HXR FEE Commissioning

The experiment for the HXR FEE commissioning is xppx43118.

If you are in the XPP control room, you find the usual operator session open.

For running the DAQ remotely, you need to ssh to one of the two XPP machines (xpp-daq or xpp-control) as xppopr from psdev (ssh xppopr@xpp-daq). Let Silke know if this does not work for you and you think it should.

I would suggest to work with two open sessions, one for the Beamline python (started via xpp3 in one of the terminals) and one for other scripts. Most of the other script will also work from lfe-console, but you should have write permissions from XPP as well. The most important control scripts are described in the Controls User Guide. The relevant ones are listed here:

- camViewer <-h>: starts the python viewer. You can pass the relevant camera by supplying via the '-c <camname>' argument. Adding '-m' will start the edm screen for the relevant area detector
- iocmanager <lfe> (turns out, you need to be on lfe-console to see most of the processes)
- eloggrabber <-h>: by defaults starts the operator eloggrabber of the hutch that is guessed by the hostname or pathname. '-u <username>' will start the eloggrabber as yourself, but you need to start it from a session in which you have a personal kerberos token.
- lfe : 'lucid' home screen for the FEE

The DAQ operation is described in detail in the DAQ User Guide.

You can add any 'point detector' (single value) to the DAQ EPICS archiver. I suggest to add them to /reg/g/pcds/dist/pds/xpp/misc/epicsArch_feeCam.txt. This file is owned by xppopr. The data available in the DAQ can be used to e.g. make scatter plots of variable A versus variable B (e.g. intensity seen on the imager versus a mirror motor, ...). Imager data can be saved this way when condensed to a single number (e.g. mean from the stats plugin or the centroid x or y).

We are investigating the use of the iocrecorder capabilities: the by far most convenient way to do this is to include these cameras/waveforms(gas detector or imager projections) in the DAQ via the .iocrc file. This data will show up in the xtc file, but NOT in ami.

We have set up automatic translation of the recorded xtc files to hdf5 files. You can check on the status of this translation in the 'workflow/control' section of the data manger web interface. These files will end up in /reg/d/psdm/xpp/xpp43118/hdf5/smalldata . xpp3 also has function to record the faster data of the powermeter as well as hdf5 files with imager data. These files should be rsync'ed to /reg/d/psdm/xpp/xpp43118/usrdaq.

A few python notebooks will be provided in /reg/d/psdm/xpp/xpp43118/results/smalldata_tools/notebooks. If you want to use those, I will suggest a softlink of this directory from your home as the Jupyterhub will also ways starts from your home area.

Samples

The new elog allows us to define 'samples' or run groups. For e.g. the run tables & elog, you can toggle between the run groups or show all the data.

Beamline Python (xpp3)

Components in the lfe beamline will be provided in xpp3. Their names follow the standard device names w/o caps (e.g. im1l0). We have also defined a few convenience functions which will be deceived below (the complete list can be found by typing x.<tab>:

x.savePowermeter(colltime=None, rate=None, filename=None, pwm=None):

- colltime <in sec>: time we will record data for
- filename: name of output file, will default to xppx43118_Run<xxx>_IM3L0_powermeter.data where xxx is the next run to be taken
- rate: data comes at 1Hz and contains 1000 traces. By default, the data is reduced to 10 Hz, but you can also pass 100 for only a 10-fold reduction or anything else for full rate/

x.takeRun(nEvents, record=True):

- runs the DAQ for nEvents with or without recording

x.get_ascan(motor, start, end, steps, nEvents, record=True):

- returns a 'plan' to move motor <motor> in <steps> steps from <start> to <end>, taking <nEvents> at each step. This will take control of the DAQ. To actually RUN the scan, you can either
  - myscan=x.get_ascan(...); RE(myscan); do something ; RE(myscan) or
  - RE(x.get_ascan(...))
- you can scan any motor and most other variables that you have permissions to change as well. If in doubt, test & ask if there are issues.

x.get_dscan(motor, start, end, steps, nEvents, record=True):

- relative scan version: here, we scan from <current-start> to <current+start>

x.im1l0_h5.prepare(baseName=None, pathName=None, nImages=None, nSec=None):

- by default, files will be save in the /reg/d/iocData/<imager-ioc>/images directory and rsync'ed to the usrdaq folder
- unless specified, the filename are xppx43118_Run<xx>_<idx> where <xx> is the upcoming run when 'prepare' is run
- nImages: specify the number of frames to be saved
- nSec: specify the number of seconds to record. We will use the acquiring rate to calculate how many images to record.

x.im1l0_h5.write()

- starts the acquisition & recording

x.im1l0_h5.wait()

- will block until the recording is finished.

x.im1l0_h5.status()

- prints relevant information about the hdf5 plugin.

x.im1l0_stats.status()

- prints relevant information about the stats plugin

x.im1l0_stats.prepare(threshold=None)

- enables the plugin
- if a threshold is provided, start to compute the centroid (and thresholded projections)

x.im1l0_stats.stop()

- stops the stats plugin (this plugin can hog CPU...)

x.im1l0_stats.setThreshold(nSigma=1):

- sets the threshold for the centroid/projection to the (current mean + nSigma* current noise)

The im10l0 functions also exist for the other imagers.