Cheetah configuration and documentation

Table of contents

Downloading, compiling and installing Cheetah

Cheetah at LCLS
http://www.desy.de/~barty/cheetah/Cheetah/Configuration.html

Cheetah at CFEL/DESY
Cheetah is insstalled in /cfel/common.
`> cheetah-gui should just work provided /cfel/common/bin is in your path.`
More instructions to follow (or ask Anton Barty)

Cheetah elsewhere
At any other location you will have to install Cheetah from scratch.Installing Cheetah itself is not too hard; however installing the LCLS framework required to read XTC files directly can be an adventure. Your mileage may vary. Please see the developer pages for details on installing Cheetah from scratch.

Alternatively, if your data comes from somewhere other than LCLS, Cheetah can be called from code able to read any other file format: it is simply a matter of passing the frame data to Cheetah for processing. Once again, see the developer pages for more details.

Cheetah for developers

Cheetah is open-source and has been released under the GNU GPL v3 license. The latest releases and updates Cheetah are best downloaded from the Github repository: https://github.com/antonbarty/cheetah/

Please follow the download instructions on that page: (assuming you have a version of git already installed)
`> git clone git://github.com/antonbarty/cheetah.git`

Please refer to the website for further details: http://www.desy.de/~barty/cheetah/Cheetah/Developers.html

Updates

Check here for most recent updates to Cheetah
http://www.desy.de/~barty/cheetah/Cheetah/Changes.html

Running Cheetah

Cheetah at LCLS

The pre-installed Cheetah package at LCLS is in /reg/g/cfel/cheetah/cheetah-latest
Please follow the instructions at http://www.desy.de/~barty/cheetah/Cheetah/Cheetah_at_LCLS.html for getting Cheetah running on your data at LCLS. Cheetah has a very handy GUI for launching batch hit finding jobs, keeping track of hit finding results, generating darkcals and bad pixel masks from them and viewing hits.
This is the most reliable route for using Cheetah at LCLS.

After you run `tar -xvf /reg/g/cfel/cheetah/template.tar` in your scratch/<username> directory, the sub-directories created will include:

calib

Calibration files: beam (for beam files used by older versions of CrystFEL), darkcal – where you should store the darkcals created by Cheetah; gaincal – for gain calibration files; geometry – geometry files; and mask – where to store bad pixel masks, peak masks etc.

gui

Files needed by cheetah-gui. You will need to modify crawler.config before running. Instructions on what to change are on the 'Cheetah at LCLS' web page.

hdf5

Location where output from Cheetah will be saved: HDF5's (diffraction data) and a bunch of hit finding configuration files. A separate directory, `rXXXX—<tag>` is created for each run and each tag (so you can try different hit finding parameters without overwriting. The amount of "clean" data grows quickly, so remember to delete all but your best hit finding results when finished.
XXXX is the run number and <tag> is the name of your ini file if you launch jobs from the Cheetah GUI, or a user-specified tag when launching from a terminal using `./process <run> <inifile.ini> <tag>` (*process*, the script, can be found in your `cheetah/process` directory).

indexing

Location for output from CrystFEL indexing launched from the Cheetah GUI. See "lys.crystfel" script in the process directory.

process

Location for hit finding configuration (.ini) files.

- "process" sets up the environment variables and launches Cheetah
- "psana.cfg" is the configuration file for psana, the LCLS analysis framework (C++ and python).
- "lys.ini" an example .ini file.
- "darkcal.ini" – an ini file for generating a dark current measurement from a "dark" run; this doesn't need to be edited.

Configuration (.ini) files

Cheetah behavior is specified by the user through a configuration file. An example configuration file, `lys.ini,` is provided in cheetah/process directory. Within a configuration file is a list of "keywords" that cheetah recognizes, and the user-specified values. There are two types of keywords; "global" keywords that affect the analysis of all data, and "detector" keywords that affect only one particular detector.

Global keywords may be specified in the following way:

```
keyword = value  # comment
```

Note that whitespace is ignored completely, and everything following a # symbol is ignored. Keywords are not case sensitive, and if a keyword is unrecognized by Cheetah the program will exit. Look in the log file (typically .. scratch/<username>/cheetah/hdf5/rXXXX–tag/log.txt) for which keyword was not recognized.

Detector keywords may be grouped together. One way to group keywords is to use forward slashes, as follows:

```
group1/keyword1 = value
group1/keyword2 = value
group2/keyword1 = value
group2/keyword2 = value
```

The labels `group1` and `group2` can be any word. An alternative way to specify groups is the following:

```
[group1]
keyword1 = value
keyword2 = value
[group2]
keyword1 = value
keyword2 = value
```

Generally, the use of brackets will simply prepend the group within the brackets to all subsequent keywords. Empty brackets are allowed, which would specify global keywords. Detector keywords that have not been assigned a group will automatically be assigned to the "first" detector.

Cheetah will ultimately be capable of performing peak finding / hit finding on multiple detectors. At the moment, these operations will only be performed on the first detector in the configuration file.

Setting up Cheetah and the crystfinder script

- Configure cheetah.ini
    - Select the right detector
    - Select background processing options
    - Tune hit finding parameters

Most commonly adjusted keywords

These are the most important keywords you'll probably ever want to tweak – the rest can likely be left alone.

Detector configuration

- **geometry (geometry/cspad_pixelmap.h5)**

Calibration and masks

- **darkcal (darkcal.h5)**
- **badPixelmap (badpixelmap.h5)**
- **peakmask (peakmask.h5)**

Background subtraction

- **useRadialBackgroundSubtraction (1)**
- **useSubtractPersistentBackground (0)**
- **useLocalBackgroundSubtraction (0)**

Hit finding

- **hitfinderADC (150)**
- **hitfinderMinSNR (6)**
- **hitfinderNPeaks (20)**
- **hitfinderNpeaksMax (5000)**
- **hitfinderMinPixCount (2)**
- **hitfinderMaxPixCount (20)**
- **hitfinderLocalBgRadius (2)**

Tuning hit finding parameters

Optimising crystal hit finding

- Set hitfinderADC low enough, but not too low.
- Is there a jet streak or a bad detector region —> put it in the peak mask
- Too many spots in the solvent ring —> increase the hitfinderSNR or hitfinderMinPix
- Too few spots overall —> decrease hitfinderSNR and/or decrease the number of pixels per peak (depending on what you see for the spots not being found, too small, or too weak)
- Blank frames with little noise, finding peaks all over the place —> increase hitfinderADC (which acts as a floor on the ADC threshold computed from the radial SNR profile)
- Still stuck with too many peaks —> try restricting the radii over which hit finding is performed using hitfinderMinRes and hitfinderMaxRes (in pixels)
- It is convenient to start a new .ini file for each type of sample. The name of the .ini file is used by the GUI to tag runs and update the table, and ends up as the tag name on the HDF5 directories created. Separate names helps keep separate samples apart, and makes it easy to copy/tar/grep directories based on sample name or other eperiment parameters. This helps keep things organised. Use a symbolic link if the files are really the same.
- Review your output. Often. No analysis should ever be done completely blind. Use the 'Show hits" button to look at images and refine the hit finding parameters.

Optimising processing speed

- Set nthreads to 16 (on LCLS and most other servers) or 72 on cfelsgi
- Check I/O speed limit using ioSpeedTest
- Turn off powder pattern creation (which skips mutex locks around summation of powder patterns)
- Increase amount of time between calculation of running background (recalculation mutex blocks all worker threads) or turn off running background completely
- Increase saveInterval
- set hitfinderFastScan to 1 – it will search only the inner 16 panels (of CSPAD's 64)

Complete listing of keywords

Many of these are power user settings designed for turning on or off features in testing.

Detector configuration

detectorName (CxiDs1)

Recognized options for **cspad** are:

- "CxiDs1": CSPAD,
  - typically in position DsC, “front detector” in 1 µm chamber at CXI
- "CxiDs2": CSPAD
  - typically in position DsD, i.e. “back detector” in 1 µm chamber at CXI, downstream of Ds1
- "XppGon": cspad on XPP beamline

CxiDs**1** is the name of the CSPAD (Cornell SLAC pixel array detector). Talk to your beamline scientist to confirm.

Recognized options for **pnccd** are:

- Hooks in place, bit not yet implemented

detectorType (cspad)

Recognized options are:

- "cspad"
- "pnccd"

This field could be deleted – specifying detectorName should be enough to set this internally

geometry (geometry/cspad_pixelmap.h5)

Path to an hdf5 file specifying the real-space coordinates of each pixel. The hdf5 data fields are /x /y and /z. The x coordinate corresponds to data fast scan coordinates (pixels that are nearest neighbors in memory), while y is slow scan. The z coordinate is the relative offsets of each detector panel. Units are meters (values are first divided by pixel size variable to get coordinates of pixels in space).

Note: see the keyword pixelSize-- it is important that this is consistent with the units in the geometry!

pixelSize (0.000110)

Size of pixels in meters, defaults to values for the CSPAD detector (110 µm).

defaultcameraLengthmm (100)

The default camera length (in mm) to use in the event that the detector position encoder value is not available in the XTC data stream.

cameraLengthOffset (592.0)

Offset (in mm) to add to value from detectorZpvname to get absolute sample-detector distance.

cameraLengthScale (0)

defaultphotonenergyev

The default photon energy (in eV) to use in the event that the beamline data is not available in the XTC data stream.

detectorZpvname (CXI:DS1:MMS:06.RBV)

The LCLS EPICS process variable name needed for accessing the encoder value for the camera length of the detector in position DS1. This is the front detector encoder value for experiments in the 1 µm chamber.
- DSA for the position immediately after the SC01 (aka SC2) chamber for the 100nm focus.

- DSB for the position in front of the SC1 chamber
- DSC for the front detector position immediately after the SC1 chamber (1 µm focus)
- DSD for the back detector position

DS[0-9] are the CSPAD labels, DS[A-Z] are CSPAD locations. Typically, we expect to have the following configuration:
- DS2 in the DSC location
- DS1 in the DSD location

Calibration and masks

darkcal (darkcal.h5)

Path to an input hdf5 file containing a dark current measurement. Cheetah can create a "darkcal" from a dark run; see the generateDarkcal keyword. The hdf5 data field is "/data/data". Units are ADU. Darkcals should NOT be gain corrected. Make sure the detector gain settings of the darkcal match that of the run.

gaincal (gaincal.h5)

Path to an input hdf5 file containing the gainmap. By default the raw data will be multiplied by this map, although it can be inverted by setting invertGain=1. The hdf5 data field is "/data/data".

invertGain (0)

Divide by the gain map, rather than multiplying (in case gain map is supplied as gain per pixel, rather than value to multiply pixel values by).

peakMask (peakmask.h5)

Path to an input hdf5 file indicating where not to search for peaks, but without masking out the region which "badpixelmap" does. The hdf5 data field is "/data/data".

badPixelmap (badpixels.h5)

Path to input hdf5 file indicating bad pixels which will be masked in clean hdf5's. Essentially, this has the same effect as a gainmap. The hdf5 data field is "/data/data"
Cheetah's GUI can generate a bad pixel map automatically from the darkcal.h5. See step 4 on
http://www.desy.de/~barty/cheetah/Cheetah/Cheetah_at_LCLS.html

savePixelMask (1)

Save a copy of the pixel mask as badpixelmap.h5 in the HDF5 directory.

maskSaturatedPixels (0)

Search each image for saturated pixels, and mask them prior to further analysis. Saturated pixels are identified by a simple global threshold value set by the keyword pixelSaturationADC.

Background subtraction

Proper subtraction of electronic and photon background is essential – there are many options.

CommonModeCorrection (asic_unbonded)

Unbonded pixels only for cspad v1.6 modules (January 2014 onwards)
commonModeCorrection = {none, asic_histogram, asic_median, asic_unbonded}
One of three possible methods for subtracting common mode offsets from individual ASICs.
Common mode noise on each ASIC fluctuates randomly from frame to frame and must be estimated from the read out signal itself. Common mode is estimated as the lowest 10% of pixel values in each ASIC. 10% value can be set to something else by the user if desired.
This option assumes the lowest 10% of values represent only detector electronic noise - be careful of using this when there are no dark areas on the ASIC.

Pixel locations are hard coded for testing (generalize later)

subtractUnbondedPixels (0)

cmFloor (0.100000)

Use lowest x% of values as the offset to subtract (typically lowest 2%)

~~**subtractBehindWires (0)**~~

~~One of three possible methods for subtracting common mode offsets from individual ASICs.~~
~~For some experiments thin wires are placed in front of the detector and cast shadows; the counts behind shadows are used to estimate common mode offsets on each ASIC.~~

~~**wireMaskFile (wiremask.h5)**~~

~~Path to input hdf5 file with binary mask specifying pixels behind wires to be used for background estimation. These wires are placed in front of the detector and cast shadows; the counts behind shadows to determine common mode noise.~~

useAutoHotpixel (0)

Automatically identify and remove hot pixels. Hot pixels are identified by searching for pixels with intensities consitently above the threshold set by the keyword hotpixADC. In this case, "consistently" means that a certain fraction (user-set keyword hotpixFreq) of a certain number of buffered frames (number of frames set by the keyword hotpixMemory) are above threshold. The hot pixel map is updated every hotpixMemory frames.
Hot pixels within the corrected data will be set to zero. Note that the search for hot pixels is performed on the correcteddata (probably it should be performed on raw data instead?), so if you decide to change the corrections (e.g. darkcal, gainmap), the resulting hot pixel maps may differ.

useSubtractPersistentBackground (0)

Subtract the pixel-by-pixel median background calculated from the previous N frames (N set by the keyword bgRecalc, but apparently it cannot exceed 50 frames – it can, if bgbuffer is made large enough).

useLocalBackgroundSubtraction (0)

Prior to peak searching, transform the image by subtracting the median value of nearby pixels. The median is calculated from a box surrounding each pixel. The size of the box is equal to localBackgroundRadius*2 + 1
Bad pixels and detector edge effects are not accounted for (i.e., if most nearby pixels are bad, the local median will be equal to zero). This is somewhat slower, but very effective for nanocrystal data.

localBackgroundRadius (3)

See keyword useLocalBackgroundSubtraction.

Background calculation tuning

bgMemory (50)

See keyword useSubtractPersistentBackground.

bgRecalc (50)

Strange, this *almost* does the same thing as bgMemory, butif bgRecalc is less than the default value of bgMemory, that default value will be used?
This sets how often the program pauses to recalculate background and hot pixel values. It is typically the same as the buffer size, but since recalculation is a thread blocking process, setting this to happen less frequently (eg: every 200 or 500 frames) speeds up execution.

bgMedian (0.5)

Rather than using the usual median value for background, you can optionally choose any arbitrary K-th smallest element equal to bgMedian*bgMemory. Neat!

bgIncludeHits (0)

Include hits in the background running buffer.

bgNoBeamReset (0)

Used to reset the background buffer whenever LCLS beam dies, defined as when GMD<0.2mJ. (not implemented)

bgFiducialGlitchReset (0)

Used to restart backgrounds when LCLS unexpected changes operating frequency. This was a problem on hot days in the June 2010 data set. (not implemented)

Automatic hot pixel calculation

hotpixFreq (0.9)

See keyword useAutoHotPixel.

hotpixADC (10000)

See keyword useAutoHotPixel.

hotpixMemory (50)

See keyword useAutoHotPixel.

Pixel saturation

maskSaturatedPixels (0)

Search each image for saturated pixels, and mask them prior to further analysis. Saturated pixels are identified by a simple global threshold value set by the keyword pixelSaturationADC.

pixelSaturationADC (65535)

See keyword maskSaturatedPixels.

Hit finding algorithms

hitfinder (0)

Specify the hitfinder algorithm. Various flavours of hitfinder:

1 - Number of pixels above ADC threshold
2 - Total intensity above ADC threshold
3 - Count Bragg peaks (intensity threshold)
4 - Use TOF
5 - Count Bragg peaks (threshold + gradient + extras)
6 - Count Bragg peaks (based on signal-to-noise ratio)
7 –
8 – Count Bragg peaks (based on signal-to-noise ratio with radial background averaging)

Note that the choice of hitfinder influences what is reported in the log files.

First, do these steps (regardless of the hitfinder choice): Build a buffer, which is a replica of the corrected data. Values in the buffer array will be set to zero as those pixels are analyzed and rejected. If hitfinderUsePeakmask != 0, then multiply the buffer by this array before moving on.
Now, depending on the algorithm, do these steps:

Algorithm 1:

1. Count the pixels within the buffer that are above the (user-defined) hitfinderadc value.
2. Also, sum the values of the pixels that meet the criteria of step 1.
3. If the number of pixels is greater than the value of (user-defined) hitfindernat, count this frame as a hit.
4. Report the number of pixels in the log files as npeaks and as nPixels.
5. Report the total counts (intensity) as peakTotal

Algorithm 2:

Same as algorithm 1, except that the criteria for a hit is now that the *intensity* is greater than hitfindernat, rather than the pixel count.

Algorithm 3:

Briefly, this is what happens:

1. Scan the buffer, module-by-module, searching for "blobs" of connected pixels which all meet the criteria of being above the threshold defined by the keyword hitfinderADC. A pixel can be "connected" to any of its eight nearest neighbors. If its "connected" neighbor is "connected" to another pixel, then all three are mutually "connected" to each other.
2. If a blob contains more than hitfinderMinPixCount connected pixels, and less than hitfinderMaxPixCount pixels, it is counted as a peak.
3. The center of mass and integrated intensity is calculated for the blob (this is the peak position and integrated intensity).
4. If there are more than hitfinderNpeaks peaks, and less than hitfinderNpeaksMax, then count this as a hit.

Some important keywords:
- hitfinderNAT
- hitfinderADC
- hitfinderMinPixCount
- hitfinderMaxPixCount
- hitfinderNPeaks
- hitfinderNPeaksMax
- hitfinderCheckMinGradient: Before considering a peak candidate, check that the intensity gradient is above this threshold. Here, the "gradient" is the mean square derivative of the "above/below" and "upper/lower" pairs of pixels connnected to the pixel of interest.
- hitfinderMinGradient: Threshold for the above keyword.
- hitfinderCheckPeakSeparation: After locating peaks, throw out all the peak pairs that are too close together.
- hitfinderMaxPeakSeparation: Threshold for the above keyword.

This algorithm also calculates a quantity called "peak density"?
What's being reported in the log files?

Algorithm 4:
Use time-of-flight data. Detail coming soon.

Algorithm 5:
Similar to 3. Details coming soon.

Algorithm 6:
Firstly, create a combined mask which indicates hot pixels, saturated pixels, bad pixels, pixels specifically masked at the peakfinding stage (keyword peakmask), and pixels outside the specified resolution range (keywords hitfinderLimitRes, hitfinderMinRes, hitfinderMaxRes). These pixels will never be considered for further analysis.

Each pixel in each detector panel (one panel at a time) will be inspected. Initially, we are seaching for a simple "trigger" to indicate the possibility of a peak, with more stringent tests to follow. Here's how it works:
1) If the pixel intensity (in raw ADC units) is below the threshold set by hitfinderADC, skip this pixel.
2) If any of the eight nearest neighbor pixels has a greater intensity, skip this pixel.

3) If the above tests pass, the signal-to-noise ratio (SNR) will be calculated as follows: the mean background intensity <I> and the standard deviation sig(I) are calculated from a concentric square annulus, with its radius specified by the keyword hitfinderLocalBGRadius. The thickness of the annulus is specified by the keyword hitfinderLocalBGThickness. (For example, if the radius is 1, and the thickness is 1, then only the nearest 8 pixels will be considered in this calculation.) The SNR for this pixel is equal to (I-<I>)/sig(I).
4) If the background-corrected intensity (I - <I>) is less than the threshold hitfinderADC, skip this pixel.
5) If the SNR value is below the value hitfinderMinSNR, skip this pixel.
6) If the above tests pass, a test for how many *connected* pixels also meet the above criteria will be performed. If the number of connected pixels falls within the (inclusive) range [hitfinderMinPixCount , hitfinderMaxPixCount], then this will be counted as a peak. Note that connected pixels are masked, and will not be considered for further analysis.
7) The centroid of the peak will be calculated (within the box of radius equal to hitfinderLocalBGRadius).
8) Once a peak is found, a test will be performed to check that there are not other peaks that are too close to this one. The limiting distance is set by the keyword hitfinderMaxPeakSeparation. If a closer peak is found, it will be eliminated if it has lower SNR than this peak, else the current peak will be eliminated. (Note that, currently, this does not guarantee that some closely-spaced peak pairs will not be found, but will eliminate most of them).
9) Once the last pixel has been analyzed, if the number of peaks found is in the (inclusive) range [hitfinderMinPeaks, hitfinderMaxPeaks], then this pattern will be considered a hit.

Algorithm 7:
Coming soon.

Algorithm 8:
This is now the most commonly used hit finding algorithm for serial crystallography experiments.

This algorithm uses a radial average to determine the SNR and intensity threshold. If there are shadows in your patterns, be sure to mask these regions during peak finding by using a PeakMask (see peakmask section above).

Some important keywords:
- MinADC
- MinSNR
- Npeaks
- NpeaksMax
- MinPixCount
- MaxPixCount
- LocalBgRadius
- MinPeakSeparation
- MinRes
- MaxRes
- FastScan

Hitfinder tuning

hitfinderAlgorithm (8)

See the keyword hitfinder.

hitfinderADC (100)

See the keyword hitfinder.

hitfinderNAT (100)

See the keyword hitfinder.

hitfinderNPeaks (50)

See the keyword hitfinder.

hitfinderNPeaksMax (100000)

See the keyword hitfinder.

hitfinderMinPixCount (3)

See the keyword hitfinder.

hitfinderMaxPixCount (20)

See the keyword hitfinder.

hitfinderLocalBGRadius (4)

See the keyword hitfinder.

hitfinderLocalBGThickness (1)

See the keyword hitfinder.

hitfinderLimitRes (0)

See the keyword hitfinder.

hitfinderMinRes (0)

See the keyword hitfinder.

hitfinderMaxRes (0)

See the keyword hitfinder.
For weak diffraction spots, it may help to restrict the search for spots to within the diffuse ring from water, LCP etc, to avoid picking up false peaks. However, note that limiting the peak finding resolution impacts on geometry refinement and both of CrystFEL's resolution prediction and integration radius refinement)

hitfinderUseTof (0)

Does choosing hitfinding algorithm 4 accomplish the same thing?

hitfinderTofMinSample (0)

?

hitfinderTofMaxSample (1000)

?

hitfinderTofThresh=1283604304

?

Specifying what gets saved in exported frame HDF5 files

saveCXI (1)

As of February 2015 the default file format changed from individual HDF5's to CXIDB format (www.cxidb.org). A description of the .cxi format for multi-event data collected on a CSPAD at LCLS can be found at the bottom of this document or on the CXIDB website. In summary – all hits and metadata are stored in a single HDF5. This format is compatible with the Cheetah GUI and the online data analysis software OnDA.

To generate many small HDF5's, set saveCXI to 0 (this is the original approach).

saveHits (0)

Save the hits to individual hdf5 files. Exactly what will be saved is determined by the keywords *saveRaw*, *saveAssembled*, *savePeakInfo*, *saveDetectorCorrectedOnly*, *saveDetectorRaw*, and possibly more...

saveNonAssembled (1)

Save detector data as hdf5 without geometry corrections, i.e. in the layout as read from the detector, without interpolation into a physically realistic image.

saveAssembled (1)

Save the data after it has been interpolated into a physically correct image (as would be seen on a sheet of film), based on the geometry file. Note that this will take up more space on disk, but provides an image that can be analysed/displayed as if the detector were one CCD. Also, note that geometry is updated sometimes, and you will need to re-run all of your hit finding if you intend to store the data only in assembled form (not recommended). The hdf5 field is /data/assembleddata, and has zeros where there is no data. If present, it will be symbolically linked to the field /data/data.

saveDetectorRaw (0)

Save detector data as hdf5 without photon corrections, i.e. no dark current subtraction and no other background subtraction or masks applied.

The hdf5 data field is /data/rawdata. Note that the word "raw" does not mean uncorrected (!) as one might think; it just means that it has not

saveDetectorCorrected (1)

Save detector data in the hdf5 files with dark current subtraction, gain calibration (if gaincal is specified), and bad pixel mask applied. This does not include further background subtraction, nor does it involve geometry in any way.

saveDetectorCorrected (1)

Recommended mode. Even if background subtraction is used for hit finding, back up to image with only detector corrections subtracted and save this instead. Useful for preserving the water ring, for example.

If set to non-zero value, save the data which has only the following operations done to it (in this order):

1) Subtract darkcal
2) Subtract common mode offsets
3) Apply gain correction
4) Multiply by bad pixel mask

If set to zero, then you get these additional corrections (in this order):

5) Subtract running (persistent) background
6) Subtract local background
7) Zero out hot pixels
8) Multiply by bad pixel mask (again)

If the keyword saveDetectorRaw is set, then none of the above corrections will be applied (therefore, this keyword has no effect).

saveDetectorAndPhotonCorrected (0)

Same as saveDetectorCorrected = 1 followed by background subtraction.
If set to 1, save the data which has the following operations done to it (in this order):

1) Subtract darkcal
2) Subtract common mode offsets
3) Apply gain correction
4) Multiply by bad pixel mask
5) Subtract running (persistent) background
6) Subtract local background
7) Zero out hot pixels
8) Multiply by bad pixel mask (again)

If the keyword saveDetectorRaw is set, then no corrections will be applied (i.e. this keyword will have no effect).

saveDetectorRaw (0)

Image will be saved exactly as represented in the XTC data stream (even before dark current subtraction), regardless of what detector corrections and photon background subtraction is used for hit finding. This option is mainly of interest to

detector groups who want to look at data in as raw a form as possible, or for low-level diagnostics on common mode, electronic noise, etc.
This keyword trumps the keywords saveDetectorCorrected and saveDetectorAndPhotonCorrected.

hdf5dump (0)

Write every nth frame to an hdf5 file, regardless of whether it was found to be a hit.

dataSaveFormat (INT16)

Accepted formats = INT16, INT32, float.

saveEpicsPVfloat (no default)

What EPICS process variables you want saved to the .cxi format output, up to 99 entries. Not available if saveCXI=0
E.g. saveEpicsPVfloat=CXI:TTSPEC:FLTPOS
saveEpicsPVfloat=CXI:LAS:MMN:06.RBV

Creation of calibration files

generateDarkcal (0)

Create a darkcal from a given run (which should contain dark data -- i.e. data without the X-ray beam on). Takes the average of all patterns, and output a "darkcal" hdf5 file named rXXXX-darkcal.h5 in the end. Essentially, this option tricks cheetah into thinking every frame is a "hit". The darkcal is the average, not the sum, unlike the usual "powder" patterns. If you set generatedarkcal=1, the following keywords will be modified so everything works as expected:

```
cmModule = 0;
cmSubtractUnbondedPixels = 0;
subtractBg = 0;
useDarkcalSubtraction = 0;
useGaincal=0;
useAutoHotpixel = 0;
useSubtractPersistentBackground = 0;
hitfinder = 0;
savehits = 0;
hdf5dump = 0;
saveRaw = 0;
saveDetectorRaw = 1;
powderSumHits = 0;
powderSumBlanks = 0;
powderthresh = -30000;
startFrames = 0;
saveDetectorCorrectedOnly = 1;
```

generateGaincal (0)

Automatically create a gain map file from flat field data. Works, but the output likely needs tweaking by hand in IDL/Matlab. All patterns will be summed to form an average, which is then divided by the median value of the image. (The median value is therefore gain = 1.) The gainmap will be saved as "rXXXX-gaincal.h5". At the moment, the gainmap is set to zero where it is outside of the bounds 0.1 and 10. When setting generategaincal=1 the following keywords will be modified so everything works as expected:

```
cmModule = 0;
cmSubtractUnbondedPixels = 0;
subtractBg = 0;
useDarkcalSubtraction = 1;
useAutoHotpixel = 0;
useSubtractPersistentBackground = 0;
useGaincal=0;
hitfinder = 0;
savehits = 0;
```

hdf5dump = 0;
saveRaw = 0;
saveDetectorRaw = 1;
powderSumHits = 0;
powderSumBlanks = 0;
powderthresh = -30000;
startFrames = 0;
saveDetectorCorrectedOnly = 1;

Image summation (powder patterns)

powderSumHits (1)

Record and save the summed (not averaged) intensities from frames determined to be hits. Will be saved as the file named rXXXX-detector0-class1-sum.h5 where XXXX is the run number (e.g. 0013). The hdf5 data field is /data/data.

powderSumBlanks (1)

Record and save the summed (not averaged) intensities from frames determined to be non-hits. Will be saved as the file named powderSumBlanks.h5. The HDF5 data field is /data/data.

powderThresh (-20000)

Apply this intensity threshold before powder summation. Setting to ~500 typically captures only peaks; setting to 0 sums only positive values; setting to -20,000 typically sums everything.

savePowderDetectorRaw (0)

Apply this intensity threshold before powder summation. Setting to ~500 typically captures only peaks; setting to 0 sums

savePowderDetectorCorrected (1)

Same steps as "saveDetectorCorrected" applied to summed intensities. HDF5 dataset is /data/non_assembled_detector_corrected, linked to from /data/data and /data/correcteddata. A dataset of the detector corrected pixels' sigma is also recorded, in /data/non_assembled_detector_corrected_sigma

savePowderDetectorAndPhotonCorrected (1)

Same as savePowderDetectorCorrected = 1, followed by background subtraction. HDF5 dataset is /data/non_assembled_detector_and_photon_corrected. A dataset of the detector and photon corrected pixels' sigma is also recorded, in /data/non_assembled_detector_and_photon_corrected_sigma

savePowderNonAssembled (1)

Save the summed detector data as hdf5 without photon or geometry corrections, i.e. in in data layout as read from the detector, without interpolation into a physically realistic image.

savePowderAssembled (0)

Save the summed (not averaged) intensities which has been interpolated into a physically correct image (as would be seen on a sheet of film), based on the geometry file.
The hdf5 field is /data/assembleddata, and has zeros where there is no data. If present, it will be symbolically linked to the field /data/data.

usePolarizationCorrection (0)

Radial intensity profiles (SAXS/WAXS profiles)

saveRadialStacks (0)

Save hdf5 files containing radial profiles, image-by-image. Masked pixels will not be integrated. More details here some day.

radialStackSize (10000)

How many radial profiles in each hdf5 "radial stack" file.

Energy spectrum

espectrum (0)

Toggles the creation of a 2D array from the readout of the energy spectrum CCD.

espectrum1D (1)

Toggles the creation of a 1D line output of the beam energy profile, calculated by integration of the energy spectrum CCD along the line of the stripes in the spectral profile.

espectrumDarkSubtract (1)

Toggles the removal of a mean background from the CCD before calculation of the 1D integrated spectrum. A background is calculated when the *generateDarkcal* flag is set to 1

espectrumTiltAng (0)

Input which specifies the tilt of the spectral stripes with respect to the horizontal axis of the CCD. The angle is defined so that a positive value relates to a clockwise rotation of the spectral stripes from the horizontal. This can be calculated from the first run of the experiment from an image of the spectrum CCD camera in the event hdf5 file in the field "/data/espectrumCCD". This value is likely to change if the spectrum CCD is moved, i.e. in the event of an energy change or X-ray beam re-focusing.

espectrumLength (1080)

Number of pixels along the energy axis of the spectrum CCD camera. Defaults to the value for the long axis of an Opal2K CCD of 1080 pixels

espectrumWidth (900)

Number of pixels along the beam profile axis of the spectrum CCD camera. Defaults to the value for the short axis of an opal2K CCD of 900 pixels

espectrumSpreadeV (40)

Value of the energy spread seen by the energy spectrum CCD camera in eV. Defaults to 40

espectrumDarkfile ()

Path to the energy spectrum dark calibration file created from a dark calibration run. The filename will have the format rXXXX-energySpectrum-darkcal.h5. If the *espectrumDarkSubtract* keyword is set to 1 and no file path is given or the specified file does not exist the program will terminate.

espectrumScalefile (0s)

Path to a file which is used in an attempt to calibrate the spectral profile to absolute energy values. At the end of each run a spectrum integrated over the whole run is calculated, the maximum in this run-integrated spectrum is assigned the average beam energy value (as reported by *global->meanPhotonEnergyeV*). A scale is generated based on the known energy spread of the spectrum CCD and saved in the rXXXX-integratedEnergySpectrum.h5 file. The scale can be assigned to individual events by either re-running cheetah with the scale file belonging to that run or the scale from a previous run can be applied. If no file is given or the file specified does not exist the individual event spectra are output with zeros for the energy scale (the program does not terminate)

Time of flight spectrometer (Acqiris)

tofName (CxiSc1)

Is this relevant to our pump laser diode trace?

tofName (CxiSc1)

Name of Acqiris device in XTC data stream

tofChannel (1)

Acqiris channel number

Multithread tuning and speed optimization

nThreads (16)

Run this many worker threads in parallel (one worker thread per LCLS event). Set to 1x-2x the number of cores on the machine.

- 72 or 144 on cfelsgi (72 physical cores)
- 16 is more than adequate on most servers (eg: compute farm at SLAC uses 12 or 16 core machines)

Speed may saturate before all threads are busy if data transfer makes cheetah I/O limited (check with ipSpeedTest), or if competing access to shared variables results in mutex locks (happens when too many threads write to powder patterns or running background buffer at once).

useHelperThreads (0)

This currently does nothing. It was intended for computing backgrounds asynchronously with processing data frames.

saveInterval (1000)

Periodically save running sums and update the log file at this interval.

Data processing flow: skipping XTC frames

startAtFrame (0)

Skip all frames in the xtc file prior to this one (no processing done).

stopAtFrame (0)

Skip all frames in the xtc file after this one. Setting to 0 means to ignore this setting.

startFrames (0)

Number of frames at the start of processing used for background estimates, etc, before starting hit finding etc.

ioSpeedTest (0)

Run through events in xtc file, reading in all data, but do no data processing (don't spawn worker threads). Useful for checking raw I/O speed to determine whether the process is I/O bound or CPU/mutex bound.

Time resolved work

sortPumpLaserOn (0)

Run through events in xtc file, reading in all data, but do no data processing (don't spawn worker threads). Useful for

useTimeTool (0)

pumpLaserScheme (evr41)

What scheme of sorting by pump laser (probe) to use for separating time-resolved data. Options: evr41, LD57,
More schemes will be added in the future. This string is used to toggle the information fed into laserOn and the way in which patterns are sorted. Places that need changing if this ever changes are: (a) what is in laserOn, (b) setting the number of powder pattern, (c) setting the powder class.

saveByPowderClass (1)

• laserOn can be more than on or off. Let this be 0,1 or 2 depending on which laser is on. This scales in many ways e.g.: it can be a simple number of a binary mask if multiple lasers are on at once..
• Introduce a pumpLaserSignal field which can take the strings “evr41” or “Schmidt2014" or anything else in the future.

Misc

debuglevel (2)

Sets verbosity level of the output (how much diagnostic junk is printed to screen)

useFEEspectrum (0)

Used for diagnostic purposes

Cheetah output files

Along with diffraction hits, virtual powder patterns and statistics, all configuration files necessary to reproduce this hitfinding result are copied into each hdf5 directory.

rXXXX-detectorX-class0-sum.h5:

The summation of frames considered non-hits. (blanks, mostly), i.e. virtual powder pattern from rejected frames. Unless hdf5dump=1, frames contributing to this summation are not saved. This is useful to see if you are missing a lot of hits. Try viewing hdf5 datasets ending in “corrected_sigma” as peaks show up with much higher contrast than in the sum.

rXXXX-detectorX-class1-sum.h5:

The summation of hits, i.e. virtual powder pattern from hits.

data1 (data2…) directories

If saveCXI=0, individual HDF5’s are saved in data directories of up to 1000 files each. HDF5 filename:
LCLS_year_monthday_rXXXX_hhmmss_tttt.h5 (if saveCXI=0)

If saveCXI=1 (default), all hits and corresponding metadata are saved in CXI format in a single HDF5.
https://github.com/FilipeMaia/CXI/raw/master/cxi_file_format.pdf

There is a symbolic link called /data/data, but what it links to depends on parameters in the ini file.

frames.txt:

Contains a list of all detector readout events (hits and non-hits) with various attributes.

eventData->_____	*meaning:*
eventName	HDF5 filename: LCLS_year_monthday_rXXXX_hhmmss_tttt.h5 (if saveCXI=0)
filename	“---“ if non hit; “data*/filename “ if hit.
stackSlice	
xtcFrameNumber	

hit	
powderClass	0 = non hits; 1 = hits
hitScore	
photonEnergyeV	
wavelength	Å
gmd1	gas monitor detector 1 (for incident flux measurement)
gmd2	gas monitor detector 2 (for incident flux measurement)
detector[0].detector	sum of “detectorZpvname” and “cameraLengthOffset” in .ini file
energySpectrumExist	was the spectrometer in place and recorded to datastream?
nPeaks	Number of peaks found
peakNpix	Total number of pixels that contribute to peaks in pattern
peakTotal	Total intensity of all peak pixels
peakResolution	in pixels
peakDensity	
pumpLaserCode	process variable where laser trigger is recorded (evr41, evr183, LD57)
pumpLaserDelay	
pumpLaserOn	trigger for pump laser experiments

cleaned.txt

Filename	info->eventname	
frameNumber	threadNum	
npeaks	info->nPeaks	
nPixels	info->peakNpix	
totalIntensity	info->peakTotal	
peakResolution	info->peakResolution	pixels
peakResolutionA	info->peakResolutionA	Å
peakDensity	info->peakDensity	

rXXXX-class0-log.txt and rXXXX-class1-log.txt

Similar to frames.txt and cleaned.txt but class0 = non hits, and class1 = hits. (so cleaned.txt and class1-log.txt will contain the same files). The columns are:
eventData->eventname, eventData->filename, eventData->stackSlice, eventData->xtcFrameNumber, eventData->hitScore, eventData->photonEnergyeV, eventData->wavelengthA, eventData->detector[0].detectorZ, eventData->gmd1, eventData->gmd2, eventData->energySpectrumExist, eventData->nPeaks, eventData->peakNpix, eventData->peakTotal, eventData->peakResolution, eventData->peakDensity, eventData->pumpLaserCode, eventData->pumpLaserDelay

darkcal.h5

Copy of the dark current measurement specified in original.ini as darkcal.

geometry.h5

Copy of pixel map specified in the .ini file under geometry.

Log.txt

Progress of hit finding, updated at the rate set by keyword saveInterval. If hit finding has finished, a summary is appended, including total frames processed, number of hits, hit rate, average photon energy and its sigma.

bsub.log

Log from batch job submission. Look here for errors when hit finding doesn’t work. It will report misuse of keywords and other problems.

original.ini

Copy of your original ini file (renamed).

cheetah.ini
Same as original.ini but with commented lines removed.

cheetah.out
Full list of parameters used by Cheetah for this hit finding. You can see here if any of your keyword values from cheetah.ini were overwritten automatically due to clashes.

Peakmask.h5
Copy of mask used while peak finding. See keyword peakmask.

Peaks.txt
Space separated column file with peak information. One line per peak.
frameNumber, eventName, photonEnergyEv, wavelengthA, GMD, peak_index, peak_x_raw, peak_y_raw, peak_r_assembled, peak_q, peak_resA, nPixels, totalIntensity, maxIntensity, sigmaBG, SNR
Where GMD is a gas monitoring detector (proportional to incident flux), and peak_index: is

Psana.cfg
Copy of psana.cfg from your cheetah/process directory: the configuration file for psana, the LCLS analysis framework.

Status.txt
Status of hit finding.

Xtcfiles.txt
List of xtc files from this run. If you started your hit finding before all the xtc files finished writing to the offline storage, this list may be incomplete. You will not see an error in the output from Cheetah if everything runs correctly, but rerunning it at a later date will show more frames processed. In the directory where the raw data are saved, /reg/d/psdm/cxi/cxiXXXXX/xtc (XXXXX = the experimental ID with the last 2 digits corresponding to the year of the experiment), while the xtc files are being written, their names are appended with .inprogress and Cheetah deliberately excludes them until the extension is solely .xtc.

Cheetah output HDF5 structure

This table attempts to show the structure of the individual event output HDF5 file datafields

LCLS_2013_Feb12_r0194_053144_17343.h5		
/LCLS	/data	/processing
/detector0-EncoderValue	/data	/energySpectrum-tilt
/detector0-Position	/energySpectrum1D	/hitfinder
/detector1-EncoderValue	/energySpectrumCCD	/peakinfo
/detector1-Position	/energySpectrumScale	/peakinfo-assembled
/ebeamCharge	/radialAverage0	/peakinfo-raw
/ebeamL3Energy	/radialAverage1	/pixelmasks
/ebeamLTUAngX	/radialAverageCounter0	
/ebeamLTUAngY	/radialAverageCounter1	
/ebeamLTUPosX	/rawdata	
/ebeamLTUPosY	/rawdata0	
/ebeamPkCurrBC2	/rawdata1	
/eventTimeString		
/evr41		
/f_11_ENRC		
/f_12_ENRC		
/f_21_ENRC		
/f_22_ENRC		
/fiducial		

	/machineTime				
	/phaseCavityCharge1				
	/phaseCavityCharge2				
	/phaseCavityTime1				
	/phaseCavityTime2				
	/photon_energy_eV				
	/photon_wavelength_A				

This table shows the structure of the virtual powder patterns, rXXXX-detectorX-classX-sum.h5

R0016-detector0-class1-sum.h5			
/data (group)		/data (link)	
	/nframes		/correcteddata --> /data/non_assembled_detector_corrected
	/non_assembled_detector_and_photon_corrected		/data --> /data/non_assembled_detector_corrected
	/non_assembled_detector_and_photon_corrected_sigma		
	/non_assembled_detector_corrected		
	/non_assembled_detector_corrected_sigma		
	/peakpowder		
	/radial_average_detector_and_photon_corrected		
	/radial_average_detector_and_photon_corrected_sigma		
	/radial_average_detector_corrected		
	/radial_average_detector_corrected_sigma		

Need to check if other 2D datasets are created when other savePowder options are turned on.

CXI format: https://github.com/FilipeMaia/CXI/raw/master/cxi_file_format.pdf
If saveCXI=1 (default), all hits and corresponding metadata are saved in CXI format in a single HDF5.

3.6 CXI stack of images from a modular detector

This file exemplifies the use of Modular Pixel Detector together with Scans to store a large dataset of 30,000 images.

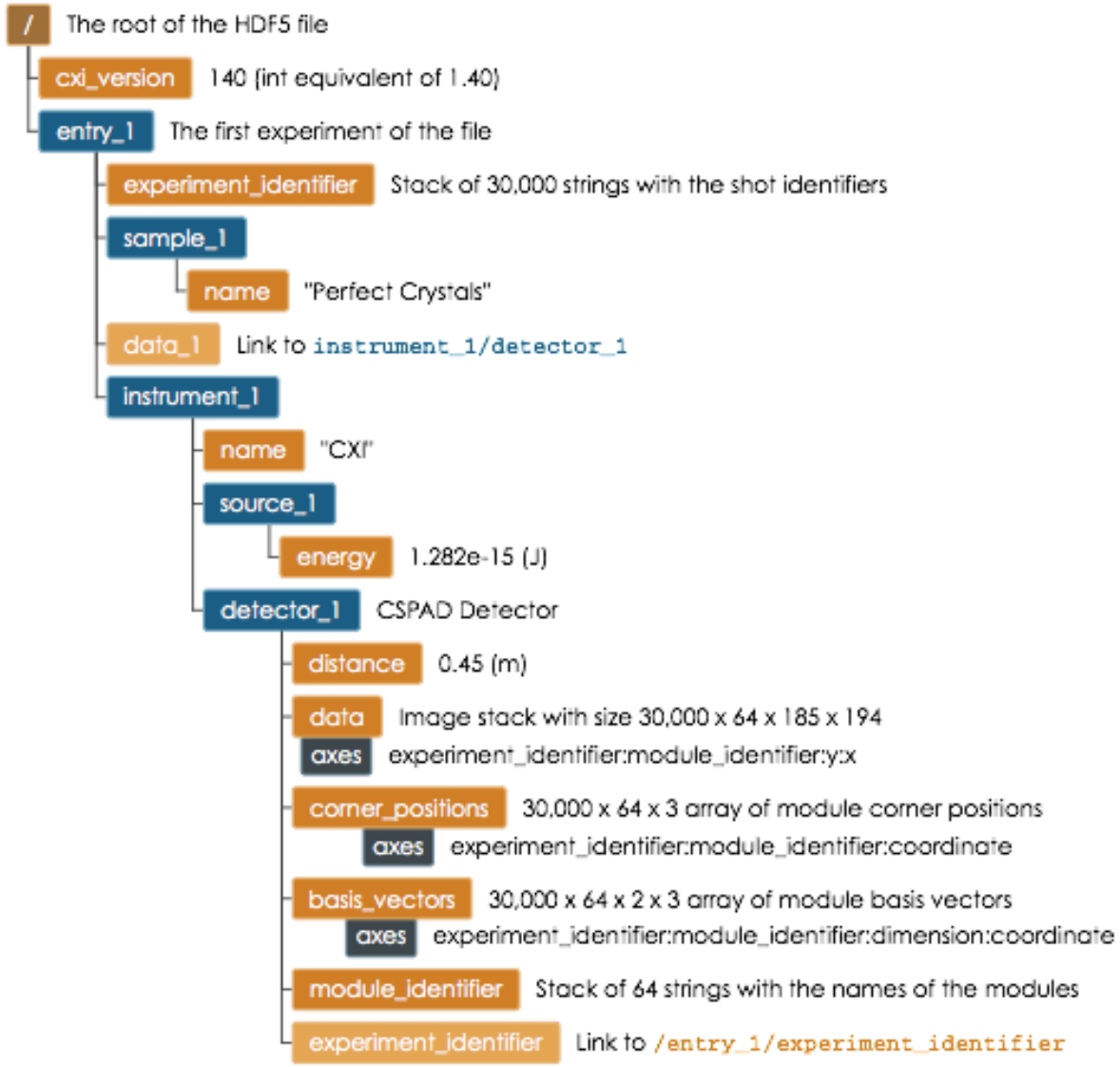

Figure 6: Diagram of a CXI file with a stack of 30,000 images from a pixel detector made of 64 modules, such as the LCLS CSPAD.

Cheetah GUI

For LCLS users:
The Cheetah website has detained instructions for setting up your experimental directories to use the centrally installed Cheetah: http://www.desy.de/~barty/cheetah/Cheetah/Configuration.html

Scripts

Scripts to help with miscellaneous tasks while hit finding and doing preliminary analysis can be downloaded from https://www.bioxfel.org/resources/scripts

peakogram
Log from batch job submission. Look here for errors when hit finding doesn't work. It will report misuse of keywords and

hits
A script to quickly calculate hits and hitrates from SFX experiments

stream2stats
A script to quickly calculate data quality metrics from SFX experiments

visibly_bad_mask.py
A script to generate masks (bad pixel or peak) manually. Useful for shadows, rings from substrate/sample holder. The resultant binary (0/1) hdf5 mask needs to be multiplied by the bad pixel mask Cheetah created from the darkcal.h5. You can use h5mult to do that.

h5mult
A script to multiply hdf5 files, used for combining masks.

Running Cheetah without a GUI or the *process* script

Cheetah can be run on its own, but is most easily used in conjunction with a script such as crystfinder (or process) – which does a lot of useful housekeeping. Here's a quick example of a workflow without using a script:

0) Check that the cheetah program is in your path:

```
> cheetah -h
```

If all goes well, you'll see a brief list of Cheetah options. Currently, this is not very helpful, hence the need for this section…

1) Create a new directory, which is where you would like the output of Cheetah will be written.

```
> mkdir r0090
> cd r0090
```

2) Let's say you would like to analyze run 90, in which case there are a few XTC data files in the directory /reg/xtcdir with names like "e158-r0090-s01-c00.xtc". Firstly, you should write the paths to those files into a text file, which we will then provide to Cheetah as an input:

```
> find /reg/xtcdir -name '*r0090*.xtc' > xtcfiles.txt
```

3) In order to run Cheetah, you'll need a configuration file, which contains the instructions and tunable parameters needed to carry out this processing job (more on that later…). Cheetah automatically searches for a configuration file called "cheetah.ini" in the current directory. You'll probably have a bunch of ".ini" files in some directory (say, "/reg/myinis"), so you should copy the appropriate .ini file for this job into the current directory

```
> cp /reg/myinis/test.ini cheetah.ini
```

4) Now you are ready to run cheetah:

```
> cheetah -l xtcfiles.txt
```

Once Cheetah has completed the job, you will find several log files, along with hdf5-formatted data files containing individual "hits", summed "powder" patterns, SAXS profiles, etc. (more on Cheetah output later).

Planned additions to this document

- How to use Cheetah for single particle hit finding, and SAXS / WAXS experiments.
  - Psana.cfg
  - Psana-spi.cfg: do_gain (yes)
  - Psana-spi.cfg: do_peds (yes)
  - Psana-spi.cfg: do_cmod (yes)

- How to implement Cheetah on non CXI detectors, e.g. Pilatus.
  - From http://www.desy.de/~barty/cheetah/Cheetah/Cheetah_at_home.html
  - If your data comes from somewhere other than LCLS, Cheetah can be called from code able to read any other file format: it is simply a matter of passing the frame data to Cheetah for processing. Once again, see the developerpages for more details.
  - Basic hit finding from Pilatus detectors (cbf output) can be performed with cbf2hdf5, a cbf to HDF5 converter that creates HDF5's ready for analyzing with CrystFEL.
  - Download *cbf2hdf5* from http://www.desy.de/~twhite/crystfel/programs.html