MEC Python and Terminal commmands

by the MEC team

January 28, 2024

Contents

1 General

1.1 Python

This document describes references of the commands used in mecpython. It is a document that is constantly in flux, since mecpython is too. If you find any errors in here, send an email to the MEC staff. To start MEC Python, open a terminal on any control computer in the mec control room, or any of the laptops in the hutch and type:

```
> mecpython
```

Currently, the various functions used to control the beamline or take laser shots during a standard configuration experiment are located here:

```
> cd /reg/g/pcds/pyps/apps/hutch-python/mec/mec/macros
> vi operation.py
```

This path has been added to the file `mecenv` in the folder `/reg/g/pcds/pyps/apps/hutch-python/mec/` so that it is automatically loaded when a python instance is launched in a terminal (no need to be in a specific folder). For now, we have not the module being automatically loaded when running the `mecpython`. We are still testing its use, but in the future, we might have it being part of the mecpython. To access the functions contained in this module, type the following when you are in a python invite:

```
In [1]: import operation as op   # preferred way to load the module to increase our ability to
    use it
```

This way to load the module allows to post automatically message to the elog. It has the small disadvantage to require the use of `op.` infront of all the functions included in it (lasers, beamline, scripts...). It might be improved in the future.

Usually, Python commands in the MECpython interactive prompt are written in the format:

```
In [1]: command(parameter1, parameter2=8.99)
```

To have a detailed explanation of what command does, write:

```
In [1]: command_name??
```

1.2 Command line scripts

In addition to python commands, here is a list of the most commonly used scripts to control and monitor the MEC beamline. They are all accessible from any temrinal launched from the MEC control room computers or hutch laptop.

The best and only way to start a camviewer (WATCH THE CAPITAL V, without it you get another old version!)

```
> camViewer
```

add -h to get the help info.

To start the GUI to control the MEC beamline:

```
> mechome
```

To start a striptool that provides the pulse energy at the GMD and w8

```
> gas_detector_striptool -f\\
> w8_detector_striptool\\
> w8only_striptool
```

To start a striptool that provides the gage readout for the target chamber vacuum:

```
> tc_vacuum
```

To start and restart the DAQ:

```
> restartdaq
```

To stop the DAQ:

```
> stopdaq
```

To show where the DAQ is currently running (usefull to make sure mec-daq is not being used for example):

```
> wheredaq
```

To start the offline DAQ (you can create an alias to this script to make it easier):

```
> /reg/g/pcds/engineering_tools/mec/scripts/startami
```

To listen to a PV name PVNAME:

```
> caget PVNAME
```

To change the value of the PVNAME:

```
> caput PVNAME value
```

To turn off the Unix computers, you need to open a terminal and type the following command:

```
> systemctl poweroff -i
```

1.3 For Users

Zoom link for the 8 am ACR meeting that the users have to attend:

```
https://stanford.zoom.us/j/742958069?pwd=NGFTTU1pSjZIU3RaMFg3eVhkVVpVZz09
Meeting ID: 742958069
Passcode  : 002151
```

To start a zoom meeting on the streaming computer, first use the following command in a temrinal (do not try to double click on the Zoom icon):

```
flatpak run us.zoom.Zoom
```

To start the viewer which stream the bottom screen of the mec-daq computer, type the following in a terminal:

```
showcam -r uhd show 2
```

The `uhd` argument stands for ultra high definition, used to stream the highest quality resolution of the DAQ monitor. `show 2` is the indicate which screen to share. Number 2 is the bottom screen of the DAQ computer, while 1 is the top. Finally, share this stream on zoom to allow everyone see the DAQ through the zoom window.

1.4 FEL status

Here are few web links to display the status of the FEL (photon energy, electron energy, GDET, repetition rate, ...):

```
https://pswww.slac.stanford.edu/ctl/grafana/d/NsZbmqOGz/l-line-lcls-stats?orgId=1&refresh=5s
```

For the ACR on-shift operator:

```
http://mccelog.slac.stanford.edu/elog/wbin/display.html
```

For the mirror coating positions, the link is the following (needs Unix login):

```
https://pswww.slac.stanford.edu/ctl/grafana/d/nNrOH8LMk/lfe-optics?orgId=1&refresh=5s
```

At the moment, the mec-control monitor on the south west wall has these link displayed so that everybody in the control room can see the status.

2 VNC

For these to work, ’operation’ needs to be imported as ’op’.

VNC for visar computers:

```
In [1]: op.visar1_remote()
In [2]: op.visar2_remote()
```

VNC for timing scope:

```
In [1]: op.scope_timing_remote()
```

3 Detectors

3.1 ePix geometry files

To create a folder with the right geometry file for the current experiment, run this command in a terminal:

```
makepeds -r <run_with_quads> -g <experiment_with_correct_geometry_for_relevant_detectors>
```

If you are on psana, you can use:

```
makepeds_psana -e <current_experiment> -r <run_with_quads> -g <
    experiment_with_correct_geometry_for_relevant_detectors>
```

3.2 ePix pedestals

Background subtraction of the ePix10k detectors is done using two different commands to execute in a terminal. The first step is to set the event sequencer to 120 Hz otherwise it would take too long:

```
In [1]: op.x.start_seq(120)
```

Then, you run a command which will save a run with the appropriate number of runs and the appropriate change of gains of the detectors. The DAQ needs to be in `beam mode` and `allocated`. You do not need to check to record run since the command will do it automatically. Then, the command simply runs as follow:

```
> takepeds
```

Once the run is finished (usually in 5 min at 120 Hz), use the saved run number (here 136) into the following command (here username *egaltier* to be replaced by the operator and the run number *136* to be replaced by the appropriate one):

```
> makepeds -u egaltier -r 136 -q milano
```

Because it needs access to the data, you need to enter a username and the correponding password. Let the script finish, which should take 10 min as well. To make sure all worked properly, `shutdown` the DAQ (you do not need to restart the DAQ, you just need to place it in the `shutdown` state), and `allocate` right after. After clicking `correct gain` on each of the quad AMI of interest, the noise should be within few ADU.

4 Laser

4.1 Python initialization

The laser commands are loaded automatically when importing the module `operation`. For now, it is loaded in the file using with the following command:

```
In [1]: import meclas
```

Because this module is imported in the `operation` module itself, the instance with which you named the import (usually `op` or `opr`) needs to be used to access each functions of `meclas`.

4.2 Laser operation

Some conventions:

- Arms ABEF are on the West of the target chamber TCC (away from FEL beam dump)
- Arms GHIJ are on the East of the target chamber TCC (towards FEL beam dump)

To turn on and off the front end of the nslaser:

```
In [1]: op.YFEon()
In [2]: op.YFEoff()
```

To set the transmission of all the waveplates to 0.6 of maximum (to keep the pulseshapes, you should always shoot 4 arms, with all waveplates on the same transmission):

```
In [1]: op.HWPon('all', set_T=0.6)
```

To save Lecroy1 to a file, and pushe channel 2 to the elog:

```
In [1]: op.save_scope_to_eLog()
```

To optimizes the doubler efficiency. They are usually between 55 and 60%. If they drop lower, you can run this script. The LSS transport shutter should be closed, but the other shutter (S4) should be open, and the laser should be triggered at 10 Hz from the nslaser opetator screep. You may need to run it twice initially, if the values were far of from the maximum:

```
In [1]: op.SHG_opt()
```

To check the status of the internal shutters of the laser (the one at the input of the big heads and in between the lenses of the final beam expander):

```
In [1]: op.TTL_shutter_status()
```

The reset command `op.TTL_shutter_status()` allows you to reset the status fo the shutters to all open, and could be used to fix some light on target issues. But it needs physical check in the hutch of the shutter status and possible manual reset before executing the command. To use the TTL shutters and close individually the laser arms, use the following commands:

```
In [1]: op.TTL_shutter.Toggle("openABEF")
In [2]: op.TTL_shutter.Toggle("closeABEF")
In [3]: op.TTL_shutter.Toggle("openABEFGHIJ")
```

To run before shooting the main beam (not sure about details of what it does):

```
In [1]: op.LPL.pspreshot()
```

To run this after the shot. Saves the pulseshapes to daq folder, and does some optimization (not sure about details):

```
In [1]: op.LPL.pspostshot()
```

4.3 LPL pulse shapes

Pulse shapes are in the following directory:

```
In [1]: cd ~mecopr/mecpython/pulseshaping/recipes
In [2]: ls
```

To check what pulse shapes are available:

```
In [1]: op.LPL.psmenu()
```

To view the shape of a recipe:

```
In [1]: op.LPL.psviewwvfm('08ns00grad')
```

This will print the requested pulse shape (without the ';', you'll get a annoying print to terminal). To load the pulse shape:

```
In [1]: op.LPL.psloadwvfm('08ns00grad')
```

To optimize the pulse shape in case its shapes slowly drift (this can be done at any time):

```
In [1]: op.LPL.psrefrwvfm('15ns00grad')
```

It executes the following commands:

- reload the recipe for a 15ns flat-top (0% gradient) pulse
- close all the shutters
- turn off the bias dither if it isn't already (note: dither doesn't re-engage; specifically done this way to incorporate into pspreshot(). . .)
- turn on 10Hz operation (with the shutters preventing any light from entering the target chamber)
- adjust the pulse shaper to re-converge to the desired front-end optical waveform
- return to Event Code for single-shot operation
- re-open all shutters (to prepare for shots again)

In addition, you can display the resulting pulse shapes by using other options:

```
In [1]: op.LPL.psrefrwvfm('15ns00grad',displayPlotQ=True,stepSizeQ=0.0,numStepsQ=1)
```

while you can increase the number of steps to allow for a better convergence using this command:

```
In [1]: op.LPL.psrefrwvfm('15ns00grad',displayPlotQ=True,numStepsQ=150)
```

For timing, use the pulse shape 'YFEedge'. You may need to refresh twice to get a nice clean edge.

4.4 Timing

To start the scope that is used for the timing (which has the main trigger and the diode at TCC), use this command:

```
In [1]: op.scope_timing_remote()
```

It will open the VNC to the scope which does not require any password. Currently, the EVR used for trigger the scope is not great so the change in event code on a channel seems not to be followed by the right offset in time, so we have to set them up manually. Using the following command does it:

```
In [1]: op.lpl_check_timing("10Hz")
In [1]: op.lpl_check_timing("single")
```

where the argument `10Hz` means triggering the scope at 10 Hz and the argument `single` set the scope to allow for single shot acquisition (for example, setting the channel to 1V/div would allow you to see the plasma emission on shot and ocnfirm coarse timing on shot).

The timing changes uses the same syntax as a motor. Positive values of the timing, mean the X-ray are after the optical. Uses a bunch of PV values in the second column of the user pv (CHA T0, CHC T0, etc) To move the delay of the FEL relative to the Optical laser (here, changes the timing 1ns later than the current):

```
In [1]: op.nstiming.mvr(1e-9)
```

To move the delay of the FEL in absolute vs the Optical laser (here, changes the timing to 6ns absolute, vs t0):

```
In [1]: op.nstiming.mv(6e-9)
```

To display the current delay (full sentences are displayed, using previous example, of 1 ns relative delay vs t0):

```
In [1]: op.nstiming.get_delay()
X-rays arrive 1.0000000000157452e-09 s after the optical laser
X-rays arrive 1.0000000000021927e-09 s after the optical laser
X-rays arrive 1.0000000000021927e-09 s after the optical laser
X-rays arrive 1.0000000000021927e-09 s after the optical laser
```

To Saves the current time as the new t0 (the old one is backed up in the backup user PVs):

```
In [1]: op.nstiming.save_t0()
```

To restore the timing to values saved in the backup user PVs, on all the DG channels that are relevant to the change in timing. It means that it recalls the original timing state even after you changed t0:

```
In [1]: op.nstiming.restore_t0()
```

5 Beamline

5.1 Python initialization

Access to the functions in the `operation` module is done by importing the module direclty in mecpython:

```
In [1]: import operation as op
```

To show a barebone version of our running status; lots is missing here and no pretty colors:

```
In [1]: op.rs()
```

5.2 Components

To insert and remove the yag screens in the MEC beamline:

```
In [1]: op.yag0.insert()
In [2]: op.yag0.remove()
In [3]: op.yag1.insert()
In [4]: op.yag1.remove()
In [5]: op.yag2.insert()
In [6]: op.yag2.remove()
In [7]: op.yag3.insert()
In [8]: op.yag3.remove()
```

To display a status of the transmission produced by the Si attenuators:

```
In [1]: op.SiT()
```

To set the transmission of our Si attenuators to 0.5 (1 corresponds to 100% transmission, 0 corresponds to 0%):

```
In [1]: op.SiT(0.5)
```

To set the current photon energy used to calculate the SiT transmission (might be necessary to execute if transmission value is abnormal):

```
In [1]: op.SiT.set_energy()
```

Slit convention as shown in the `mechome` GUI:

```
slit1: XPP slits (UM6)
slit2: MEC slits upstream the hutch CRL, low Z (Upstream)
slit3: MEC slits upstream the hutch CRL, high Z (Downstream)
slit4: MEC slits downstream the hutch CRL, high Z (XT2)
```

For example, to fully open XPP (UM6) slits:

```
In [1]: op.slit1.open()
```

To open XPP (UM6) slits to a square of 2 mm size:

```
In [1]: op.slit1.move(2)
```

To close XPP (UM6) slits to 0 mm:

```
In [1]: op.slit1.close()
```

To open and close the pulse picker:

```
In [1]: op.pp.open()
In [2]: op.pp.close()
```

To set the pulse picker mode to FliFlop:

```
In [1]: op.pp.flipflop()
```

To run the pulse picker at 5Hz in flip/flop mode setting the sequencer properly and get the DAQ setup accordingly with the right delta beams:

```
In [1]: op.pulse_picker(5)
```

The description of this method is the following:

```
def pulse_picker(rate = 5):
        '''
        Description:
            Set the event sequencer to allow for the pulse picker to run at a dedicated rate
    while the DAQ is also set to run (for the cameras to trigger).
        IN:
            rate : default 5 Hz, but can be anything between 0.5, 1, 5, 10, 30, 60. For 120 and
    360 Hz, just open the pulse picker.
        OUT:
            set a sequence in the event sequence and run it
        '''
```

The fast photodiodes can now be read in the DAQ via a digitizer. There are currenlty two functions associated to this device. One to start the control GUI:

```
In [1]: op.digitizer_gui()
```

and one command to plot the out of the diode signal via python command if needed:

```
In [1]: op.digitizer_plot(xmin=0, xmax=32000)
```

with the range of interest being plotted in pixel (one pixel is about 200 ps). The location of the signal should be around `xmin = 5400` and `xmax = 6400`.

5.3 Target chamber: light, shutters

To turn on and off the light in the target chamber:

```
In [1]: op.ligths_on()
In [2]: op.lights_off()
```

To close shutter 1 through 6:

```
In [1]: op.shutter1.open()
In [2]: op.shutter1.close()
In [3]: op.shutter2.open()
In [4]: op.shutter2.close()
In [5]: .
In [6]: .
In [7]: op.shutter6.open()
In [8]: op.shutter6.close()
```

A condensed function to open/close all the shutters from above at once:

```
In [1]: op.shutters_open()
In [2]: op.shutters_close()
```

5.4 Targets and detectors presets

To move the target stages to the correct sample (we use Ti for the timing sample, even if it turns out to be iron when we are above its edge). DOES NOT CHANGE THE ANGLES OF THE HEXAPOD:

```
In [1]: op.yag()
In [2]: op.ti()
In [3]: op.pin()
In [4]: op.pinhole()
In [5]: op.grid()
In [6]: op.ceo2()
In [7]: op.lab6()
In [8]: op.cu()
In [9]: op.zn()
```

You can save the positions of these to the user PVs using the same name but with _s, so for example:

```
In [1]: op.yag_s()
```

This save the current values of the stage to the yag user PV.

To move the 500 mm stage to the NEO position, when set appropriately (can be save in the same way with _s):

```
In [1]: op.neo()
In [2]: op.diode_air()
In [3]: op.gige4()
In [4]: op.spectro_air()
```

When using the regular target cartridges, in a standard configuration experiment, for example, you can use the following commands to move from target to target:

```
In [1]: op.target_up()   # when viewed from Q1, the target physically moves up by 3.5 mm
In [2]: op.target_down() # when viewed from Q1, the target physically moves down by 3.5 mm
In [3]: op.target_next() # when viewed from Q1, the target physically moves to the right by 3.7
    mm
In [4]: op.target_prev() # when viewed from Q1, the target physically moves to the left by 3.7 mm
```

Additionally, you can move by `n` samples using the following type of commands (`n` can be a decimal value):

```
In [1]: op.target_up(2)
```

5.4.1 Move to target

Here is a function to move to a target which is on a frame (designed for single shot XRD standard configuration, but can be used whenever appropriate). It can also disables the visar laser if necessary to avoid saturating the cameras while moving in between the targets. When looking at the target holder from the back (opposite view from Questar 1), frame 1 is closest to the pin (to the left, or south), frame 3 is furthest away from the pin (to the right, or north). The `frame_cfg` defines the current configuration of frames installed on the target holder, with 1 being a full U shape cartridge, and the following character being a user defined tag for this frame. Colmun A is on the left (or south) and column I is on the right (or north). Raw 1 is at the top and raw 7 is at the bottom. Here, we move to frame 1 on target A2, and then to frame 3 target I7, while the configuration has 3 frames called 'TP1, 'TP8' and 'TP3' by the users:

```
In [1]: op.move_to_target(frame_cfg=[1, 'TP1', 1, 'TP8', 1, 'TP3'], frame=1, target='A2',
    visar_disable=True) # TP1 will be the value pushed to the elog for the frame shot
In [2]: op.move_to_target(frame_cfg=[1, 'TP1', 1, 'TP8', 1, 'TP3'], frame=3, target='I7',
    visar_disable=True) # TP3 will be the value pushed to the elog for the frame shot
```

Here is the doctring of the function:

```
def move_to_target(config='colinear', frame_cfg=[1, 'F1', 1, 'F2', 1, 'F3'], frame=1, target='A1'
    , visar_disable=True):
    '''
    Description: script to move to a predefined target in the target holder. Assuming
    for now that columns are letters, and raws are numbers. Columns start from A and
    finish at I from left to right and raws start from 1 to 7 from top to bottom. All
    these while looking at the target holder from the back (opposite view from questar 1).

    Since every target frame is position a few 100um differt due to screw slop, there are two
    epics user pv that can be set as a correction for each target. These corrections should be
    small (<1mm), and are best reset when going to a different frame. They are the user PVs
    57 and 58.

    IN:
        config          : set the experimental configuration to use. It will enable or disable
    arguments accordingly.
                          colinear: for std colienar LPL beam delivery
                          perp: for perpendicular LPL beam delivery
                          long: for colienar beam delivery but using the elongated holes
        frame_cfg       : the configuration of the frames on the target holder as viewed from
                          the back, meaning the opposite view of Q1. TO DO: need to confirm size
                          of the half-U frame.
        frame           : the number of the frame where the targets are located. Can be full size
                          or half size U frame. TO DO: need to confirm size of the half-U frame.
        target          : the number of the target to go to within this frame.
        visar_disable    : disable the visar laser between shots if True to not burn the streak
    whn gain/energy are increased due to poor target reflectivity
    OUT:
        move to target
    '''
```

There are two user PV (57 and 58) that can be set and correct for screw slop in the target frame mounting. These are absolute corrections in mm. There should generally be less than 1mm, and are best set to 0 for a new frame.

We can use this fuction to move to a pillar/target with the sawtooth mount and the MXI instrument. The first pillar closest to the pin is not named. The next pillar is called 'A' and numbering goes alphabetic from there. To top sample is called '1' and the bottom is called '8'. We can move to a pillar sample combination using:

```
In [1]: op.move_to_target(config='perp', target='B6')
```

which will move to target to pillar B, sample 6. Incidentally, the pillars are spaced 11.3mm in X, and the samples are spaced 2.9mm

5.5 VISAR

To display the VISAR remote control PC screens:

```
In [1]: op.visar1_remote()
In [2]: op.visar2_remote()
```

It effectively wraps the following bash commands into python:

```
vncviewer 172.21.46.71 &    # for VISAR 1
vncviewer 172.21.46.88 &    # for VISAR 2
```

Password is `Mechutch`.

How to upload VISAR data to server (call Mike Brown if problems), in a terminal:

```
cd                          # the file .syncmecrc is in /cds/home/opr/mecopr
vi .syncmecrc               # put experimental name in this file, and pres :wq to save and quit
telnet mec-monitor 40000    # press Ctrl X
quit
df /mnt/*                   # to check uploading status
```

5.5.1 in the DAQ

To save the VISAR data in the DAQ:

1. On both VISAR PCs, run the Xserver (icon is an eagle with a wheel): do not touch anything else (the IOC starts the GUI itself)
2. Push the `Reboot` buttons under each GUI where the streak windows are set on the LPL operation GUI at if IOC down (white buttons)
3. Set the Streak EVR to 169 by pressing the small button `DAQ` below the regular buttons of the LPL operation GUI
4. Open the configuration GUI for the VISARs: button `Expert` on the LPL GUI
5. Set the DAQ to beam mode (Laser_Sim would work, but might be less stable)
6. Use the functions described below to take a reference with the DAQ or take a driven shot (`ref_only`, `optical_shot`)

Here are some general remarks on the process:

- you can reboot the IOC at any time, it is harmless
- once up and running, do not touch the VISAR PCs GUI anymore, besides color scales
- whatever is not on the edm GUI can be changed on the remote (like color scales)
- if you close the PC window: you have to restrat the Xserver and then reboot the IOC
- the two IOCs of each VISARs are disconnected
- after an IOC reboot, some settings of the camera might have been badly loaded. If the data window looks to small, make sure that the binning is at 1×1, depth at 16 bit,and slow scan (these are accessible on the button `details` on the acquisition window first tab of the VISAR PCs)

- keep the CCD gain to ~ 150
- you need to set the DAQ to `SHUTDWON` before changing any settings of the VISAR configurations (otherwise it will make the DAQ unhappy)

5.5.2 not in the DAQ

To set the EVR of the VISAR laser and the streak cameras to be ready for an optical laser shot or to do alignment (it also forces the trigger buttons to be enabled so that we do not have to touch the Laser Operator screen at all), or to be saved in the DAQ:

```
In [1]: op.visar_mode(status='align') # set event code 43, to allow for alignment
In [2]: op.visar_mode(status='ready') # set event code 182, to allow for single shot
In [3]: op.visar_mode(status='daq')   # set event code 169, to listen to the daq
```

5.5.3 Alignment

When the visar window is very thick, you need to move the visar lens to accomodate for the path change. With few approximations, the equation to use is:

$$z = d \times \left(\frac{n_2}{n_1} - 1\right) \tag{1}$$

where z is the distance to move AWAY from the current sample surface (by introducing the window, you are moving the 'focus' of the visar further upstream, so the lens needs to move downstream TCC), n_1 and n_2 are the index of refraction of vacuum and the window respectively and d is the thickness of the VISAR window. Here are few index of refraction values at 532 nm:

$$n_{\text{LiF}} = 1.393 \tag{2}$$
$$n_{\text{SiO2}} = 1.4607 \tag{3}$$
$$n_{\text{sapp}} = 1.772 \tag{4}$$
$$n_{\text{ptfe}} = 1.519 \tag{5}$$

For example, if the LiF window is 500 μm thick, you should move 196 μm away from the sample plan. You might have to add to this value the actual imaging plan position to take into account thick target package (100 μm or more).

5.5.4 Etalons

To show the status of visar bed 1 and 2:

```
In [1]: op.bed1.status()
In [2]: op.bed2.status()
```

These use PVs for the etalon length and the white fringes t0, but the *correct stage position* now needs to be put in the motor stage in epics.

5.5.5 Streak Cameras timing

Python controls of the strak cameras is done via three different functions. First, with `op.streak_save`, you can save the current DG box timing value corresponding to a window directly into a PV. Second, with `op.streak_window`, you can change the streak window, and/or add an offset to the basline timing of a streak window. Last, with `op.streak_timing_status`, you can display the overall streak camera configurations and/or save it to the elog. Some details for each of these functions are below, but here are few common features that come together with these functions:

- the two small configuration windows that contained the PVs for each of the streak window timing should not be considered as more than simple displays. It means that in practice we do not need to open and interact with them anymore.

- these small windows were associated with an error-prone process to change the delays when an offset to the initail window timing was necessary. We needed to change the value in these windows, and then push the respective buttons to propagate the expected values to the right channels of the DG box. These two-step process is not necessary anymore since the python commands mentioned above push directly the values to the relevant DG box channels.

The firt command corresponds to the saving process of the timing of a window to a storage PV. The process used to save the channels would starts with moving the rigth streak channel using the GUI (it can be improved lated, bypassing the GUI, but at the moment, it is a convenient way to play with the timing). Once we are happy with where the timing sits, use the following command:

```
In [1]: op.streak_save(visar=1, window=10)
```

This will direclty save the current values of the channel into the PV corresponding to the streak window. Because we save the value directly, it reduces the error we can make by typing them through reading/talking. The header of this function looks like the following:

```
def streak_save(visar=1, window=10):
    '''
    Description: use this command to save timing information for the VISAR window into a
    designated PV value.
    IN:
        visar   : number of the visar, valid values are 1 and 2.
        window  : the window size for which the timing is being added, in ns, valid entries are
    2, 5, 10, 20, 50, 100 and 200.
    OUT:
        push the current timing from a DGbox channel to the right PV value.
    '''
```

This command is working whether the VISAR is being saved through the DAQ or not, so it should be our prefered method to save the timing from now on. Using the DAQ to save the reference file should aslo make our life easier when taking the reference.

The second command corresponds to the change of window and the potential addition of an offset to this window, whether it is positive (the VISAR sees later than t0) or negative (the VISAR sees earlier than t0). Use the following commandM

```
In [1]: op.streak_window(visar=1, window=50, offset=5)
```

In this example, we will set the visar 1 window to a 50 ns range, and add a 5 ns offset so that the VISAR sees later than the original timing. If you don't specify the window, the function will extract the current window from the Xremote application. This function has two features: it change the window on the actual streak camera PC wia the remote application, and it also pushes the right stored values to the DG box. For the remote application, it works also if we are using the DAQ. For example, during a beamtime, the DAQ is open and we need to put the DAQ in shutdown mode before changing the windows of the streak. The function does that for us. It goes in 'shutdown' mode, apply the change in streak window on the Xremote application, and get ready for the next driven shot in such a way that we can run the `optical_shot` script right after within the python environment. If the DAQ doesn't run, it will ignore it. The header of the functions reads like this:

```
def streak_window(visar=1, window=None, offset=0):
    '''
    Description: allows to change the VISAR streak window and/or add an offset to the current
    visar streak window.
    IN:
        visar   : the number of the visar to consider, 1 and 2.
        window  : set the visar window, in ns, valid entries are 2, 5, 10, 20, 50, 100 and 200.
    If no window is provided, the current window is used.
        offset  : delay in ns to add to the current window settings.
    OUT:
        push the offset to the DGbox
    '''
```

Finally, you can print the status of the VISAR streak timing with the following functionm

```
In [1]: op.streak_timing_status()
In [2]: op.streak_timing_status(verbose=True)
In [3]: op.streak_timing_status(save=True)
```

The default version just print out the current timing window and offset in use for both VISARs. Adding the verbose gives the saved PV for initial t0 if you want to check that nothing is wrong. The detailed info can then be printed to the elog adding the `save=True` flag. The header is the following:

```
def streak_timing_status(verbose=False, save=False):
    '''
    Description: displays the status of the timing configuration for both VISAR system. It will
    require to have the Xremote up and running.
    IN:
        verbose : add some more info on the timing configuration.
        save    : push the entire current configuration to the elog.
    OUT:
        displays timing info for the VISAR system
    '''
```

5.5.6 TODO

- add maximum threshold of 30000 on the python viewer and color code this red so we can check for camera saturation level
- add the streak configuration to the shot script so user can log the streak status

6 Scripts

6.1 For single shot: basic use

Going to continuous 120, 60, 30, 10, 5, 1, and 0.5 Hz can now be done in either `Beam mode` and `LaserSim` in the daq, using (here, going to 120 Hz):

```
In [1]: x.start_seq(120)
```

If it does not work, try running it again, and/or go to the event sequencer and start the sequency running continuously. For shooting the nanosecond laser in single shot, we use:

```
In [1]: p = x.nsl.shot(record=True, end_run=True)
In [2]: RE(p)
```

x stand for 'experiment', nsl for 'nanosecond laser'. The first line defines the variable 'p' which is a bleujean python plan. Second line runs the plan. p is an itterator, so needs to be recreated again after it is executed (it is exhausted after each use). `RE` stands for Run Engine (it exhaust the plan p). To stop a execution, press CTRL-C twice, and the RE.abort().

6.2 For single shot: untested options

THESE ARE NOT ALL TESTED! You can specify the prex, predark etc. To see the full scope try:

```
In [1]: x.nsl.shot?
```

To show the config of the shot:

```
In [1]: x.nsl.config
```

This can be changed by the commands below.

For example, to set the predark parameter to 1 in the config. `prex`, etc. have the same command structure:

```
In [1]: x.nsl.predark = 1
```

If you have need to change between a few configurations, that are sufficiently different, you can save a configuration in a local variable:

```
In [1]: config1 = x.nsl.config
```

And later you can reload it with:

```
In [1]: x.nsl.configure(config1)
```

The way to run the femtosecond laser, is the same, but with `x.fsl.shot()`.
You may need to connect/disconnect from the daq at times, or end the run manually with:

```
In [1]: daq.connect()
In [2]: daq.disconnect()
In [3]: daq.end_run()
```

6.3 Scanning scripts with short pulse laser

DAQ settings prior any scan: laser sim , allocate, DO NOT 'Begin Running'
To set the spacing of the targets and their compensations, and some additional commands: factor per target, in mm

```
In [1]: x.grid.x_spacing = -0.363     # negative to move target south (go to the left target)
In [2]: x.grid.y_spacing = 0.363      # positive to move target up (go to the bottom target)
In [3]: x.grid.x_comp = 0.0032        # for X offset when moving vertical: when line is off
    clockwise, value neeeds to be positive
In [4]: x.grid.y_comp = 0.00286       # for Y offset when moving horizontal: when line is off
    counter-clockwise, value neeeds to be positive
In [5]: x.grid.up()                   # move the target up to bring the target below current at
    TCC
In [6]: x.grid.down()                 # move the target down to bring the target above current at
     TCC
```

For 1D scanning: Starting taking a shot where it is currently located. At the end of the scan, returns to original location. xrays=True means Xrays will be shot, if false it would be an optical only shot:

```
In [7]: p = x.x_scan(nshots=1, record=True, xrays=True)
In [7]: RE(p)                         # will start the scan so make sure all the EVRs are set
In [7]: daq.end_run()                 # end the run itself, but cannot change the DAQ status
In [7]:  daq.disconnect()              # end the run and disconnect the user end so we can change
     the DAQ status
```

For 2D canning: Take a 2D scan, first moving in X then in Y. When 'carriage return' is true, the motors come back to initial positions:

```
In [7]: p = xy_scan(nxshots=40, nyshots=3, record=False, xrays=False, carriage_return=True)
In [7]: RE(p)
```

6.4 X-ray only shots

The following command is used to save a single dark image of the detectors, and shoot 10 X-ray pulses at 1% attenuation, while saving the resulting run (VISAR must not be in the partition in the shown case):

```
In [1]: op.ref_only(xray_trans=1.0e-2, xray_num=10, save=True, visar=False, rate=10)
```

It does also save the target postions on which the X-ray only has been taken. It saves the target sample type, not the XYZ positions (this is saved automatically in the DAQ).

6.5 X-ray only calibration shots

Moves to calibrants (like `CeO2`, `LaB6` , `Ti`, `Cu` and `Zn`), and save a run with 10 x-ray only shots taken at 1% attenuation (it will also save visar references at the same time, event if irelevant):

```
In [1]: op.ref_only(xray_trans=0.01, xray_num=10, calibrant='Cu', visar=True, save=True) # CeO2,
    ceo2, LaB6, lab6, Ti, ti, Cu, cu, Zn and zn are valid entries
```

The option rate is set to 1 by default. It should work well for standard configuration but will need tweaking when other expeeriments are running. Setting the rate at 1 allows:

- to take calibration shots without having to change anything on the VISAR if it is set in the DAQ

- to easily move manually around the calibrant default position to get to a fresh spot if the users request it

If the VISAR is not set in the DAQ (unselected from the partition) and you only need < 50 shots, setting the rate to 10 Hz is good enough. The full list of options and their descriptions are the following:

```
def ref_only(xray_trans=1, xray_num=10, shutters=False, dark=0, daq_end=True, calibrant='', rate
    =1, visar=False, save=False, slow_cam=False):
    '''
    Description: script to take xray only events and/or VISAR references.
    IN:
        xray_trans : decimal value of the xray transmission
        xray_num   : number of x-rays to send on target
        dark       : default is 0, we do not record a dark run before the reference
        shutters   : default is False, we do not need to close the shutters for references
        calibrant  : if not empty, will move to specified calibrant and take calibration run only
        visar      : True if you want to take visar references
        daq_end    : close the run at the end of a shot. Set to True allows a user to see the
    result of the shot for longer.
        rate       : rate used to take the reference, it is set to 1 by default bu is changed
    depending on the options (visar, calib)
        save       : True to save to the DAQ, False otherwise
    OUT:
        execute the plan
    '''
```

Alternative way to take X-ray calibration shots (redundant with `ref_only` function, so obsolete):

```
In [1]: op.xray_calib(xray_trans=0.01, xray_num=10, calibrant='CeO2', rate=1, save=True)    #
    CeO2, ceo2, LaB6, lab6, Ti, ti, Cu, cu, Zn and zn are valid entries
```

6.6 X-ray and VISAR references saved on DAQ

The following command is used to save a single dark image of the detectors, and shoot 10 X-ray pulses at 1% attenuation, taking 10 visar references while saving the resulting run:

```
In [1]: op.ref_only(xray_trans=1.0e-2, xray_num=10, save=True, visar=True)
```

The conditions of Section 5.5.1 shall be met before runnign this command since you will use the VISAR.

6.7 Pump-Probe LPL shots (XRD configuration)

To set all the arms of the optical laser with 8% of its total maximum energy, at a delay of 18 ns vs the X-rays, with the Si attenuator transmission set at 30%, while adding a message `template` with the tag `sample` and making sure the triggers of the NS slicer and the lamps are enabled, and that we are charging automatically while saving the visar to the DAQ:

```
In [1]: op.optical_shot(lpl_ener=0.08, timing=18.0e-9, xray_trans=0.3, arms='all', msg='template'
    , tags_words=['sample'], auto_trig=True, auto_charge=True, visar=True)
```

If `autocharge=True`, the script waits until it has premission to charge from the PFN epics GUI, and then executes. Therefore, it can be run whenever the target is aligned, and references have been taken. This script takes 40 sec to execute up to the laser shot and an additional 60 sec to clean up. So, the script can be started at 6 min 20 sec on the charge clock and the shot will be done at 7 min. It means you can start the charging at 6 min on the charge clock. The `msg` and the `tags_words` arguments can be ignored if only the target positions are to be loaded to the elog as it does use the target `move_to_target` position to automatically push the target positions. The full list of options and their descriptions are the following:

```
def optical_shot(shutter_close=[1, 2, 3, 4, 5, 6], lpl_ener=1.0, timing=0.0e-9, xray_threshold
    =0.1, xray_trans=1, prex=0, save=True, daq_end=True, msg='', ps_opt=True, arms='all',
    tags_words=['optical', 'sample'], uxi=False, auto_trig=False, auto_charge=False, visar=True,
    slow_cam=False, debug=True):
    '''
    Description: script to shoot the optical laser and time it with the xrays. It automatically
    push to the elog the laser energy, the timing and the xray SiT transmission.
    IN:
        shutter_close    : array of shutters in front of viewport to close during laser shots
        lpl_ener         : waveplate settings for the lpl energy, decimal value, meaning 1. =
    100%, 0.5 = 50%
        timing           : moves absolute, in s
        xray_threshold   : threshold energy in mJ below which the shot is NOT proceeding and loop
    until FEL energy is back
        xray_trans       : X ray transmission, meaning 1. = 100%, 0.5 = 50%
        prex             : when True, allows to take one Xray or visar reference
        save             : save the run when True (default).
        daq_end          : if True, it will allow the DAQ to keep the data on screen until daq.
    disconnect() is used.
        msg              : message to post to the elog
        ps_opt           : if True, enable the optimization of the pulse shaping routine
        arms             : all, ABGH, EFIJ are valid
        tags_words       : accompagnying tags to the elog
        uxi              : when True, it sets the repetition rate of the sequencer to 0.5 Hz so
    that the pulse picker delta beam are properly calculated
        auto_trig        : True to make sure the triggers are enabled, False otherwise (simulation
     test for example).False by default.
        auto_charge      : True to charge automatically the PFN. False by default.
        visar            : True to check that the VISAR triggers are set properly.
        slow_cam         : Change the state of the configuration file for slow devices like
    cameras or gas jets
        debug            : True to enable debugging functions
    OUT:
        execute the plan and post a comment to the elog.
    '''
```

Here are few steps to save references with both the X-rays and the VISAR simultaneously, then take a driven shot:

1. move to target, e.g.:

   ```
   In [1]: op.move_to_target(frame_cfg=[1, 'TP1', 1, 'TP8', 1, 'TP3'], frame=3, target='I7')
   ```

   a) tweak front position for drive access using Questar 1
   b) tweak back position for visar quality using Visar1 or 2 gige cameras and the streak images (either from the remote PC screens which are open, or from the python viewer: one day we should be able to close the remote access, but for now we keep it to keep an eye on how it performs).

2. take references, e.g.:

   ```
   In [1]: op.ref_only(xray_trans=1.0e-2, xray_num=10, save=True, visar=True)
   ```

   - you do not have to pay attention to the triggering status of the visar since it checks it automatically
   - you can still just take X-ray only references without VISAR reference by setting the visar argument to `False`

3. at 6 min on the PFN clock, take a driven shot, e.g.:

   ```
   In [1]: op.optical_shot(lpl_ener=0.08, timing=18.0e-9, xray_trans=0.3, arms='all', msg='
       template', tags_words=['sample'], auto_trig=True, auto_charge=True, visar=True)
   ```

   - you do not have to pay attention to the trigger status in general for both the visar and the laser if you use `auto_trig`,`auto_charge` and `visar` set to True.
   - elog entries are set and you get ready for the next shot

6.8 Pump-Probe LPL shots in perpendicular geometry using the UXI detector

The UXI detectors have been tested with the perpendicular beam delivery and the LPL laser to allow for direct Xray imaging with the MXI device. A dedicated script was made to provide warming up sequence used for this detector. The following command allows to create 4 runs make of 10 dark and white fields when the target is out, then 10 white firelds with the target in, followed by a driven shot, finishing by 5 dark images:

```
In [1]: op.uxi_shot(save_run=True, target_out_dark=10, target_out_white=10, target_in_white=10,
    post_dark=5, offset = -0.5, lpl_ener_val=1.0, timing_val=0.0e-9, arms_val='all')
```

with te detailed description as follows:

```python
def uxi_shot(save_run=True, target_out_dark=10, target_out_white=10, target_in_white=10,
    post_dark=5, offset = -0.5, lpl_ener_val=1.0, timing_val=0.0e-9, arms_val='all'):
    '''
    Description:
        Creates a complexe sequence of dark/white fields with and without the target when using
    the UXI detectors.
    IN:
        save_run            : True if run is to be saved. Default is True.
        target_out_dark     : number of dark images with target OUT. Default is 10.
        target_out_white    : number of white field images with target OUT. Default is 10.
        target_in_white     : number of white field images with target IN. Default is 10.
        post_dark           : number of dark images after the driven shot. Default is 5.
        lpl_ener_val        : LPL total energy output. Default 1.0 (full energy).
        timing_val          : delay between the LPL and the FEL. Default is 0.0e-9 s.
        arms_val            : laser arms to shoot. Default is 'all'.
        offset              : offset in mm to move the sample from its current position. Default
    is -0.5mm on hexapod Z.
    OUT:
        Runs the complex sequence, saving 4 runs, one with dark and white and target out, then
    white with target in, driven shot and dark after shot.
    '''
```

7 Data analysis

7.1 Image extraction

If a user is looking for tiff images from runs and the automatic tiff image converter is not working (might be due to network or server issues), you can run the following command to extract tiff images from Opals, Zyla, ePix10k and ePix100 detectors. after login with your unix credential and `ssh psana`, use the following commands:

```
source /reg/g/psdm/etc/psconda.sh
/reg/g/psdm/sw/tools/smalldata_tools/mec/examples/smalldata_producer.py --full --image --tiff --
    experiment meclx5319 --run 3
```

This example shows how to extract all the images from run 3 in the meclx5319 experiment. More options are available using:

```
/reg/g/psdm/sw/tools/smalldata_tools/mec/examples/smalldata_producer.py -h
```

This script has been written by Silke, so you can contact her if you have any problems.

8 Versions

Table 1 used to keep track of the changes in this document.

Table 1: Version control for this document.

revision #	date	changes
9	2024	∘ Added ACR info, zoom process ∘ Updated pulse shaping scripts ∘ Updated shot scripts
8	2023	∘ Not sure what was added....blip in time
7	2022	∘ Added pulse picker and UXI related functions ∘ Added debugging functions for `optical_laser` script ∘ Added functions related to pulse shape (up to date) and shutters
6	2021	∘ Added remote commands for VISAR vnc's and scope for timing ∘ Added minor arguments in LPL command ∘ Reorganized script sections ∘ Added data analysis section
5	2021	∘ Added revision table ∘ Removed code listings (too cumbersome) ∘ Update mecpython code access (no need to cd to directory) ∘ Cleaned `mecpython` use ∘ Added new function to take calibrant shots ∘ Added VISAR saved in the DAQ ∘ Added Cu and Zn references on the alignment cartridge ∘ Added new options for `ref_only` and `optical_shot` ∘ Added VISAR streak timing functions
4	2020	∘ Added std configuration operation functions ∘ Added presets for beamline access ∘ Added rolling status prototype
3	2020	-
2	2017	-
1	2015	-
0	2014	Original release