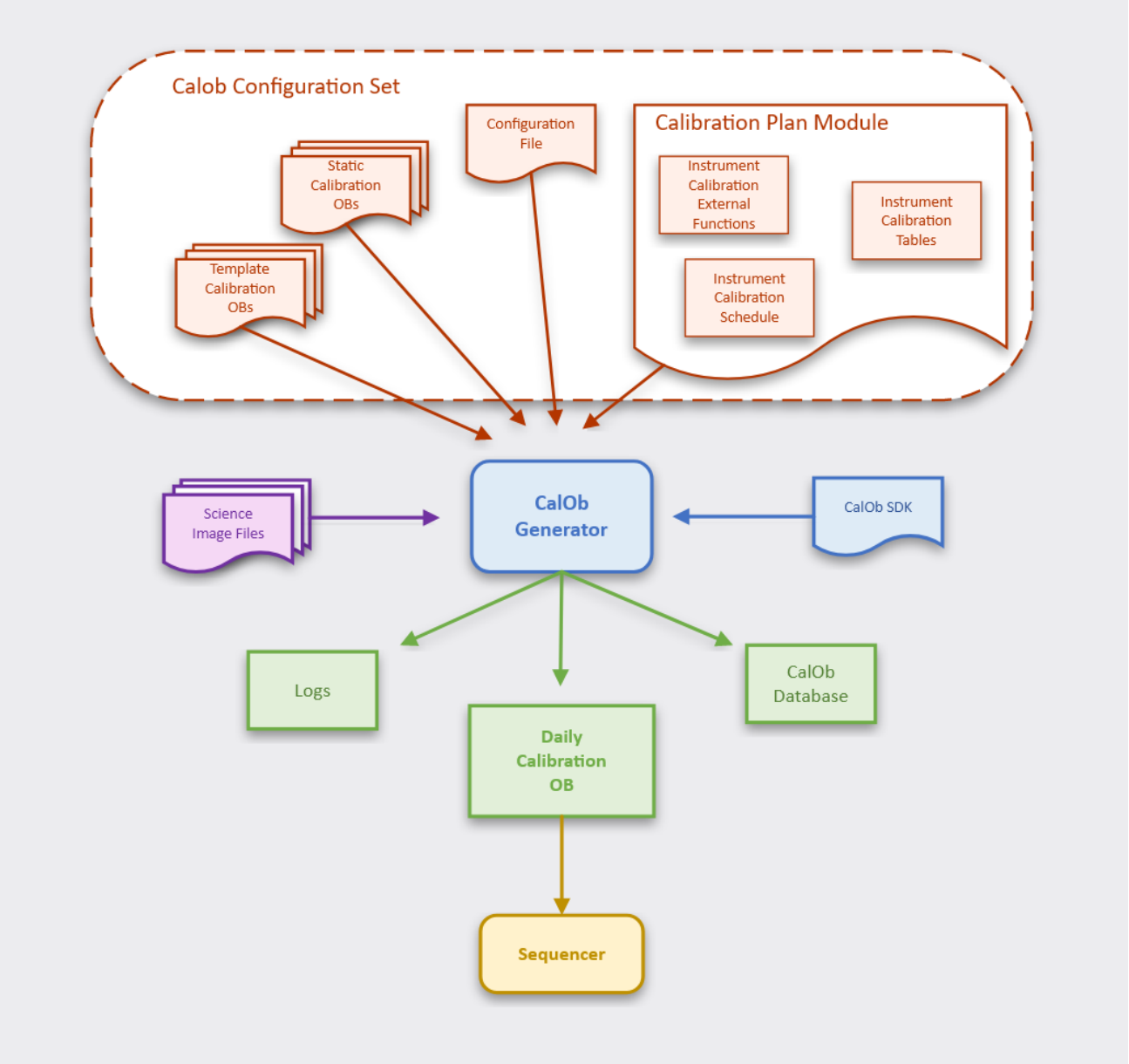
DB	Database
CALOB	Calibration Framework
ELT	Extremely Large Telescope
FITS	Flexible Image Transportation System
HW	Hardware
ICS	Instrument Control System
OB	Observation Block
SDK	Software Development Kit
SW	Software

1.5 Nomenclature

Built-In Function	Function used to define rules for scheduling Calibration Observations. This category of function are provided by CalOb.
Calibration Data	Data needed to carry out the data reduction/analysis of the science data produced by an instrument.
Calibration Framework	The CalOb tool, used to generate the the Daily Calibration OB for an instrument.
Calibration Observation(s)	The set of observations contained in the Daily Calibration OB.
Calibration Schedule	Python script implemented using the CalOb SDK, following a certain syntax, used to generate the Daily Calibration OB.
CalOb Configuration (Set)	The set of documents to be provided for an instrument to define the properties for scheduling Calibration Observation (*Configuration*).
CalOb Database (DB)	Internal database (SQLite) used to keep track of when the various types of Calibration Observations have been scheduled previously. The location on disk is defined in the Configuration File.
CalOb Generator	Command line utility, based on the CalOb SDK, which is invoked to generate the Daily Calibration OB.
CalOb SDK	SDK providing the classes/features needed to implement the Calibration Plan for an instrument and to execute it.
Configuration File	Contains the necessary properties for invoking the CalOb Generator. It references the data, pertinent to the instruments CalOb Configuration Set. Based on the CII Configuration Service.
Daily Calibration OB	OB generated by the CalOb Framework, normally on a daily basis, to generate the Calibration Data needed for a given set of science data.
External Function(s)	Python functions, contained in a standard Python module, loaded and executed by the CalOb SDK as specified in the Calibration Plan.
Log Properties File	Log Properties File as required by the CII Logging Service (“log4py”).
Science Image File(s)	Image FITS files, generated by the instrument during operation. This data is located on the local disk, in the “DATAROOT” area.
Sequencer	The ELT Sequencer Tool, used to execute OBs.
Static Calibration OB(s)	Calibration OB files, with a fixed set of parameter values. The Static Calibration OBs are also referred to as “OBD”.
Template Calibration OB(s)	Calibration OB files, with a variable set of parameter values, which are resolved by the CalOb SDK, when parsing the Calibration Plan. The Template Calibration OBs are also referred to as “OBDX”

2 Overview

The following diagram provides an overview of the CalOb, indicating the various input and output data types involved:



The elements shown in the diagram above are as follows:

- **CalOb Generator:** The CalOb command line tool, “calobGenerator”, which is invoked with the CalOb Configuration as input to generate the Daily Calibration OB (*CalOb Generator - Execution - Example*).

- **CalOb SDK:** The CalOb SDK, implementing all the services for doing the actual processing to obtain the Daily Calibration OB. It is imported by the CalOb Generator, when this is invoked (*The CalOb SDK*, *Classes*).
- **Configuration File:** The configuration used as input to the CalOb Generator tool, defining the dependencies for the processing. It is implemented in YAML (*Configuration File*).
- **Calibration Schedule:** The Calibration Schedule defines the properties of the various Calibrations to be processed for scheduling during that execution. It is an executable Python script, which is imported and executed by the CalOb SDK by invoking the Python interpreter (*Calibration Schedule*).
- **External Functions:** An External Function is a Python function which is provided for the specific instrument to handle special logic for the given context. They are referenced from within the Calibration Plan (*External Functions*).
- **Static Calibration OBs:** These are OB documents that can be scheduled, based on various conditions, defined in the Calibration Plan, which are based on a fixed set of paremeter values (*Static Calibration OBs*).
- **Template Calibration OB:** These are OB documents that can be scheduled, based on various conditions, defined in the Calibration Plan, which allows for the usage of computer parameter values, in addition to fixed parameter values (*Template Calibration OBs*).
- **Science Image Files:** Science image files (FITS), is the data generated by the instrument during the observations, normally at night-time. In the context of CalOb, it is possible to specify the data set for a given period of time, as basis for the generation of the Daily Calibration Plan. The image files are normally located in the “DATAROOT”.
- **Daily Calibration OB:** The main output from CalOb. The Daily Calibration OB is a standard OB file (JSON) which can be scheduled for execution by the ELT Sequencer to generate the required Calibration Data.
- **CalOb Database:** Internal database (SQLite), which is used to store information about previous days Calibration Observations scheduled. This is used to determine when to schedule certain Calibration Obervations, which are scheduled periodically. The CalOb DB is typically stored in “$DATAROOT/calob/db/calibration_history.db, this is defined in the Configuration File”.
- **Logs:** Logs, generated by CalOb during execution, using the CII Log Service. The Log Properties File is defined in the CalOb Configuration File.
- **Sequencer:** The ELT Sequencer tool, which executes the Daily Calibration OB.

2.1 Daily Calibration OB Processing

The processing to obtain the Daily Calibration OB can be split into the following steps:

2.1.1 1: Input Stage:

During this step the CalOb Configuration File is loaded, as well as the other documents in the CalOb Document Set (*Configuration*).

This includes loading the Calob Configuration File, which is the main entry point to access all data relevant for the execution.

The External Functions, the Calibration Library and Calibration Plan Classes (Python), defined in the specified Instrument Calibration Plan, are instantiated (example[1]).

A scan of data files (FITS) to take into account is carried out, properties defined in the Configuration File for this (folder, filename filter/wildcard, period), and all the FITS header information of the FITS files matching, loaded into a dictionary (Python dict).

2.1.2 2: Processing Stage:

During this step, the method in the Instrument Calibration Plan instance to carry out the processing of the list of candidate Calibration Observations is invoked ("CalibrationPlan.init_calibration_plan()"; example[2]).

First the Static Calibration OBs are processed. The processing involves loading the OB document (JSON) containing the predefined parameters (example[3]).

During the processing, CalOb loops over all FITS files that were found, matching the data file search criteria, and invokes all rules an all files, to generate the Calibration OB template instances in the memory representation of the Daily Calibration OB.

As the Calibration Plan is executable code, no explicit parsing/execution of the contents is needed; the "CalibrationPlan.add_static_calib()" is automatically invoked, including the additional methods to define the conditions for scheduling the Calibration Observation.

For each invocation of "add_static_calib()", the contents of the Static Calibration OB (OBD) is added in a temporary list which will be filtered in the following steps.

The rules to apply for scheduling the Calibration Observation, is based on periodicity of the execution, and matching of keyword values.

The processing of the Template Calibration OBs (OBDX) is done in a similar way, by invoking the method "CalibrationPlan.add_template_calib()" method to do the processing. The same rules functions apply.

[1] https://gitlab.eso.org/ifw/ifw-calob/-/blob/main/common/src/ifw/calob/example/FORSUP_calibration_schedule.py
[2] https://gitlab.eso.org/ifw/ifw-calob/-/blob/main/common/src/ifw/calob/example/FORSUP_calibration_schedule.py
[3] https://gitlab.eso.org/ifw/ifw-calob/-/blob/main/common/resource/obd/ifw/calob/example/FORSUP-dark-monthly.obd.json

The main difference between the Static and the Template Calibration OBs, is that latter may contain expressions for keyword values, which will be resolved by CalOb. The expressions are written in Python code (example[4] - e.g. key “DET.READ.CLKIND”).

2.1.3 3: Filtering Stage:

During the filtering stage, the output OB, contained in memory, is cleaned for redundant Calibrations. In addition the Calibrations are ordered according to the ordering specified in the Calibration Schedule using the values entered for each calibration using *set_group()* and *set_order()*.

If the history feature is enabled, the information about Calibration Observations scheduled, will be stored in the CalOb DB, where the historic about when the various candidate Calibration Observations have been scheduled, is stored.

2.1.4 4: Export Stage:

During this step, the output Daily Calibration OB is generated.

2.1.5 5: Clean-up Stage:

Final clean-up of temporary files etc.

More details, including examples, about the various data formats contained in the CalOb Configuration Set, is presented in the next chapter (*Configuration*).

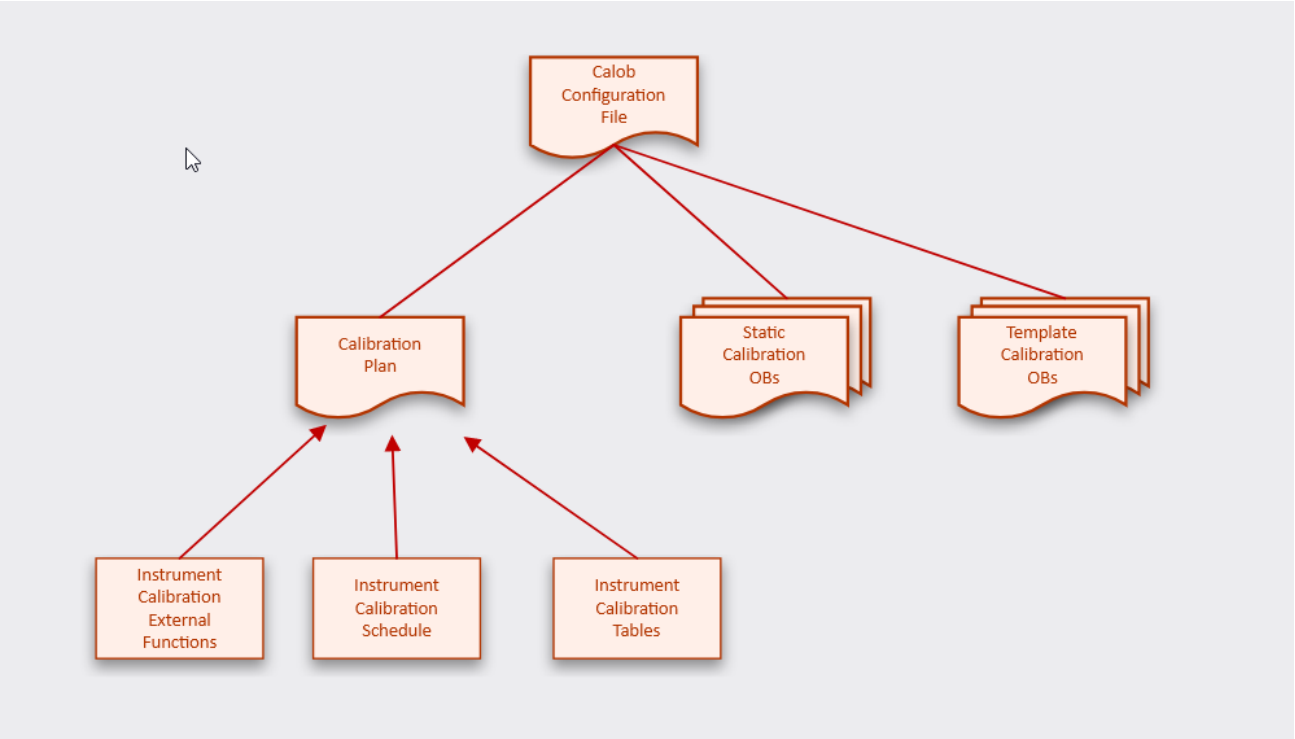
[4] https://gitlab.eso.org/ifw/ifw-calob/-/blob/main/common/resource/obdx/ifw/calob/example/FORSUP_img_cal_bias.obdx.json

3 Configuration

The CalOb Configuration Set consists of the following types of documents:

- Configuration File.
- Calibration Schedule.
- External Functions.
- Static Calibration OBs.
- Template Calibration OBs.

These documents shall be provided in order to define the CalOb scheduling properties for a given instrument. The documents are described below, providing examples.



3.1 Configuration File

An example of the configuration is shown in the following:

```
!cfg.include config/ifw/calob/schema/calob_cfg.schema.yaml:

# IFW Calob Tool Example Configuration
```

(continues on next page)

(continued from previous page)

```
database: !cfg.type:DatabaseDef
  db_path: "calob/db/calibration_history.db"

calob: !cfg.type:CalobDef
  calib_schedule_module: "ifw.calob.example.FORSUP_calibration_schedule"
  log_property_file: "config/ifw/calob/log/log_config.json"
  console_logger: "elt.log.test"

fits_crawler: !cfg.type:FitsCrawlerDef
  # Note: Normally data will be stored under the "DATAROOT" area and "data_path"
  # should be set to the relative path under "DATAROOT".
  data_path: "$INTROOT/resource/image/ifw/calob/example"
  file_pattern: "*.fits"
  timeframe_start: ""
  timeframe_end: ""

exporter_ob: !cfg.type:ExporterObDef
  path_dc_template: "resource/template/ifw/calob/daily_calib.obd.json"
  path_dc: "calob/daily_calibration/daily_calib.obd.json"
  obd_path: "resource/obd/ifw/calob/example"
  obdx_path: "resource/obdx/ifw/calob/example"
```

The parameters are as follows:

3.1.1 Category: “database”

db_path	Path to the name of the SQLite database file, in which the history of previous executions of the CalOb is recorded. It is given as a path, relative to “CFGPATH”.

3.1.2 Category: “calob”

General configuration parameters.

calib_schedule_module	Name of the Calibration Plan contained in a Python module. Since this is loaded by the Python interpreter it needs to be located in a path defined by the “PYTHONPATH” environment variable.
log_property_file	Log properties file for the CII Log Service (“log4py”).
console_logger	The CII Logger defined in the configuration

3.1.3 Category: “fits_crawler”

Parameters related to the internal FITS scanning service.

data_path	Location top directory where to scan for science data (FITS files). Given relative to the “DATAROOT”.
file_pattern	File pattern to apply when performing the FITS file scanning. Example: “FORSUP*.fits”.
timeframe_start	Start of timeframe to use for FITS file filtering. Example: “2023-07-01”
timeframe_end	End of timeframe to use for FITS file filtering. Example: “2023-08-01”

3.1.4 Category: “exporter_ob”

Parameters defining the properties for exporting (generating) the resulting Daily Calibration OB.

path_dc_template	Template JSON document, used for creating the Calibration Observations entries in the Daily Calibration OB.
path_dc	Name of output Daily Calibration OB document.
obd_path	Path for location where to look for Static Calibration OBs.
obdx_path	Path for location where to look for Template Calibration OBs.

3.2 Calibration Schedule

A Calibration Plan consists of the following elements:

- **External Functions**: External Functions are user defined functions to carry out any particular procedure required in the processing of the Template Calibrations. They return a value that can be assigned to a key defined in the calibration, which will be stored in the output Daily Calibration OB. They take the headers of the current FITS file being processed as input. Implentation wise they are methods in a class, derived from “ifw.calob.calibration_plan.InstrumentCalibrationExternalFunctions”.
- **Calibration Tables**: Class derived from the “ifw.calob.calibration_plan.InstrumentCalibrationTables” base class, which defines tables with key/value associations, mapping into an output value, assigned to a key in the target Daily Calibration OB.
- **Calibration Schedule**: Class derived from “ifw.calob.calibration_plan.InstrumentCalibrationSchedule”, which defines the keywords for the output Daily Calibration OB and how the values of these keywords shall be processed.

A realistic example of a Calibration Schedule, can be found here[5].

[5] https://gitlab.eso.org/ifw/ifw-calob/-/blob/main/common/src/ifw/calob/example/FORSUP_calibration_schedule.py

In the following, the main structure of the FORSUP Calibration Schedule is shown.

```
from ifw.calob.calibration_plan import InstrumentCalibrationExternalFunctions,␣
↪InstrumentCalibrationTables, InstrumentCalibrationSchedule

class FORSCalibrationExternalFunctions(InstrumentCalibrationExternalFunctions):
    def __init__(self):
        super().__init__()

    ...

class FORSCalibrationTables(InstrumentCalibrationTables):

    def __init__(self):
        super().__init__()

    ...

class FORSCalibrationSchedule(InstrumentCalibrationSchedule):

    def __init__(self,
                 cc = None):

        super().__init__(cc)
        super().set_calibration_library(FORSCalibrationLibrary())

       ...
```

An important element used in the tool is the “HeaderDict”, which is Python dictionary with some additional functionality. This is defined as follows:

```
class HeaderDict(dict):
  """
  @brief Class used to hold a FITS file header
  @details
  Subclass from dict which adds some additional handling
  when retrieving keys

  @ingroup common
  """
  def set_g(self, g):
      self.g = g

  def get(self, key):
      """
```

(continues on next page)

(continued from previous page)

```
    Function that gets a key from g first and in case
    of failure, retrieves the value from the header
    """
    if key in self.g:
        return(self.g[key])
    else:
        return(self.__getitem__(key))

def __getitem__(self,key):
    """
    @brief Override of __getitem__ that handles some variations
    @details
    in the key definition:
    1. Replaces . for spaces in the key name
    2. Tries to get the key in that form
    3. In case of failure it adds ESO to the start and tries
       to get the key again
    4. In case of failure it returns an empty string
    """
    try:
        key = key.replace("."," ").upper()
        val = dict.__getitem__(self, key)
    except KeyError:
        try:
            val = dict.__getitem__(self, f"ESO {key}")
        except KeyError:
            val = ""

    return val

def __contains__(self, key):
    """
    @brief Override of __contains__ that handles some variations
    @details
    in the key definition:
    1. Replaces . for spaces in the key name
    2. Tries to get the key in that form
    3. In case of failure it adds ESO to the start and tries
       to get the key again
    4. In case of failure it returns False
    """

    key = key.replace("."," ").upper()
```

(continues on next page)

(continued from previous page)

```
    if dict.__contains__(self, key) :
        return True
    elif dict.__contains__(self, f"ESO {key}") :
        return True
    else:
        return False
```

This dictionary is called “h” in the external functions and template calibrations. Used as a regular dictionary, for example:

```
time = h["INS.LAMP1.TIME"]
```

will retrieve the key “INS.LAMP1.TIME” from the header, but will try to standardise the input and also attempt to match a variation. The dictionary also add the function “get”, for example:

```
time = h.get("INS.LAMP1.TIME")
```

In this case the key will be searched on the Template Calibration keys that have been already produced, so if for example the Template Calibration had already defined the searched key by another means, then that would be the result. In case the keyword is not found, it will then fallback and look into the available headers.

3.3 Built-In Functions

The following Built-In Functions are provided to define rules for when to schedule Calibration Observations:

rule_min_days_since_last_exec	Sets the minimum days that have to pass before the CalibrationProp is executed again
rule_only_if_key_eq	Adds a rule where *key* in the FITS header has to match *value*
rule_only_if_key_diff	Adds a rule where *key* in the FITS header has to be different to *value*

3.4 External Functions

An example of a user defined (external) function is shown in the following:

```
def forsup_mos(self, h, clb):
    txt = ""
    mosmax = int(clb.tbl['USERVAR']['MOS_MAX_SLITS'])
    for i in range(1, mosmax + 1):
        key = f"INS.MOS{i}"
```

(continues on next page)

(continued from previous page)

```
        pos = float(h[key + ".POS"])
        wid = float(h[key + ".WIDTH"])
        txt += f"{key}.POS {pos:.3f} {key}.WIDTH {wid:.3f} "

    return txt
```

The “clb” parameter is a reference to the Calibration Plan object.

3.5 Calibration Tables

An example of a table definition in the Calibration Table class is shown in the following:

```
self.tbl['DET.MODE'] = {
        ("100Kps/2ports/high_gain", 2, 2) : "100kHz,2x2,high",
        ("100Kps/2ports/high_gain", 1, 1) : "100kHz,1x1,high",
        ("200Kps/2ports/low_gain",  2, 2) : "200kHz,2x2,low",
        ("200Kps/2ports/low_gain",  1, 1) : "200kHz,1x1,low",
        ("625Kps/2ports/low_gain",  2, 2) : "200kHz,2x2,low",
        ("HIT-MS",                  2, 2) : "HIT-MS",
        ("HIT-OS1-1sec",            1, 1) : "100kHz,2x2,high",
        ("HIT-OS2-4sec",            1, 1) : "100kHz,2x2,high",
        ("HIT-OS3-16sec",           1, 1) : "100kHz,2x2,high",
        ("HIT-OS4-64sec",           1, 1) : "100kHz,2x2,high",
        ("HIT-OS5-256sec",          1, 1) : "100kHz,2x2,high",
        ("HIT-OS6-1024sec",         1, 1) : "100kHz,2x2,high",
        ("HIT-OS1-1sec",            2, 2) : "100kHz,2x2,high",
        ("HIT-OS2-4sec",            2, 2) : "100kHz,2x2,high",
        ("HIT-OS3-16sec",           2, 2) : "100kHz,2x2,high",
        ("HIT-OS4-64sec",           2, 2) : "100kHz,2x2,high",
        ("HIT-OS5-256sec",          2, 2) : "100kHz,2x2,high",
        ("HIT-OS6-1024sec",         2, 2) : "100kHz,2x2,high"
}
```

The tables are used in the Template Calibration OBs to map keyword values or set of keyword values into a value to assign to another key.

An example of an assignment in a Template Calibration OB, involving the table above, can be found in FORSUP_img_cal_bias.obdx.json[6].

In this Template Calibration OB, the key “DET.READ.CLKIND”, is assigned a value from the “DET.MODE” table above, as shown in this snippet from the template:

[6] https://gitlab.eso.org/ifw/ifw-calob/-/blob/main/common/resource/obdx/ifw/calob/example/FORSUP_img_cal_bias.obdx.json

```
{
  "name": "DET.READ.CLKIND",
  "type": "string",
  "value": "<% clb.tbl['DET.MODE'][(h.get('DET.READ.CLOCK'), h.get('DET.WIN1.BINX'), h.
↪get('DET.WIN1.BINY'))] %>"
},
```

The values from the keys “DET.READ.CLOCK”, “DET.WIN1.BINX” and “DET.WIN1.BINY”, are used to index the “DET.MODE” table to return the value to assign to “DET.READ.CLKIND”.

E.g., “clb.tbl[‘DET.MODE’][(‘HIT-MS’, 2, 2)]” yields the value “HIT-MS”.

3.6 Calibration Schedule Class

The Calibration Schedule Class in the Calibration Plan python module, is used to define the sequence Static and Template Calibration OBs to be scheduled in the output Daily Calibration OB.

For each Calibration Observation to consider for scheduling, it is possible to specify various conditions for when to schedule it.

The following snippet from the example Configuration Schedule illustrates this:

```
def init_calibration_schedule(self):
  condition  = "(h['ESO DPR CATG'] == 'SCIENCE' or "
  condition += " h['ESO DPR CATG'] == 'CALIB' or "
  condition += " h['ESO DPR CATG'] == 'ACQUISITION') and "
  condition += " h['ESO DPR TYPE'] != 'BIAS'"
  super().add_filter_condition(condition)

  # Static calibrations
  ...
  super().add_static_calib("FORSUP-focus-blue.obd.json") \
      .rule_only_if_key_eq("CCD", "BLUE") \
      .rule_min_days_since_last_exec(7)

  # Template calibrations
  ...
  super().add_template_calib("FORSUP_mos_cal_daycalib.obdx.json") \
      .rule_min_days_since_last_exec(1) \
      .rule_only_if_key_eq("INS.MODE", "MOS") \
      .rule_only_if_key_diff("TPL.ID", "FORSUP_specphot_obs_exp_fast")
  ...
```

The first part defines an overall filter to pass to the crawler in this case, were only the FITS files for which the condition is evaluated to *True* are included in the subset for which to create calibrations.

Next, the Static Calibration OBs are defined.

In the above example the FORSUP-focus-BLUE.obd.json[7] Static Calibration OB, is scheduled if the FITS header key “CCD”, has the value “BLUE”. In addition it is specified to schedule this OB once per week (“every 7 days”).

For the Template Calibration OB example above, the FORSUP_mos_cal_daycalib.obdx.json[8] Template Calibration OB, is scheduled on a daily basis the keyword “INS.MODE” is found in a FITS header in the set of FITS files analysed with value “MOS” and if the file is generated by executing the “FORSUP_specphot_obs_exp_fast” template.

3.7 Static Calibration OBs

The Static Calibration OBs are ready-to-use ‘sub-OBs’ which can be inserted in the Daily Calibration OB ‘as-is’, if the conditions for scheduling these are fulfilled.

An example excerpt of a Static Calibration OB is shown in the following (FORSUP-lampcheck-new.obd.json[9]):


```
{
 "obId": 0,
 "itemType": "OB",
 "name": "FORSSUP_lampcheck_new",
 "executionTime": 0,
 "runId": "string",
 "instrument": "FORSUp",
 "ipVersion": "string",
 "obsDescription": {
  "name": "f2ocal-lampcheck-new",
  "userComments": "A",
  "instrumentComments": "AA"
 },
 "templates": [
  {
   "templateName": "FORSUP_img_cal_coll",
   "type": "string",
   "parameters": [
    {
     "name": "TPL.NAME",
     "type": "string",
     "value": "Collimator template"
    },
```


(continues on next page)

[7] https://gitlab.eso.org/ifw/ifw-calob/-/blob/main/common/resource/obd/ifw/calob/example/FORSUP-focus-BLUE.obd.json

[8] https://gitlab.eso.org/ifw/ifw-calob/-/blob/main/common/resource/obdx/ifw/calob/example/FORSUP_mos_cal_daycalib.obdx.json

[9] https://gitlab.eso.org/ifw/ifw-calob/-/blob/main/common/resource/obd/ifw/calob/example/FORSUP-lampcheck-new.obd.json

(continued from previous page)


```
      {
        "name": "TPL.MODE",
        "type": "string",
        "value": "IMG"
      },
      {
        "name": "TPL.SEQNO",
        "type": "integer",
        "value": 2
      },
      {
        "name": "INS.COLL.NAID",
        "type": "string",
        "value": "COLL_SR+6"
      }
    ]
  },
  ...
```


In the example above, the template “FORSUP_img_cal_coll”, is scheduled for execution in the Daily Calibration OB. It is being assigned the fixed values as indicated.

3.8 Template Calibration OBs

Template Calibration OBs are OBs, which contain keyword values that will be computed by CalOb, when processing the Calibration Plan.

An example of a Template Calibration OB is shown in the following (FORS2_img_cal_coll.obdx.json[10])


```
{
  "obId": 0,
  "itemType": "OB",
  "name": "FORS2_img_cal_coll",
  "executionTime": 0,
  "runId": "string",
  "instrument": "FORSUp",
  "ipVersion": "string",
  "obsDescription": {
    "name": "FORS2_img_cal_coll",
    "userComments": "A",
    "instrumentComments": "AA"
```


(continues on next page)

[10] https://gitlab.eso.org/ifw/ifw-calob/-/blob/main/common/resource/obdx/ifw/calob/example/FORSUP_img_cal_coll.obdx.json

(continued from previous page)

```
  },
  "templates": [
    {
      "templateName": "FORS2_img_cal_coll",
      "type": "string",
      "parameters": [
        {
          "name": "TPL.NAME",
          "type": "string",
          "value": "Collimator template"
        },
        {
          "name": "INS.COLL.NAID",
          "type": "string",
          "value": "<% h['INS.COLL.NAME'] + h['INS.COLL.ID'] %>"
        }
      ]
    }
  ]
}
```

In the example above, a template invocation of the “FORSUP_img_cal_coll”, with the parameters and values listed, is scheduled in the Daily Calibration OB.

Note that the values of the key “INS.COLL.NAID” is calculated with the expression “<% h[‘INS.COLL.NAME’] + h[‘INS.COLL.ID’] %>”, whereby this is executed (evaluated) by the Python interpreter. The value of the keys “INS.COLL.NAME” and “INS.COLL.ID” are addressed in the FITS header dictionary, for the FITS file currently being processed.

4 The CalOb SDK

In this chapter an overview of the various classes, provided by the CalOb Package is given. Not all details may be given, these can be found in the Doxygen documentation.

4.1 Classes

In this section some details of some classes in the CalOb package is given. The detailed information about these, can be found in the Doxygen documentation.

The following classes are provided by the CalOb Package:

Table 1: Classes

Module	Class	Description
adapters	*CalobInputPluginAdapter*	Class that defines the input adapters
adapters	*CalobOutputPluginAdapter*	Class that defines the output adapters
calibration	*CalibrationPlan*	Base class for the Instrument Calibration Schedule
calibration	*CalibrationProp*	Class that holds a calibration proposal, as defined in the Calibration Schedule
calibration	*Keyword*	Base class that defines a key
calibration_plan	*InstrumentCalibrationExternalFunctions*	Class that provides the base for the External Functions
calibration_plan	*InstrumentCalibrationLibrary*	Class that provides the base for the Calibration Library
calibration_plan	*InstrumentCalibrationSchedule*	Class that provides the base for the Calibration Schedule
calibration_plan	*InstrumentCalibrationTables*	Class that provides the base for the Calibration Tables
calob	*CalObApplication*	Class that contains the main part of the application
calob_config	*CalobConfig*	Class that handles the configuration for the tool
calob_db	*Calibration*	Class that models the calibration data in the DB
calob_db	*CalobDB*	Class that takes care of the interaction with the Calibration DB
calob_exporter_ob	*ExporterOB*	Class that exports the daily calibration
calob_generator	*CalobGenerator*	Class that calls the SDK and produces a usable application
calob_lib	*CalobLib*	Class containing general supporting functions
calob_lib	*HeaderDict*	Class used to hold a FITS file header
fits_crawler	*FitsCrawler*	Class that searches for FITS files

5 Installation & Deployment

In this chapter some information in connection with the installation and usage of CalOb are provided.

5.1 Installation

The CalOb Package is a standard waf/wtools build project and shall be configured, built and installed accordingly.

The CalOb Control is relying on the following environment variables:

- **DATAROOT:** Directory on the host machine in which Output Data Products will be generated. The storage location will be rendered as "$DATAROOT/<cfg key: server.recording.image_dir>". The 'image' sub-directory in "DATAROOT" will be created automatically by CalOb Control, if not existing.
- **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 configurations delivered with the CalOb Package as well as the libraries, header files and binaries, are installed into the location pointed to by "INTROOT". This may change in the future.

Note, in accordance with the "Instrument Software Specification", one system, typically an instrument, shall deliver a ready-to-use Resource Tree as part of the software package. This Resource Tree shall be referenced to by the "CFGPATH" variable.

5.2 Deployment Module

Details about the configuration of a CalOb Control instance are given in the "Configuration" chapter in this manual (*Configuration*).

The CalOb Package provides an SDK that can be used for integrating 'CalOb solutions'. The default tool, "calobGenerator" is an example of a deployment module.

The example from "ifw-calob/generator" is shown here:

```
#!/usr/bin/env python3

  import argparse
  import importlib
  import inspect

  from elt.log import CiiLogManager

  from ifw.calob.calob_config import *
```

(continues on next page)

(continued from previous page)

```
class CalobGenerator:
    """
    @brief Class that calls the SDK and produces a usable application

    @ingroup common
    """

    def __init__(self):
        self.args = self.parse_arguments()
        self.init_parameters()

    def parse_arguments(self):
        arg_parser = argparse.ArgumentParser()
        required_args = arg_parser.add_argument_group('required arguments')
        required_args.add_argument("-c", "--cfg", help="path to the yaml configuration file",
↪required=True)

        arg_parser.add_argument("-l", "--log-level", help="set log level")
        arg_parser.add_argument("-p", "--log-property-file", help="set the log property file")
        arg_parser.add_argument("-n", "--no-history", help="do not consider history stored in the
↪DB",
                                action="store_true")
        arg_parser.add_argument("-x", "--console", help="output to console", action="store_true")

        return(arg_parser.parse_args())

    def init_parameters(self):

        if self.args.cfg:
            self.cfg = self.args.cfg

        self.cc = CalobConfig(config_path=self.cfg)

        if self.args.log_property_file:
            self.cc.log_property_file = self.args.log_property_file

        if self.args.console:
            self.cc.console_logging = True

        self.cc.load_config()
```

(continues on next page)

(continued from previous page)

```
        if self.args.log_level:
            self.cc.logger.setLevel(self.args.log_level)

        if self.args.no_history:
            self.cc.no_history = True

    def generate(self):
        """
        @brief Method that works as starting point
        @details
        1. Loads the module defined as Instrument Schedule
        2. Initializes all data
        3. Calls generate_calibrations to produce the daily calibrations
        """

        self.cc.logger.info("Searching for calibration module")
        c = importlib.import_module(self.cc.get("calob:calib_schedule_module"))

        # Find the class which is a subclass of InstrumentCalibrationSchedule
        for k, v in inspect.getmembers(c, predicate=inspect.isclass):
            if issubclass(v, c.InstrumentCalibrationSchedule) and k !=
↪"InstrumentCalibrationSchedule":
                cp = v(self.cc)

        self.cc.logger.info("Initializing Calibration Schedule.")
        if cp:
            cp.init_calibration_schedule()
        else:
            self.cc.logger.error("No Calibration Schedule found")
            raise Exception("No Calibration Schedule found")

        self.cc.logger.info("Generating calibrations")
        cp.generate_calibrations()
```

5.3 CalOb Generator - Execution - Example

In the following an example is shown, which can be executed directly from the shell after the ICS software package have been installed. To do this, carry out the following steps:

1. Ensure “DATAROOT”, “CFGPATH” and “INTROOT” are defined. Example:

```
$ echo $DATAROOT
/scratch/eltdev/DATAROOT
```

(continues on next page)

(continued from previous page)

```
$ ll /scratch/eltdev/DATAROOT
drwxr-xr-x 2 eltdev vlt 80 May 11 14:48 image/
$ echo $CFGPATH
/scratch/eltdev/INTROOT/resource:/elt/ifw/resource
$ echo $INTROOT
/scratch/eltdev/INTROOT
```

2. Ensure that the CII Services are up and running: These will normally be started automatically on the ELT development machines.

3. Execute the example CalOb Generator executable "calobGenerator":

```
$ calobGenerator -c config/ifw/calob/example/configuration.yaml
$
```

To execute without the historic feature and showing output in the console:

```
$ calobGenerator -c config/ifw/calob/example/configuration.yaml -n -x
$
```

4. Show/explain generated output files: After executation, the output Daily Calibration OB generated can be found in "DATAROOT/calob/daily_calibration/daily_calib.obd.json", with the example CalOb Configuration File provided.

The Calibration DB file can be found in "$DATAROOT/calob/db/calibration_history.db". Note, the user shall normally not tamper with this file.

Logs generated during execution are stored in the log file "./ifw-calob.log", relative to the execution path. Normally the user shall not access this file directly, but shall use the CII Log Service GUI to browse the logs.

6 Initiating a Calibration Config Set for an Instrument

A CalOb Configuration Set consists of the documents listed in (*Configuration*).

The template instrument now provides a Cookiecutter template of its calibration schedule. Here are the steps to use the IFW Calob Template configuration.

6.1 IFW Software Configuration

If not yet done, retrieve and install the complete ICS Framework from the ESO RPMs repository. For more details, please have a look to the installation procedure here[11].

Please create your software based on the provided template by following the procedure in the Getting Started guide here[12].

6.2 IFW Calob Template Configuration

Assuming that the instrument name used for the template was *micado*, the files pertaining to the CalOb Configuration are presented in the following table:

File	Description
micado/mic-resource/config/calob/configuration.cfg.yaml	Holds the general configuration and paths
micado/mic-resource/obd/calob	Holds two OBD files to be used for the template calibrations
micado/mic-resource/obdx/calob	Holds one OBDX file to be used for the template calibrations
micado/mic-ics/calob/src/mic/calob/calob_schedule.py	The template schedule which will produce the daily calibrations
micado/mic-ics/calob/src/mic/calob/calob_tables.py	The tables to be used inside the schedule or obdx files
micado/mic-ics/calob/src/mic/calob/calob_ext_functions.py	The tables to be used inside the schedule or obdx files

These files should be modified according to the instrument needs following the instructions given in the previous chapters of this document.

[11] https://www.eso.org/projects/elt/develop/ifw/ifw-doc/manuals/ifw/src/docs/installation.html
[12] https://www.eso.org/projects/elt/develop/ifw/ifw-doc/manuals/ifw/src/docs/guide.html

6.3 IFW Calob Template Data

At the moment of writing this document, the best way of providing the CalOb Configuration template is to use simulated/pre-generated data for simplicity. Find these FITS files in:

micado/mic-ics/calob/resource/tmp_image/mic/calob

The configuration provided with the IFW Calob Template will use these files and produce a simulated Daily Calibration OB.

6.4 Executing the Generator

1. **Ensure "DATAROOT", "CFGPATH" and "INTROOT" are defined.**

   Example:

```
$ echo $DATAROOT
/scratch/eltdev/DATAROOT
$ ll /scratch/eltdev/DATAROOT
drwxr-xr-x 2 eltdev vlt 80 May 11 14:48 image/
$ echo $CFGPATH
/scratch/eltdev/INTROOT/resource:/elt/ifw/resource
$ echo $INTROOT
/scratch/eltdev/INTROOT
```

2. **Ensure that the CII Services are up and running:**

   These will normally be started automatically on the ELT development machines.

3. **Install the template instrument.**

```
$ waf configure
$ waf build install
```

4. **Execute the example CalOb Generator executable "calobGenerator":**

```
$ calobGenerator -c config/ifw/calob/example/configuration.yaml
$
```

   To execute without the historic feature and showing output in the console:

```
$ calobGenerator -c config/ifw/calob/example/configuration.yaml -n -x
$
```

5. **The following output files have been generated:**

   After executation, the output Daily Calibration OB generated can be found in:

   DATAROOT/calob/mic/daily_calibration/daily_calib.obd.json

with the template CalOb Configuration File provided.

The Calibration DB file can be found in:

$DATAROOT/calob/mic/db/calibration_history.db

Note, the user shall normally not tamper with this file.

Logs generated during execution are stored in the log file:

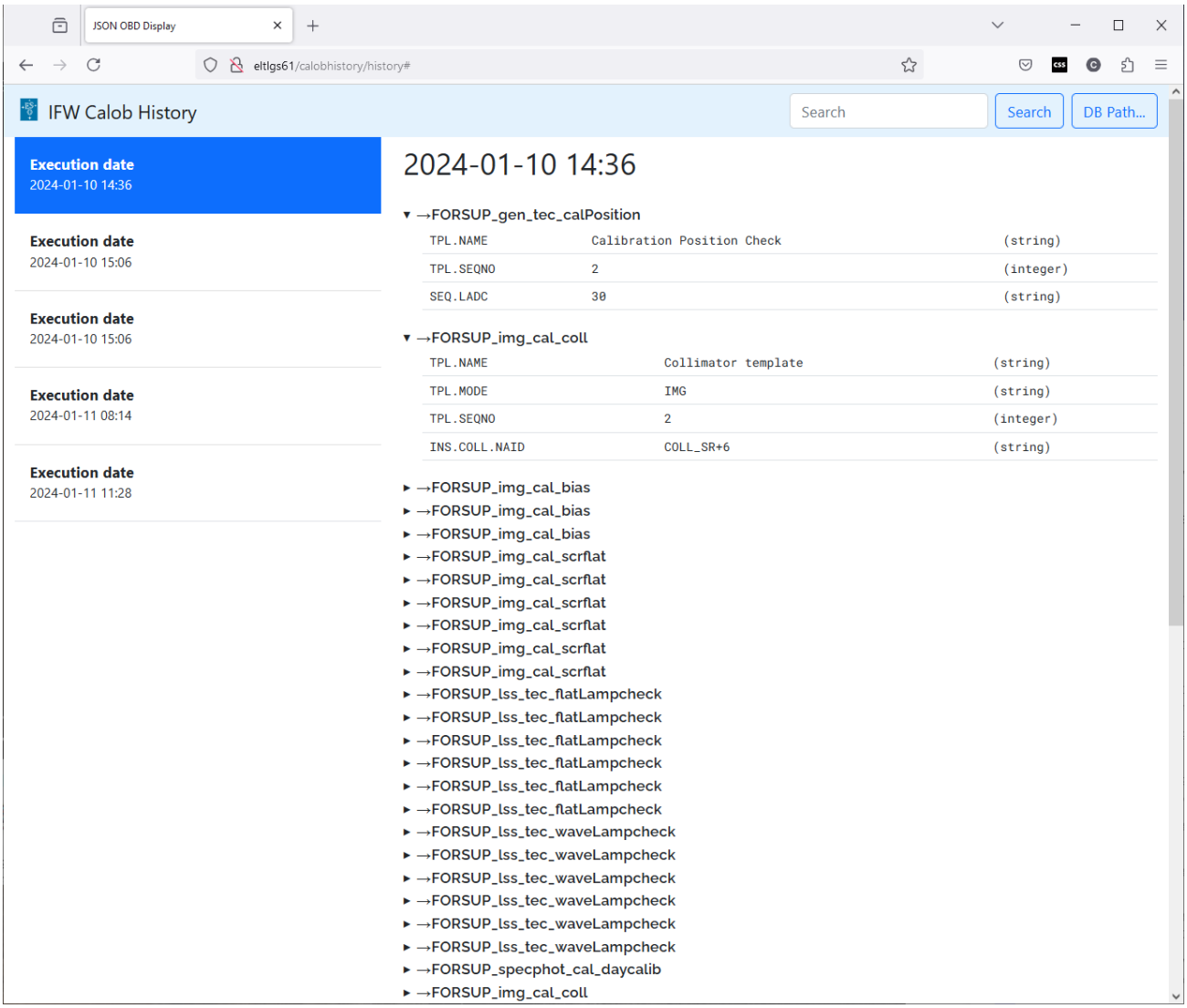
./ifw-calob.log

Normally the user shall not access this file directly, but shall use the CII Log Service GUI to browse the logs.

7 Calibration History WebUI

7.1 Overview

The IFW Calob WebUI allows the user to review the history of generated calibrations. The interface uses the calibration history database as its information source and displays the JSON OBD generated as a searchable web page.



7.2 Installation

7.2.1 Dependencies

The Web application depends on the rpm modules *nginx*, *flask* and *gunicorn* which should be installed in the workstation.

7.2.2 Deployment

NGINX

The configuration for nginx will be created automatically during installation named *calobhistory.conf*. This file will be installed in *<prefix>/resource/webapp/ifw/calob/deployment/calobhistory.conf* the prefix will depend on if it was installed as an RPM or through *waf install*.

A symbolic link should be created by *root* to this file to be used by the NGINX service. *# ln -s <path_to_calobhistory.conf> /etc/nginx/default.d/calobhistory.conf*

GUNICORN Service

The Flask application which implements the WebUI, should be running as a service. For this the file *calobhistory.service* will be generated and installed in *<prefix>/resource/webapp/ifw/calob/deployment/calobhistory.service*. Again prefix depends on the installation method.

A symbolic link should be created by *root* to this file to be available. *# ln -s <path_to_calobhistory.service> /etc/systemd/system/calobhistory.service*

Running

1. Recognize the changes on available services:

   # systemctl daemon-reload

2. Start the service:

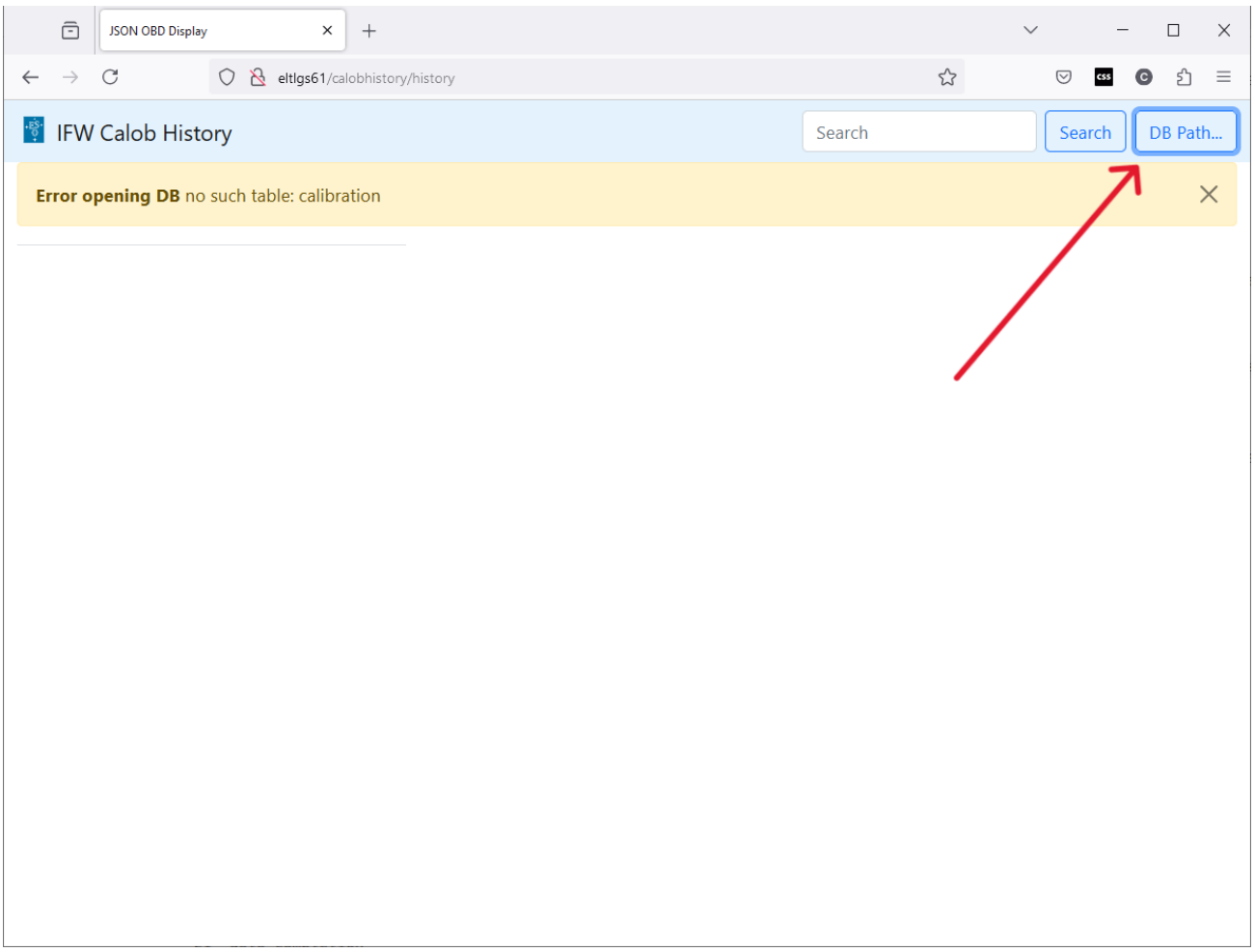
   # systemctl restart calobhistory

3. Restart NGINX:

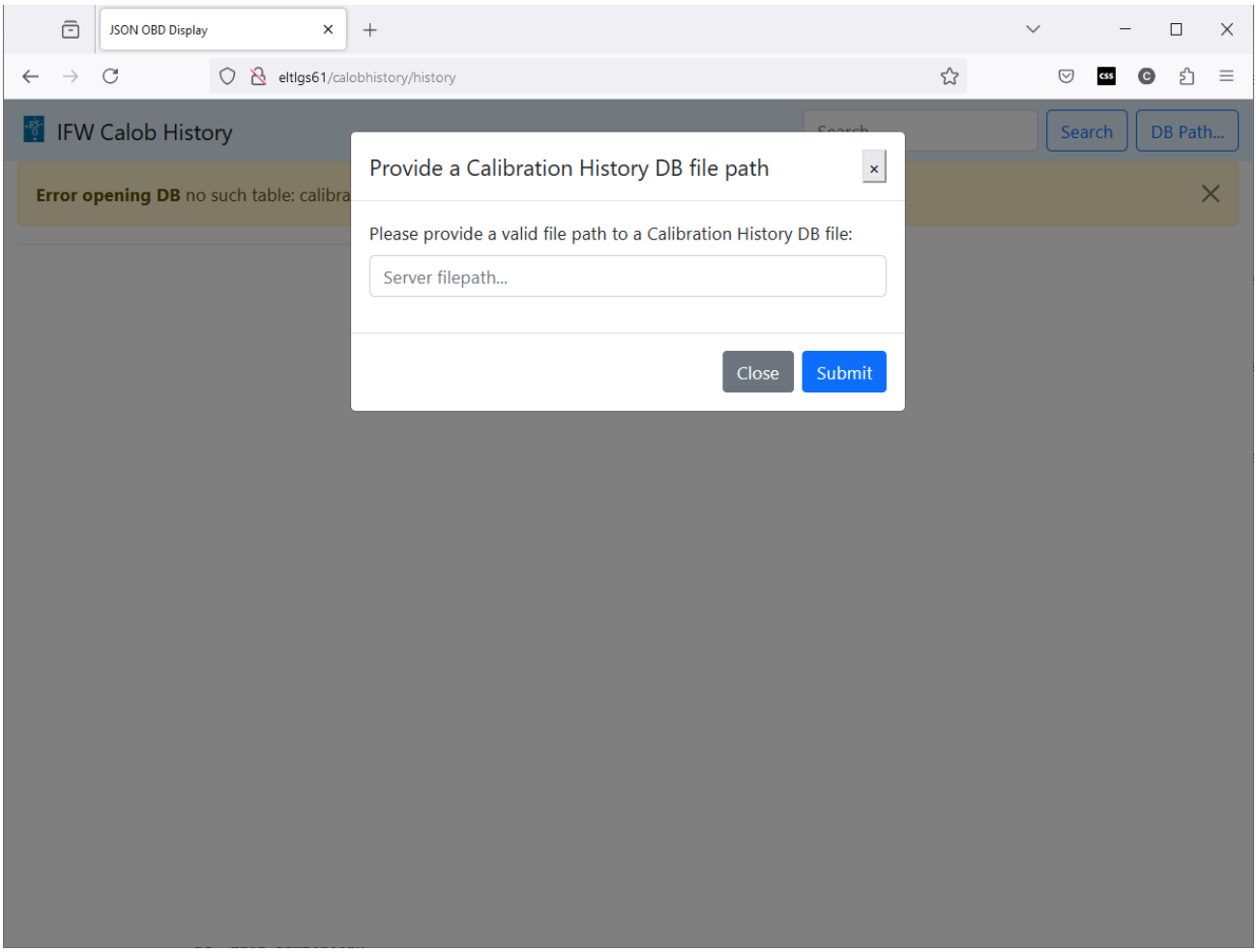
   # systemctl restart nginx

7.3 Configuration

At this point in time there is not much to configure on the interface. It needs the location of the calibration history database file, which can be provided in the interface itself here:



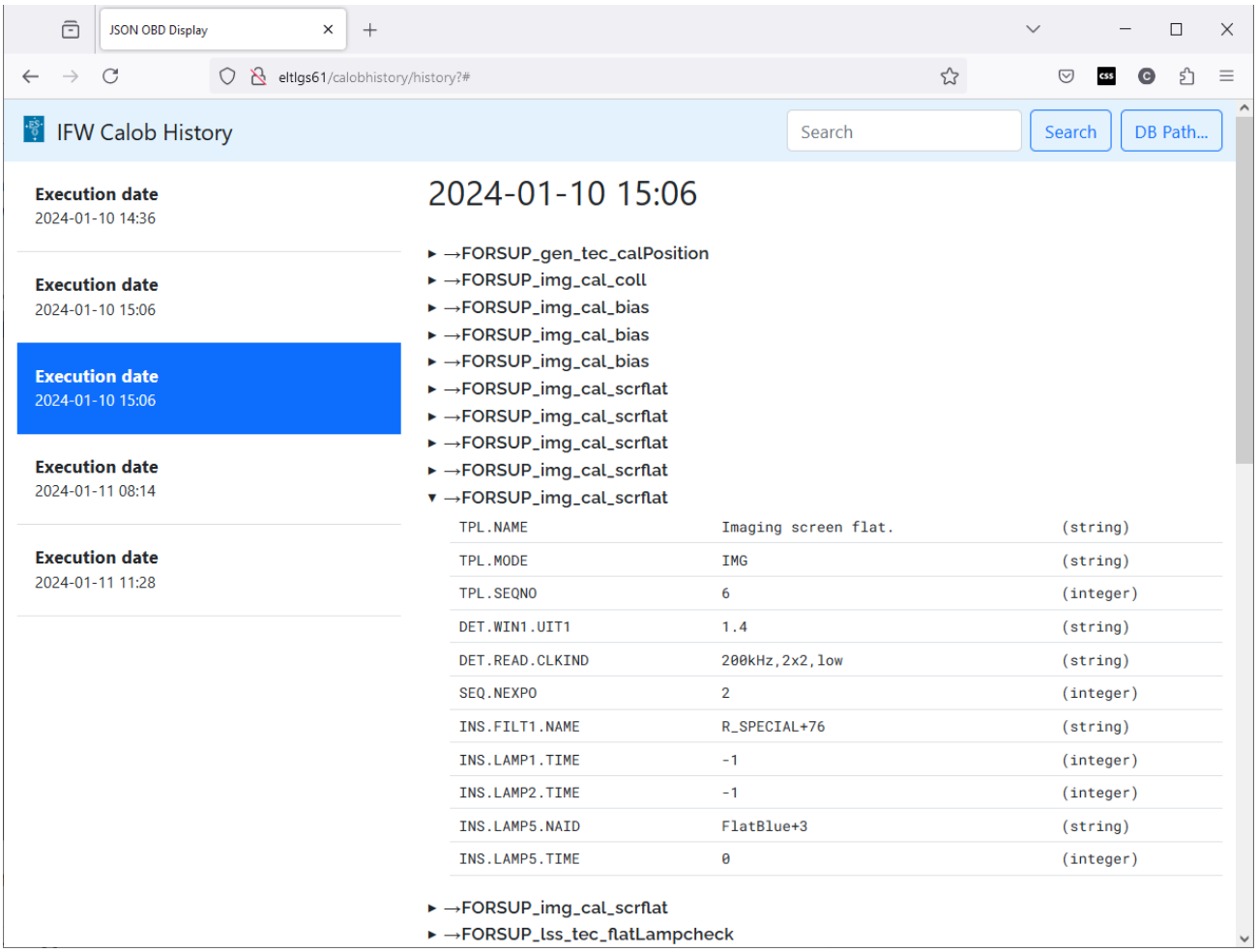
In the pop message the path to the file in the server should be provided, for example */home_local/eltdev/DATAROOT/calob/db/calibration_history.db*.



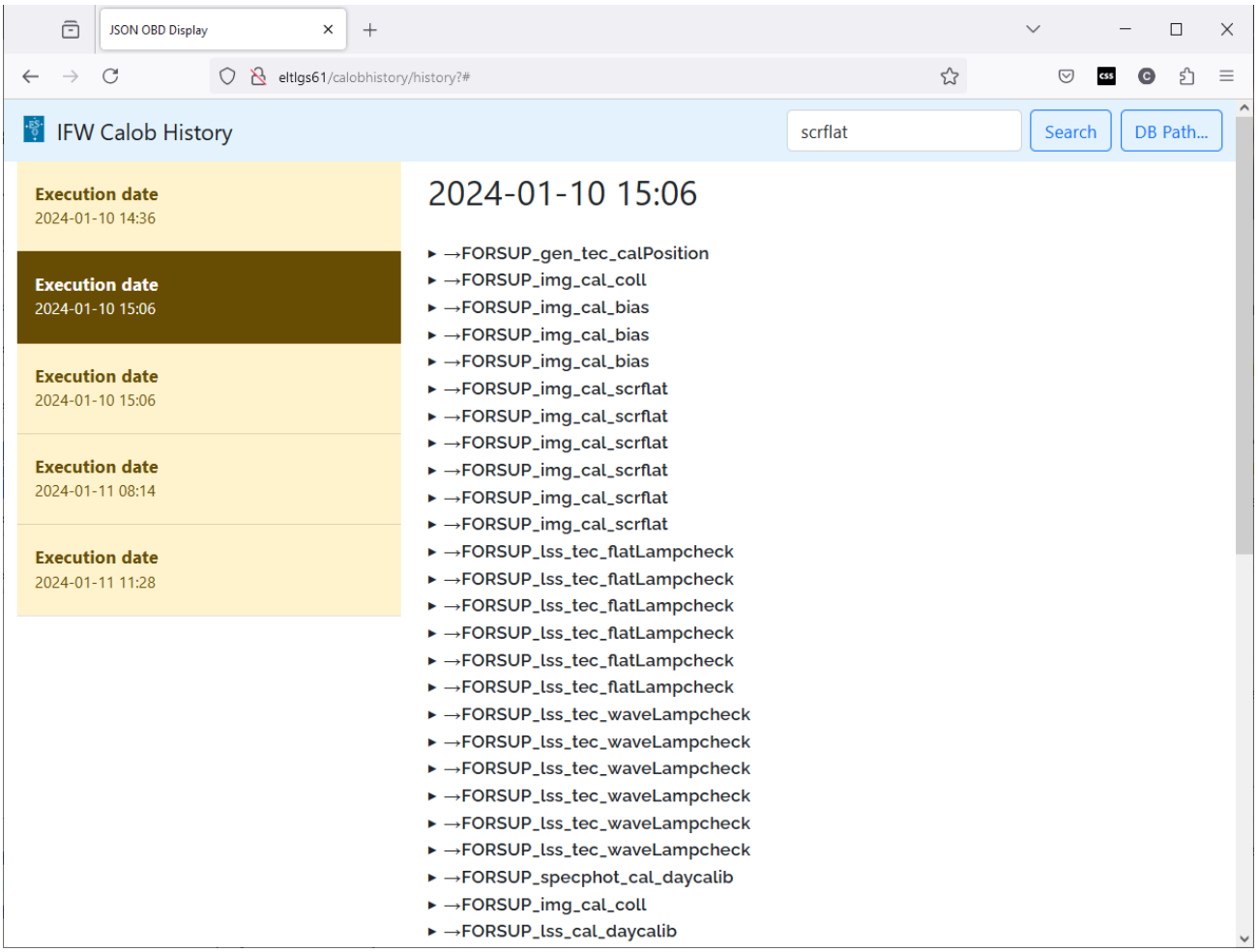
After pressing submit the interface should be populated with the contents from the database.

7.4 Usage

The UI works as any standard Web page, with any standard modern web browser.



- The history of calibrations found on the database will be displayed on the left. Each of the elements can be clicked to change the calibration been displayed.
- The selected calibration is displayed on the right side as a list of OBDs. Each OBD can be clicked to fold/unfold and display a table with its particular defined values.
- The search functionality searches inside the generated calibrations and marks with yellow the execution that contains the searched value. The value can be the name of an OBD, or a keyword, or the value of a keyword.


JSON OBD Display
eltlgs61/calobhistory/history?#
IFW Calob History
scrflat
Search
DB Path...
Execution date
2024-01-10 14:36
Execution date
2024-01-10 15:06
Execution date
2024-01-10 15:06
Execution date
2024-01-11 08:14
Execution date
2024-01-11 11:28
2024-01-10 15:06
→FORSUP_gen_tec_calPosition
→FORSUP_img_cal_coll
→FORSUP_img_cal_bias
→FORSUP_img_cal_bias
→FORSUP_img_cal_bias
→FORSUP_img_cal_scrflat
→FORSUP_img_cal_scrflat
→FORSUP_img_cal_scrflat
→FORSUP_img_cal_scrflat
→FORSUP_img_cal_scrflat
→FORSUP_img_cal_scrflat
→FORSUP_lss_tec_flatLampcheck
→FORSUP_lss_tec_flatLampcheck
→FORSUP_lss_tec_flatLampcheck
→FORSUP_lss_tec_flatLampcheck
→FORSUP_lss_tec_flatLampcheck
→FORSUP_lss_tec_flatLampcheck
→FORSUP_lss_tec_waveLampcheck
→FORSUP_lss_tec_waveLampcheck
→FORSUP_lss_tec_waveLampcheck
→FORSUP_lss_tec_waveLampcheck
→FORSUP_lss_tec_waveLampcheck
→FORSUP_lss_tec_waveLampcheck
→FORSUP_specphot_cal_daycalib
→FORSUP_img_cal_coll
→FORSUP_lss_cal_daycalib

8 Example

In this chapter, an overview of the example of a real (semi-real), Calibration Configuration for an instrument, here FORSUP, is provided.

The Calibration Configuration Set example is used internally in the CalOb Project for the purpose of testing, and to provide a 'live example' that makes it possible to execute the CalOb Generator on the shell with no configuration.

The example is provided as an 'integral part' of the CalOb source tree, to be able to use it for the software unit and integration testing, executed for the purpose of SQA.

Note: For instruments it is foreseen to have a ready-to-use Resource Tree inside the Instrument Source Tree, which is referenced by the "CFGPATH" environment variable.

8.1 Example Files in the CalOb Source Tree

The example provides the following files:

ifw-calob/common/resource/config/ifw/calob/example/configuration.yaml: The CalOb Configuration file, used when invoking the "calobGenerator" tool on the command line.

ifw-calob/common/resource/image/ifw/calob/example/FORS*_no_data.fits: Entire FITS data set, generated for one representative night with the FORS2 (later FORSUP) instrument, making it possible to obtain a realistic invocation of the "calobGenerator" tool on the command line.

ifw-calob/common/resource/obd/ifw/calob/example/FORSUP_*.obd.json: The actual set of Static Calibration OB file (OBD) for the FORS2 (later FORSUP) instrument.

ifw-calob/common/resource/obdx/ifw/calob/example/FORSUP_*.obdx.json: The actual set of Template Calibration OB file (OBD) for the FORS2 (later FORSUP) instrument.

ifw-calob./common/src/ifw/calob/example/FORSUP_calibration_plan.py: The Calibration Plan Module, executed in the Python interpreter by the "calobGenerator" tool to generate the output Daily Calibration OB. It provides examples of the Instrument External Functions Library Class, the Instrument Calibration Tables Class and the Calibration Schedule Class.

8.2 Example Files in the "INTROOT"

After installation, the files of the example are replicated into the "INTROOT" to make them easier accessible and to be able to use them on the command line, e.g. for demo invocation of the "calobGenerator" tool (listed in the same order as above, without additional comments):

- $INTROOT/resource/config/ifw/calob/example/configuration.yaml.
- $INTROOT/resource/image/ifw/calob/example/FORS*_no_data.fits.
- $INTROOT/resource/obd/ifw/calob/example/FORSUP*.obd.json.
- $INTROOT/resource/obdx/ifw/calob/example/FORSUP*.obdx.json.

- $INTROOT/lib/python<version>/site-packages/ifw/calob/example/FORSUP_calibration_schedule.py.

If the “$INTROOT/resource” path is included in the paths of the “CFGPATH” environment variable, the files will be automatically searchable by the standard ELT environment search mechanism.

Again, instruments shall not install the files, pertaining the Instrument CalOb Configuration Set in the “INTROOT” but shall provide a ready-to-use Resource Tree inside the Instrument Source Tree to keep the run-time files under configuration control. The same goes for the science FITS files, which shall be stored in the “$DATAROOT” area.