psana - Module Catalog

Unknown macro: 'html'

Unknown macro: 'html'

About

This page provides a list of existing modules for psana framework. Only the modules that are included in the standard analysis releases appear on this page.

Note that most modules have configuration parameters which are documented in tables. For example:

parameter	default value	description
`separator`	"="	separator string
`repeat`	80	number of repeats of separator string on a separator line

This page provides examples for selected modules in Psana Module Examples.

Source Code Access

You can find the source code at SLAC (e.g. for module CSPadPixCoords::PixCoordsTest) in these two directories:

```
/reg/g/psdm/sw/releases/ana-current/CSPadPixCoords/src/PixCoordsTest.cpp (.cpp for c++ source code, or .py for
python source code)
/reg/g/psdm/sw/releases/ana-current/CSPadPixCoords/include/PixCoordsTest.h (c++ include file)
```

Package psana

Psana package include several simple modules which do generic tasks that do not need knowledge of the event data types.

Module psana.EventKeys

This module dumps the list of the event keys in the event or configuration store. Event key is a triplet of data type, data source address, and string key.

Example of the output produced by this module:

```
Event keys:
  EventKey(type=Psana::EvrData::DataV3, src=DetInfo(NoDetector.0:Evr.0))
  EventKey(type=Psana::Camera::FrameV1, src=DetInfo(CxiDg1.0:Tm6740.0))
  EventKey(type=Psana::Ipimb::DataV1, src=DetInfo(CxiDg1.0:Ipimb.0))
...
```

This module have two optional parameters to control output:

parameter	default value	description
`print_cfg_in_evt`	false	Flag: true - print in event() info about env.configStore().keys()
`print_clb_in_evt`	false	Flag: true - print in event() info about env.calibStore().keys()

This module is useful to display the content of the events without dumping all the data. Example command which uses this module is:

```
% psana -m EventKeys <input-files>
```

Note that one can specify `EventKeys` instead of `psana.EventKeys` as `psana` package name is optional.

Module psana.PrintEventId

This module prints the content of the Event ID object on every event.

Example of the output produced by this module:

```
[info:psana.PrintEventId] event ID: XtcEventId(run=100, time=2010-12-12 11:09:36.300506429-08)
[info:psana.PrintEventId] event ID: XtcEventId(run=100, time=2010-12-12 11:09:36.317163082-08)
```

Module psana.PrintSeparator

This module prints separator line on every event. Can be used to indicate event boundaries in job's log.

parameter	default value	description
`separator`	"="	separator string
`repeat`	80	number of repeats of separator string on a separator line

Example of the output produced by this module with the default parameters (including output from PrintEventId module):

```
[info:/root/] ================================================================================
[info:psana.PrintEventId] event ID: XtcEventId(run=100, time=2010-12-12 11:09:36.300506429-08)
[info:/root/] ================================================================================
[info:psana.PrintEventId] event ID: XtcEventId(run=100, time=2010-12-12 11:09:36.317163082-08)
```

Package psana_examples

This package contains modules that are meant to be used as examples of accessing different data types or framework services. Can be used by developers as templates for new modules. To find the code:

- On the psana machines, from your release directory, one can do
  cd $SIT_REPOS/psana_examples
- Visit the link https://pswww.slac.stanford.edu/swdoc/releases/ana-current/psana-modules-doxy/html/files.html to find the files. For instance https://pswww.slac.stanford.edu/swdoc/releases/ana-current/psana-modules-doxy/html/DumpControl_8cpp-source.html shows one how to dump control data.

Module psana_examples.DumpAcqiris

Extracts and dumps the content of Acqiris configuration (`Psana::Acqiris::ConfigV1`) and event data (`Psana::Acqiris::DataDescV1`) objects.

parameter	default value	description
`source`	"DetInfo(:Acqiris)"	data source address

Module psana_examples.DumpAcqTdc

Extracts and dumps the content of Acqiris TDC configuration (`Psana::Acqiris::TdcConfigV1`) and event data (`Psana::Acqiris::TdcDataV1`) objects.

parameter	default value	description
`source`	"DetInfo(:AcqTDC.0)"	data source address

Module psana_examples.DumpBld

Extracts and dumps the content of beamline data objects (`Psana::Bld::BldDataEBeamV0`, `Psana::Bld::BldDataEBeam`, `Psana::Bld::BldDataPhaseCavity`, `Psana::Bld::BldDataFEEGasDetEnergy`, and `Psana::Bld::BldDataIpimb`).

parameter	default value	description
`eBeamSource`	"BldInfo(EBeam)"	data source address for `Psana::Bld::BldDataEBeam`
`phaseCavSource`	"BldInfo(PhaseCavity)"	data source address for `Psana::Bld::BldDataPhaseCavity`
`feeSource`	"BldInfo(FEEGasDetEnergy)"	data source address for `Psana::Bld::BldDataFEEGasDetEnergy`
`ipimbSource`	"BldInfo(NH2-SB1-IPM-01)"	data source address for `Psana::Bld::BldDataIpimb`

Module psana_examples.DumpCamera

Extracts and dumps the content of camera configuration (`Psana::Camera::FrameFexConfigV1`) and event data (`Psana::Camera::FrameV1` and `Psana::Camera::TwoDGaussianV1`) objects.

parameter	default value	description
`source`	"DetInfo(:Opal1000)"	data source address

Module psana_examples.DumpControl

Extracts and dumps the content of scan control configuration (`Psana::ControlData::ConfigV1`) object.

parameter	default value	description
`source`	"ProcInfo()"	data source address

Module psana_examples.DumpCsPad

Extracts and dumps the content of CsPad configuration (`Psana::CsPad::ConfigV*`) and event data (`Psana::CsPad::DataV*`) objects.

parameter	default value	description
`source`	"DetInfo(:Cspad)"	data source address

Module psana_examples.DumpEncoder

Extracts and dumps the content of encoder configuration (`Psana::Encoder::ConfigV1`) and event data (`Psana::Encoder::DataV*`) objects.

parameter	default value	description

source	"DetInfo(:Encoder)"	data source address

Module psana_examples.DumpEpics

Extracts and dumps the list of EPICS PVs and their values and status information on every event.

Module psana_examples.DumpEvr

Extracts and dumps the content of Evr configuration (`Psana::EvrData::ConfigV*` and `Psana::EvrData::IOConfigV1`) and event data (`Psana::EvrData::DataV*`) objects.

parameter	default value	description
source	"DetInfo(:Evr)"	data source address

Module psana_examples.DumpFccd

Extracts and dumps the content of FCCD configuration (`Psana::FCCD::FccdConfigV*`) objects. To dump event data objects use psana_examples.DumpCamera module.

parameter	default value	description
source	"DetInfo(:Fccd)"	data source address

Module psana_examples.DumpIpimb

Extracts and dumps the content of IPIMB configuration (`Psana::Ipimb::ConfigV1`) and event data (`Psana::Ipimb::DataV1`) objects.

parameter	default value	description
source	"DetInfo(:Ipimb)"	data source address

Module psana_examples.DumpLusi

Extracts and dumps the content of LUSI configuration (`Psana::Lusi::DiodeFexConfigV1`, `Psana::Lusi::IpmFexConfigV1`, and `Psana::Lusi::PimImageConfigV1`) and event data (`Psana::Lusi::DiodeFexV1` and `Psana::Lusi::IpmFexV1`) objects.

parameter	default value	description
tmSource	"DetInfo(:Tm6740)"	data source address for `Psana::Lusi::PimImageConfigV1`
ipimbSource	"DetInfo(:Ipimb)"	data source address for other data

Module psana_examples.DumpOpal1k

Extracts and dumps the content of Opal1000 configuration (`Psana::Opal1k::ConfigV1`) object. To dump event data objects use psana_examples.DumpCamera module.

parameter	default value	description
source	"DetInfo(:Opal1000)"	data source address

Module psana_examples.DumpPnccd

Extracts and dumps the content of pnCCD configuration (`Psana::PNCCD::ConfigV*`) and event data (`Psana::PNCCD::FrameV1`) objects.

parameter	default value	description
source	"DetInfo(:pnCCD)"	data source address

Module psana_examples.DumpPrinceton

Extracts and dumps the content of Princeton configuration (`Psana::Princeton::ConfigV1`) and event data (`Psana::Princeton::FrameV1`) objects.

parameter	default value	description
`source`	"DetInfo(:Princeton)"	data source address

Module psana_examples.DumpPulnix

Extracts and dumps the content of Pulnix configuration (`Psana::Pulnix::TM6740ConfigV*`) objects.

parameter	default value	description
`source`	"DetInfo(:Tm6740)"	data source address

Module psana_examples.EBeamHist

This module is an example of histogramming service usage. It extracts beam line data and fills couple of histograms with the beam parameters.

parameter	default value	description
`eBeamSource`	"BldInfo(EBeam)"	data source address

Package cspad_mod

Package cspad_mod contains common modules useful for analysis of data from CsPad detector.

Module cspad_mod.CsPadPedestals

This module is supposed to run on dark cspad2x2 frame data. It calculates average and standard deviation values for each pixel and writes these values to output files in text format.

parameter	default value	description
`source`	"DetInfo(:Cspad)"	name of the data source; default value is adequate if there is only one CsPad device in setup, if there is more than one device then more specific value should be provided
`output`	"cspad-pedestals.dat"	name of the output file for average values, if empty file will not be written
`noise`	"cspad-noise.dat"	name of the output file for standard deviation values, if empty file will not be written

To use this module with default parameters one could use this command:

```
psana -m cspad_mod.CsPadPedestals input-files.xtc
```

which will produce files `cspad-pedestals.dat` and `cspad-noise.dat` in the current directory (if input file contains CsPad data).

To change module parameters one could create file with name `psana.cfg` in the current directory with a contents like this:

```
[psana]
modules = cspad_mod.CsPadPedestals

[cspad_mod.CsPadPedestals]
source = DetInfo(CxiDs1.0:Cspad.0)
output = pedestals.dat
noise =
```

and then run psana:

```
psana input-files.xtc
```

This should produce file `pedestals.dat` in the current directory. As the source address is more specific this configuration file can be used even if input data contain more than one CsPad device.

Module cspad_mod.CsPad2x2Pedestals

This module is supposed to run on cspad2x2 dark frame data. It calculates average and standard deviation values for each pixel and writes these values to output files in text format.

parameter	default value	description
`source`	"DetInfo(:Cspad2x2)"	name of the data source; default value is adequate if there is only one CsPad2x2 device in setup, if there is more than one device then more specific value should be provided
`output`	"cspad2x2-pedestals.dat"	name of the output file for average values, if empty file will not be written
`noise`	"cspad2x2-noise.dat"	name of the output file for standard deviation values, if empty file will not be written

To use this module with default parameters one could use this command:

```
psana -m cspad_mod.CsPad2x2Pedestals input-files.xtc
```

which will produce files `cspad2x2-pedestals.dat` and `cspad2x2-noise.dat` in the current directory (if input file contains CsPad2x2 data).

To change module parameters one could create file with name `psana.cfg` in the current directory with a contents like this:

```
[psana]
modules = cspad_mod.CsPad2x2Pedestals

[cspad_mod.CsPad2x2Pedestals]
source = DetInfo(CxiSc1.0:Cspad2x2.0)
output = pedestals.dat
noise =
```

and then run psana:

```
psana input-files.xtc
```

This should produce file `pedestals.dat` in the current directory. As the source address is more specific this configuration file can be used even if input data contain more than one CsPad2x2 device.

Module cspad_mod.CsPadCalib

This module performs standard CsPad calibration procedures: per-pixel pedestal subtraction and common mode correction. The algorithms used in calibrations are identical to translator algorithms, the result of calibration should be the same as the data stored in HDF5 files.

The module uses three input calibration files which should be located in the standard calibration directory of the corresponding experiment:

- `pedestals` – per-pixel pedestal data (produced by the CsPadPedestals module or standalone applications)
- `pixel_status` – hot/dead status for every pixel, used by common mode algorithm
- `common_mode` – file with parameters controlling common mode calculation
- `pixel_gain` – per-pixel gain data, inverse pixel gain (1/gain) as floating number

If any of the files is missing then corresponding algorithm is not executed, if `pixel_status` file is missing then all pixels are assumed to have working status. Algorithms inside module are executed following this order:

- common mode is calculated for every segment; common mode uses pedestal and pixel status values
- if common mode is calculated successfully it is subtracted from every pixel value in a segment
- if pedestals are defined pedestal values are subtracted from every pixel value
- if pixel gains are defined then each pixel value is multiplied by corresponding gain factor
- resulting pixel value is rounded to nearest integer and stored in the output image

The calibration is performed on every CsPad image (full or 2x2) found in the event and works even if there is more than one CsPad data object. The original objects are preserved in event and result of the calibration is stored in event with different string key ("calibrated" by default, can be changed with module parameter).

parameter	default value	description
`inputKey`	""	string key used to locate uncalibrated data objects in event
`outputKey`	"calibrated"	string key used to store calibrated data objects in event
`doPedestals`	"yes"	can be set to "no" to explicitly disable pedestal subtraction algorithm
`doPixelStatus`	"yes"	can be set to "no" to explicitly disable reading of pixel status data, all pixels will be used for common mode
`doCommonMode`	"yes"	can be set to "no" to explicitly disable common mode algorithm
`doPixelGain`	"yes"	can be set to "no" to explicitly disable pixel gain correction algorithm

Module cspad_mod.CsPadFilter

This is a filter module which implements skipping for the events which have too low signal in CsPad. filtering algorithm can be controlled trough the module parameter and/or parameters in calibration file. The name of the calibration file is "filter" and it will be searched in the standard calibration directories of the experiment.

parameter	default value	description
`inputKey`	"	string key used to locate uncalibrated data objects in event
`source`	"DetInfo(:Cspad)"	data source address
`skipIfNoData`	"yes"	when set to "yes" if the module does not find CsPad data in an event then it skips event
`mode`	-1	filtering mode, see below
`parameters`	all 0	list of parameters for filtering algorithm, up to 16 floating numbers

Parameter `mode` defines where the filter get its parameters and how it works. If set to -1 (default) then it reads parameters and actual mode value from a calibration text file which should contain one integer number for mode and up to 16 floating point parameters. If parameter is set to 0 or if the value read from file is 0 then no filtering is done, all events are passed through. If mode is set to 1 (from parameter or file) then filter calculates the number of pixels above certain threshold. Filter expects two values in parameter array: threshold for pixel value and minimum number of pixels above threshold. If second parameter is negative then it's assumed to be a percentage of the full number of pixels.

Filter works on individual image arrays (quadrants). Module checks for CsPad::DataV1, CsPad::DataV2, and CsPad2x2::ElementV1 objects in that order. If any object is found then module runs filter on images in that object. If any one image passes the filter then module returns and event is passed to downstream modules, otherwise framework skips this event.

With the default parameters module works with non-calibrated data, to switch to calibrated data add the module cspad_mod.CsPadCalib before filter and change `inputKey` parameter to `calibrated`.

Package CSPadPixCoords

Package CSPadPixCoords calculates the 2x1 section, quad, and CSPad pixel coordinates and produces the image.
For complete reference select Doxygen documentation.

Module CSPadPixCoords::PixCoordsTest

This module demonstrates of how to use the PixCoords2x1, PixCoordsQuad, and PixCoordsCSPad classes
in order to pre-calculate pixel coordinates, taking into account the calibration parameters.
Relevant images are produced in combination of the pixel coordinates with event data and saved in text files.

parameter	default value	description
`calibDir`	"/reg/d/psdm/CXI/cxi35711/calib"	directory with calibration file
`typeGroupName`	"CsPad::CalibV1"	data type and group names
`source`	"CxiDs1.0:Cspad.0"	source of data
`runNumber`	32	run number for calibration file
`events`	32	number of events before stop a job

filter	false	on/off for potential selection filter

Module CSPadPixCoords::CSPadImageProducer

CSPadImageProducer works in psana framework. It does a few operation as follows:

- gets the pixel coordinates from PixCoords2x1, PixCoordsQuad, and PixCoordsCSPad classes,
- gets data from the event (from Psana::CsPad::DataV#, Psana::CsPad::ElementV# (where # stands for 1 or 2) data objects or from ndarray<const T,3> of shape [N][185][388])
- produces the ndarray<const double,2> or Image2D<double> object with CSPad image for each event,
- adds the image object in the event for processing in other modules.
  Time consumed to fill the CSPad image array (currently 1750x1750) is measured to be about 40 msec/event on psana0105.
- produces image size pixel map array where real/fake pixels are marked as 1/0, respectively. This array is saved in the env.calibStore() as ndarray<const int16_t ,2> for source. If the file name fname_pixmap is not empty the map will be saved in this file in text format.
- produces image size map of pixel indexes in the flatten [4][8][185][388] array where fake pixels are marked as -1. This array is saved in the env.calibStore() as ndarray<const int32_t ,2> for source. If the file name fname_pixnum is not empty the map will be saved in this file in text format.

parameter	default value	description
calibDir		directory with calibration files, by default it is set to .../<experiment>/calib
typeGroupName	"CsPad::CalibV1"	calibration type and group names
source	"CxiDs1.0:Cspad.0"	source of data
key	""	key for data processing stage
imgkey	"image"	output key for image saved in event
fname_pixmap	""	file name with pixel map of the image; real/fake pixels are marked as 1/0, respectively
fname_pixnum	""	file name with pixel indexes in the flatten [4][8][185][388] array; fake pixels are marked as -1
tiltIsApplied	true	on/off for tilt angle of 2x1-sections and quads.
print_bits	0	verbosity: • =0 - print nothing, • +1 - input parameters, • +2 - calibration parameters, • +4 - consumed time per event, • +8 - information about image saved in event, • +16 - type sizes and id names, • +32 - information about pixel maps saved in files, • +64 - print everything from PSCalib::CSPadCalibPars

Remarks:

- By default the empty key corresponds to raw data.

Module CSPadPixCoords::CSPadInterpolImageProducer

CSPadInterpolImageProducer works in psana framework. It does a few operation as follows:

- gets the pixel coordinates from PixCoords2x1, PixCoordsQuad, and PixCoordsCSPad classes,
- makes the arrays of neighbour addresses (quad, section, row, column) and weights as a function of the bin indexes (ix, iy) of the CSPad image,
- gets data from the event,
- produces the Image2D object with interpolated CSPad image for each event,
- adds the ndarray<const double,2> or CSPadPixCoords::Image2D<double> object in the event for processing in other modules.
  In this module we use 4-node bi-linear interpolation algorithm.
  Time consumed to fill the CSPad image array (currently 1750x1750) is measured to be about 200 msec/event on psana0106. We consider options for acceleration using GPU or multi-core processing.

Module configuration parameters are the same as for the CSPadPixCoords::CSPadImageProducer.

Module CSPadPixCoords::CSPad2x2ImageProducer

CSPad2x2ImageProducer works in psana framework. It does a few operation as follows:

- gets geometry alignment parameters center and tilt from /reg/d/psdm/<inst>/<experiment>/calib/... directory
- gets the pixel coordinates from PixCoords2x1 and PixCoordsCSPad2x2 classes,
- gets data from the event,

- produces the `Image2D<double>` or `ndarray<const double,2>` object with CSPad image for each event,
- adds the image object in the event for processing in other modules.
  The CSPad2x2 image array is currently shaped as (400,400).

parameter	default value	description
`calibDir`		path to the `calib` directory, if different from default
`typeGroupName`	"CsPad2x2::CalibV1"	calibration software version
`source`	"DetInfo(:Cspad2x2.1)"	source of data
`inkey`		key for data processing stage
`outimgkey`	"image"	output key for image saved in event
`outtype`	int16	type of output data from the list of supported (float, double, int, int16)
`tiltIsApplied`	true	on/off for tilt angle of 2x1-sections and quads - currently is not used
`print_bits`	0	verbosity: • =0 - print nothing • +1 - input parameters • +2 - calibration parameters • +4 - event ID • + 16 - type sizes and id names

Remarks:

- By default the empty `inkey` corresponds to raw data.
- If the `outimgkey` is defined as "Image2D", the image is saved in the event as a `CSPadPixCoords::Image2D<double>` object, otherwise (for other names) as a `ndarray<const double,2>` object.

See also Example for Module CSPadPixCoords::CSPad2x2ImageProducer.

Module CSPadPixCoords::CSPadNDArrProducer

CSPadNDArrProducer is a module for psana framework. It uses specified `source` and `inkey` parameters,

- gets CSPAD configuration from the environment frame `Psana::CsPad::ConfigV#`,
- gets CSPAD data from the event frames `Psana::CsPad::DataV#`, `Psana::CsPad::ElementV#`,
- creates the data array as `ndarray<const T,3>`, where T stands for `outtype`, which may be **double**, **float**, **int**, or **uint16**,
- puts the data array in the event with `outkey` tag.

The data array `ndarray<const T,3>` has a shape (N,185,388), where N32.
This array is combined from data arrays for quads, `ndarray<const int32_t,3>` with shape (M,185,388), where M8, taking into account the CSPAD configuration.

> ⚠ In real data some 2x1 may be turned off, that can be seen in configuration object.

parameter	default value	description
`source`	":Cspad.0"	source of data
`inkey`		key for data processing stage
`outkey`	"cspad_ndarr"	output key for image saved in event
`outtype`	float	type of output data from the list of supported (float, double, int, int16)
`is_fullsize`	false	size of output array; true means [32,185,388], false means "as data" [N,185,388]
`is_2darray`	false	if `is_fullsize=true && is_2darray=true` then shape of output 2-d array is [32*185,388]
`print_bits`	0	verbosity: • =0 - print nothing • +1 - input pars • +2 - configuration parameters • +4 - consumed time per event • +16 - print type size and one character id name

See also Example for Module CSPadPixCoords::CSPadNDArrProducer.

Module CSPadPixCoords::CSPad2x2NDArrProducer

CSPad2x2NDArrProducer is a module for psana framework. It uses specified `source` and `inkey` parameters,

- gets CSPAD2x2 configuration from the environment frame `Psana::CsPad2x2::ConfigV#`,
- gets CSPAD2x2 data from the event frames `Psana::CsPad2x2::ElementV#`,
- creates the data array as `ndarray<const T,3>`, where T stands for `outtype`, which may be **double**, **float**, **int**, or **uint16**,
- puts the data array in the event with `outkey` tag.

The data array `ndarray<const T,3>` has a shape (185,388,2).

In real data some 2x1 may be turned off, that can be seen in configuration object.

parameter	default value	description
`source`	":Cspad2x2.0"	source of data
`inkey`		key for data processing stage
`outkey`	"cspad2x2_ndarr"	output key for image saved in event
`outtype`	float	type of output data from the list of supported (float, double, int, int16)
`is_2darray`	false	if `is_2darray=true` then shape of output 2-d array is [185,388*2]
`print_bits`	0	verbosity: • =0 - print nothing • +1 - input parameters • +2 - configuration parameters • +4 - consumed time per event • +16 - print type size and one character id name

See also Example for Module CSPadPixCoords::CSPad2x2NDArrProducer.

Module CSPadPixCoords::CSPad2x2NDArrReshape

CSPad2x2NDArrReshape is a module for psana framework. This module reshapes ndarrays from cspad2x2 data format (185,388,2) to cspad-"natural" format (2,185,388). Input arrays for re-shaping are specified by the `source` and `keys_in` parameters. Output arrays - by the `source` and `keys_out` parameters. Currently four data type are supported for input arrays; int16, int, float, and double of const and non-const types. Output arrays have the same data type as input.

Conventions for key lists description

1. keys_in should not be empty to convert something,
2. keys should be space separated, ex.: keys_in = k1 key2 key3 k4,
3. if keys_out is non-empty then the number of keys in keys_out should be equal to the number of keys in keys_in and their names will be used without any modifications,
4. If keys_out is empty, then all keys are defined by the list of key_in
    - if the key in key_in has a suffix specified by the colon ":", this suffix will be dropped for the output key, ex.: key12:xyz -> key12
    - if the key suffix is missing in the input key, then it will be added to the output key as :2x185x388, ex.: key12 -> key12:2x185x388

Configuration parameters:

parameter	default value	description
`source`	":Cspad2x2"	source of data
`keys_in`		list of input keys separated by space. At least one should be specified. May have suffixes as <key>:<suffix>, which will be dropped for default `keys_out`
`keys_out`		list of output keys separated by space. Is empty by default, list will be generated from the list defined in `keys_in` dropping or adding suffixes

print_bits	0	verbosity: • =0 - print nothing • +1 - input parameters • +2 - input/output keys • +4 - warning about non-found keys in the event store • +8 - warning about non-found keys in the calib store • +16 - print type size and one character id name

See also Example for Module CSPadPixCoords::CSPad2x2NDArrReshape.

JIRA Issue: **PSAS-45** - Getting issue details... STATUS

Package ImgPixSpectra

For complete reference see the Doxygen documentation.

> ⚠ Package ImgPixSpectra is available since release `ana-0.4.1`. It uses classes from the package CSPadPixCoords V00-02-01 or higher. To add this package in release use command:
> `addpkg CSPadPixCoords HEAD`

Package ImgPixSpectra is intended to accumulate the spectra for all pixels of the image array.
Different modules of this package work with different data types for detectors like CSPad, CSPad2x2, Opal, Princeton camera, etc. All modules have the same interface and the same functionality.
In the loop over events from `beginJob` to `endJob` the image pixel amplitudes are accumulated in the 2-d array,
of the shape (<number-of-pixels>, <number-of-spectral-bins>). The first parameter is defined by the image size. The second is passed as an external parameter of the psana configuration file (`psana.cfg`) along with minimal and maximal amplitudes. At the `endJob` the spectral array is saved in file with specified name. Auxiliary file with the name extension `*.sha` is created in order to save the shape parameters. For example, the "cspad2x2-pix-spectra.txt.sha" outpuit file contains

```
NPIXELS  143560
NBINS    100
AMIN     500
AMAX     1000
NEVENTS  2549
ARRFNAME cspad2x2-pix-spectra.txt
```

This information can be used in analysis or presentation of this array.

Module ImgPixSpectra::CSPadPixSpectra

List of parameters:

parameter	default value	description
source	"CxiDs1.0:Cspad.0"	source of data for CSPad
events	1<<31U	number of events before stop a job
inputKey		input key for data processing stage
amin	0.	minimal spectral amplitude
amax	1000.	maximal spectral amplitude
nbins	100	number of bins in spectra
arr_fname	"..._spectral_array.txt"	output file name

See also: Example for Module ImgPixSpectra::CSPadPixSpectra.

Module ImgPixSpectra::CSPad2x2PixSpectra

The only difference in interface of this module from previous is in the default name for the source parameter and the output file name.

parameter	default value	description
source	"DetInfo(:Cspad2x2)"	source of data for CSPad2x2

See also: Example for Module ImgPixSpectra::CSPad2x2PixSpectra.

Module ImgPixSpectra::CameraPixSpectra

The only difference in interface of this module from previous is in the default name for the source parameter and the output file name.

parameter	default value	description
source	"DetInfo(SxrBeamline.0:Opal1000.1)"	source of data for Opal camera

or:

parameter	default value	description
source	"DetInfo(:Princeton)"	source of data for Princeton camera

See also Example for Package ImgPixSpectra.

Package ImgAlgos

This packages contains a few psana modules for analysis and image processing.

Module ImgAlgos::Tahometer

This module measures the time interval for entire job and for each dn events and prints the rate info as requested by the print_bits parameter.

parameter	default value	description
dn	100	number of events between printout
print_bits	2	filter verbosity: • =0 - print nothing • +1 - input pars • +2 - start notice and summary at stop • +4 - instant rate performance after each dn events • +8 - print summary parameters for parser in log file

See also Example for Module ImgAlgos::Tahometer.

Module ImgAlgos::TimeStampFilter

This module passes only the events if their time stamp is in the requested range.
The range of allowed time stamps is defined by the configuration parameters.

parameter	default value	description
tsinterval	"1970-01-01 00:00:00.000000000 / 2100-01-01 00:00:00.000000000"	time-stamp interval string
tstamp_min	"1970-01-01 00:00:00.000000000"	minimal time-stamp string
tstamp_max	"2100-01-01 00:00:00.000000000"	maximal time-stamp string
filterIsOn	true	On/Off the filter

`print_bits`	0	filter verbosity: • =0 - print nothing • +1 - input pars • +2 - summary • +4 - event ID • +8 - time stamp variables for selected event

The time-stamp string is accepted in various formats:

- `YYYYMMDD HHMMSS.FFF`
- `YYYYMMDDTHHMMSS.F`
- `YYYY-MM-DD HH:MM:SS.FFF`
  but the date field has to be presented mandatory.
  If the `tsinterval` is defined and is different from the default, it will be used in filter and override the `tstamp_min` and `tstamp_max`.

See also Example for TimeStampFilter and XtcOutputModule.

Module ImgAlgos::EventNumberFilter

This filter selects events by their number counting from the beginning of job, starting from 0. The event number is not a parameter which is associated with event. Use this filter cautiously on your own risk for debugging purpose only.

> ⚠ The unique parameter associated with the event is a **time-stamp**. The event number is not defined in the xtc file and is not a recommended to use parameter. The events are counted locally inside this filter from the beginning of job starting from 0. Be cautious of using this filter in consecutive jobs for different runs! For example, this filter returns different subsets of events for the files with raw and selected events.

parameter	default value	description
`filterIsOn`	true	On/Off filter
`first`	0	the first event from the beginning of job, starting from 0
`last`	1<<31	the last event from the beginning of job
`evtstring`		the string of events of intervals from the beginning of job
`print_bits`	0	filter verbosity: • =0 - print nothing • +1 - input pars • +2 - summary • +4 - selected event number and ID

There are two modes of operation of this filter.

1. If the `first` and/or `last` event numbers are defined, then the filter will select events in this range only.
2. If the `evtstring` is defined, only listed events of event ranges will be selected. For example, the `evtstring`parameter can be defined as

   ```
   2,5,11-15,20-25,29,30
   ```

   that means the list of events:

   ```
     2   5   11   12   13   14   15   20   21   22   23   24   25   29   30
   ```

   In the `evtstring` parameter the comma "," and sign minus "-" as a dash are the only allowed separators. Blank spaces are also allowed. Other characters may abort the program. The `evtstring` mode has higher priority than the 1st mode.
   The `filterIsOn` allows easy turn on/off this filter in `*.cfg` file.

Module ImgAlgos::EventCounterFilter

ImgAlgos::EventCounterFilter (ImgAlgos > V00-03-46) module is created by request of Thomas Kroll for experiment with mobile rack in SACLA .

Functionality:

Filter select events which numbers are listed in the input file `ifname`. This module uses local counter started from 1 for the 1-st event of the job and incremented in the event(...) method. The file `ifname` contains the list of integer event numbers in ascending order separated by space or '\n'.

parameter	default value	description
`mode`	1	filter mode: 0-off, 1-on, -1-on in inverted decision mode
`ifname`		input file name, is empty by default
`print_bits`	0	verbosity: • =0 - print nothing • +1 - input pars • +2 - input file content • +4 - filter summary • +16 - current event number • +32 - selection status ("is selected")

Input file (`ifname`)consists of integer numbers in ascending order separated by space or '\n', for example:

```
2 3 6 7 9 11 15
17 28 32 ...
```

Module ImgAlgos::EventCodeFilter

ImgAlgos::EventCodeFilter (ImgAlgos > V00-03-80) module is created by request of Silke Nelson for dark image selection based on EvrData eventCode.

Functionality: Filter selects events which have requested `evtcode` in `Psana::EvrData::DataV4, 3, ...` data objects in the list of `Psana::EvrData::FIFOEvent` objects. Events which do not have the event code are skipped.

parameter	default value	description
`source`	DetInfo(:Evr)	data source
`evtcode`	0	event code, unsigned number
`mode`	0	filter mode: 0-off, 1-on, -1-on in inverted decision mode
`print_bits`	0	verbosity: • =0 - print nothing • +1 - input pars • +4 - filter summary • +8 - current event number and selection status • +16 - list of FIFO events

Example of configuration parameters for this module:

```
modules = ... ImgAlgos.EventCodeFilter ...

[ImgAlgos.EventCodeFilter]
evtcode    = 41
mode       = 1
print_bits = 5
```

Module ImgAlgos::AndorImageProducer

Functionality:

- gets Andor data from the event store for specified `source` and `key_in` parameters,
- puts the `ndarray<const T,2>` object with camera image in the event store using specified `key_out` parameter, where the output type T is defined by the `outtype` parameter.

parameter	default value	description
`source`	"DetInfo(:Andor)"	source of data
`key_in`		key for input data
`key_out`	"andorimg"	output key for image saved in the event store
`outtype`	"asdata"	out data type: implemented values: asdata (default, uint16_t), float, double, int and int16.
`print_bits`	0	verbosity: • =0 - print nothing • +1 - input pars • +2 - data from event() • +4 - configuration parameters • +8 - table with sizes of types

Module ImgAlgos::EpixNDArrProducer

Functionality:

- gets Epix data from the event store for specified `source` and `key_in` parameters,
- puts the `ndarray<const T,2>` object with image in the event store using specified `key_out` parameter, where the output type T is defined by the `outtype` parameter.

parameter	default value	description
`source`	"DetInfo(:Epix)"	source of data, it needs to be replaced for real detector to :Epix100a
`key_in`		key for input data
`key_out`	"epix-ndarr"	output key for image saved in the event store
`outtype`	"asdata"	out data type: implemented values: asdata (default, uint16_t), float, double, int and int16.
`print_bits`	0	verbosity: • =0 - print nothing • +1 - input pars • +2 - data from event() • +4 - configuration parameters • +8 - table with sizes of types

See also Example for Module ImgAlgos::EpixNDArrProducer.

Module ImgAlgos::PnccdNDArrProducer

Functionality:

- gets pnCCD data from Psana::PNCCD::FramesV1 object from the event for specified `source` and `key_in` parameters,
- puts the `ndarray<const TOUT,3>` object of shape 4x512x512 specified `key_out` parameter.

parameter	default value	description
`source`	"DetInfo(:pnCCD)"	source of data
`key_in`		key for input data
`key_out`	"pnccd-ndarr"	output key for image saved in event
`outtype`	"asdata"	out data type: implemented values: asdata (default, unsigned short), float, double, int and int16.

print_bits	0	verbosity: • =0 - print nothing • +1 - input pars • +2 - data from event() • +4 - configuration parameters • +8 - table with sizes of types • +16 - warning about missing data

Module ImgAlgos::PnccdImageProducer

Functionality:

- gets from the event store the object with pnCCD data of type
  - Psana::PNCCD::FullFrameV1 containing four [512][512] frames with T=uint16_t, or
  - ndarray<const T,3> , where shape=[4][512][512], T=unsigned short, float, double, int, or int16,

  for specified source and inkey parameters
- puts the ndarray<const T,2> object with pnccd [1024+gap_rows][1024+gap_cols] image in the event using specified outimgkey parameter.

parameter	default value	description
source	"DetInfo(:pnCCD)"	source of data
inkey		key for input data
outimgkey	"pnccdimg"	output key for image saved in event
gap_rows	0	gap between top and bottom segments in number of pixels
gap_cols	0	gap between left and right segments in number of pixels
gap_value	0	Image effective pixel intensity value in the gap
print_bits	0	verbosity: • =0 - print nothing • +1 - input pars • +2 - initial portion of pnccd array • +4 - configuration pars • +16 - warning about missing data

See also Example for Module ImgAlgos::PnccdImageProducer.

Module ImgAlgos::CameraImageProducer

This module works with any generic camera image stored in data type Camera::FrameV#:

- gets any camera image data Camera::FrameV1 from the event store for specified source and key_in parameters,
- puts the ndarray<const T,2> object with camera image in the event store using specified type outtype and key_out parameters.

⚠ Special treatment for fccd960: if outtype is not "asdata" the gain factor depending on gain bits is applied, See FCCD-Detector.

parameter	default value	description
source	"DetInfo(:Camera)"	source of data
key_in		key for input data
key_out	"pnccdimg"	output key for image saved in event
outtype	"asdata"	out data type: implemented values: asdata (default, uint16_t), float, double, int and int16.
subtract_offset	true	on/off the amplitude offset using configuration data (not applied for outtype = asdata)

print_bits	0	verbosity: • =0 - print nothing • +1 - input pars • +2 - time stamp for each event • +4 - summary at the endJob • +8 - details about data format conversion and applied offset • +16 - configuration data for each beginCalibCycle

See also Example for Module ImgAlgos::CameraImageProducer.

Module ImgAlgos::PrincetonImageProducer

Functionality:

- gets the Princeton or Pimax camera image data Princeton::FrameV1/2 or Pimax::FrameV1 from the event store for specified source and key_in parameters,
- puts the ndarray<const T,2> object with camera image in the event using specified type outtype and key_out parameters.

parameter	default value	description
source	"DetInfo(:Princeton)"	source of data
key_in		key for input data (raw - by default)
key_out	"image"	output key for image saved in event
outtype	"asdata"	out data type: implemented values: asdata (default, uint16_t), float, double, int and int16.
print_bits	0	verbosity: • =0 - print nothing • +1 - input parameters • +2 - record for each event • +4 - summary at the endJob • +8 - first 10 elements of the data array • +16 - configuration info for each beginCalibCycle(...)

See also Example for Module ImgAlgos::PrincetonImageProducer.

Module ImgAlgos::AcqirisArrProducer

- Gets acqiris configuration and data from Acqiris::ConfigV1 and Acqiris::DataDescV1, objects using parameters source and key_in ;
- produces ndarray<const double,2> of shape[] = {nbrChannels, nbrSamples} for waveforms and time stamps;
- saves configuration data in the file defined by fname_prefix;
- saves waveforms and time stamps in the event store with keys key_wform and key_wtime.

parameter	default value	description
source	"DetInfo(:Acqiris)"	source of data
key_in	""	key for input data (by default (empty) – raw data)
key_wform	"acq_wform"	output key for waveforms saved in event
key_wtime	"acq_wtime"	output key for waveform times saved in event
fname_prefix	""	file name prefix for configuration parameters (by default (empty) – do not save file)
correct_t	true	on/off switch for time correction; if =false - array indexes are the same as in raw data waveform

print_bits	0	verbosity: • =0 - print nothing • +1 - input parameters • +2 - configuration parameters • +4 - record about saving file with configuration parameters • +8 - print part of waveform for all channels and segments in each event • +16 - print info about waveform indexes

Example for Module ImgAlgos::AcqirisArrProducer

Module ImgAlgos::AcqirisAverage

- Gets Acqiris waveforms from event store as ndarray<const double,2> object using parameters source and key_in;
- performs waveform selection controlled by parameters: thresholds, is_postive_signal, do_inverse_selection, in the range depending on local event numbers skip_events and proc_events;
- after number of events proc_events or at the end of job (whatever happens first), saves array of averaged waveforms in the text file with name constructed from fname_ave_prefix and in the event store using parameters source and key_average.

parameter	default value	description
source	"DetInfo(:Acqiris)"	Source of data.
key_in	"acq_wform"	Key for input data (raw - by default).
key_average	"acq-ave"	Keyword for averaged waveform array saved in the evt store. If empty – array is not saved.
fname_ave_prefix	"acq-ave"	Text file name prefix for averaged array, full name will be extended by the experiment name, run number and suffix "-ave-wfs.txt", for example: "acq-amo01509-r0125-ave-wfs.txt".
thresholds	""	List of threshold values for all Acqiris channels separated by space. If empty – threshold selection is not applied, all waveforms are averaged.
is_positive_signal	""	Space-separated list of 1/0 values indicating signal polarity. For example, "1 1 1 0 1" (without the quotes!) would indicate Acqiris channels 1,2,3,5 contained positive signal polarity, while channel 4 contained a negative signal polarity.
do_inverse_selection	""	Space-separated ist of 1/0 values indicating which waveforms to include in average. For example, "0 0 0 1 0" (without the quotes) would tell the code to average only waveforms that **do not** cross the threshold for channels 1,2,3,5 while channel 4 would average only waveforms that **do** cross the threshold.
skip_events	0	Number of events (from the beginning of job) to skip before begin averaging.
proc_events	10000000	Number of events for averaging.
print_bits	0	Verbosity: • =0 - print nothing • +1 - input parameters • +2 - ndarray dimensions • +4 - begin/end accumulate statistics record • +8 - record about saving file with averaged array • +16 - statistics of averaged waveforms (number of accumulated) • +32 - part of the input waveform

Example for Module ImgAlgos::AcqirisAverage

Module ImgAlgos::AcqirisCalib

- Gets Acqiris waveforms from event store as ndarray<const double,2> object using parameters source and key_in;

- processes events in the range depending on local event numbers `skip_events` and `proc_events;`
- at the 1st processed event loads the fname_base_line file with baseline `ndarray<const double,2>`;
- subtract baseline from waveforms;
- save corrected waveforms in the event store as `ndarray<const double,2>` object using `parameters source` and `key_out.`

parameter	default value	description
`source`	"DetInfo(:Acqiris)"	Source of data.
`key_in`	"acq_wform"	Key for input ndarray with raw waveforms from `evt` store. For now it is assumed that this array is produced by the `AcqirisArrProducer` or `AcqirisAverage` modules.
key_out	"wf-calibrated"	Key for output ndarray with calibrated waveforms.
fname_base_line	"acq-ave"	Name of the input text file with array of the baselines for active Acqiris channels. By default this name coincides with the name of the file produced by the module `AcqirisAverage`. This allows to use these to modules in chain with auto-generated names of files. If the file with auto-generated extended name (which looks like "`acq-ave-amo01509-r0125-ave-wfs.txt`") is not found on disk, the file name without extension will be tested. If it is not found as well, baseline subtraction will not be performed and output array will be identical to input one.
`skip_events`	`0`	Number of events (from the beginning of job) to skip before begin subtraction.
`proc_events`	`10000000`	Number of events for subtraction.
`print_bits`	`0`	Verbosity: - =0 - print nothing - +1 - input parameters - +2 - ndarray dimensions - +4 - begin/end of subtraction - +8 - Information about loading of the input file with baseline array - +16 - part of the input array - +32 - part of baseline array

Example for Module ImgAlgos::AcqirisCalib

Module ImgAlgos::AcqirisCFD

- Gets Acqiris waveforms from event store as `ndarray<const double,2>` object using parameters `source, key_wform, key_wtime`;
- runs constant-fraction discriminator algorithm on all acqiris channels using user-specified per-channel parameters;
- saves edges into the event as a set of ndarray<double,1>`;`

parameter	default value	description
`source`	"DetInfo(:Acqiris)"	Source of data.
`key_wform`	"acqiris_wform"	Key for input ndarray with waveforms (either raw, or subtracted using AcqirisCalib module) from `evt` store.
`key_wtime`	"acqiris_wtime"	Key for input ndarray with waveform times from `evt` store.
key_edges	"acqiris_edges_"	Key for output ndarray<double,1> with calibrated waveforms. This key will have the acqiris channel number (1 thru 20) appended to the end of it, and the data for that channel will be added to the event only if edges were found.
baselines	""	A list of baseline values (one per channel) to subtract from the waveform in volts.
fractions	""	A list of fractions (one per channel) between 0 and 1. The edge-time reported will be at the time when the pulse is at this fraction of the peak value.
thresholds	""	A list of threshold values (one per channel) in volts that indicate a new edge should be found. If this value is less than the baseline, then the algorithm will look for negative pulses, otherwise it will look for positive pulses.

deadtimes	""	A list of deadtimes (one per channel) in seconds. After each edge the algorithm will ignore any new hits in this time interval.
leading_edges	""	A list of 0/1 values (one per channel) indicating whether edge-times are desired for leading edges (1) or falling edges (0).

Example for Module ImgAlgos::AcqirisCFD

Module ImgAlgos::NDArrImageProducer

This module converts any (detector-associated) ndarray in to image.

For any available in the event store `ndarray<const T,NDim>`, defined by the `source` and `key_in` parameters, and which size is equal to the number of pixels in the detector, this module creates 2-d image `ndarray<const TOUT,2>` and saves it in the event store with key_out. TOUT is defined by the type_out. Pixel geometry array is retreived from the calibration file of "geometry" type from default standard or specified in `calibdir` directory.

Configuration parameters:

parameter	default value	description
`source`	""	input source of data. This parameter MUST BE specified, for example: CxiDs1.0:Cspad.0
`key_in`	""	key for input data, for example it might be "calibrated", by default (empty) raw data
key_out	"image"	key for output image in the event store.
type_out	"asinp"	type of data in output image array. Possible values - "asinp"-the same type as input array, "int16", "float", "double", "int".
`print_bits`	0	module verbosity: • =0 - print nothing • +1 - input pars • +2 - info about (non-) availability of the calibration file • +4 - event record • +8 - info about produced image • +16 - size of types table • +32 - pixel coordinates • +64 - detailed information about geometry file processing

Additional parameters which can be used for special applications like alignment:

parameter	default value	description
`calibdir`	""	non-default calibration directory for "geometry" file
`calibgroup`	""	by-default calibration group is retreived from source name as `<detector>::CalibV1`
`oname`	""	top object name in the hierarchial geometry file
`oindex`	0	top object index in the hierarchial geometry file
`pix_scale_size_um`	0	set the ize of the image bin. By-default this value is extracted from sensor.
`x0_off_pix`	0	offset of the top geometry object origin x0 coordinate on the image in number of pixels
`y0_off_pix`	0	offset of the top geometry object origin y0 coordinate on the image in number of pixels
`mode`	0	mapping algorithm for overlapping pixels of ndarray on the 2-D image; 0-last pixel substitutes intensity, 1-use pixel with maximal intensity, 2-accumulate(sum) intensity
`do_tilt`	true	on/off tilt angle correction for sensors

See also Example for Module ImgAlgos::NDArrImageProducer.

Module ImgAlgos::NDArrAverage

This module averages over events the per-element data of the image array (`ndarray<const T,NDim>`, where T is implemented for almost all types: int, int16, uint, float, double etc., NDim5) and saves files for sum, averaged, rms values, mask, and, the hot pixel map. Input `ndarray` can be specified by the `source` and `key` parameters. Averaging may have up to three stages, depending on configuration parameters:

- 0-stage: the pixel amplitudes are averaged without any constrains for events from 0 to `evts_stage1`, the preliminary averaged and rms values are defined for each pixel at the end of this stage.
- 1-stage: starting from event `evts_stage1` the pixel data are collected only for `abs(amplitude-average0) < gate_width1`. At the end of this stage the preliminary averaged and rms values are defined for each pixel.
- 2-stage: starting from the event `evts_stage1 + evts_stage2` the pixel data are collected only for `abs(amplitude-average1) < gate_width2`. At the end of this stage the preliminary averaged and rms values are defined for each pixel and saved in the files specified by the `avefile` and `rmsfile` parameters, respectively.
  This 3-stage averaging algorithm eliminates large statistical fluctuations in the pixel amplitude spectrum.

If the parameter `thr_rms_ADU` is set 0 then threshold value is defined automatically using constrained averaging of pixel rms values in 3 iterations. The threshold value is defined as `mean+5*rms`.

parameter	default value	description
`source`	DetInfo(:Opal1000)	input source of data
`key`		key for input data, for example, it might be "calibrated"
`sumfile`	""	out file with sum of amplitudes, saved if the name is not empty
`avefile`	""	out file with averaged amplitudes, saved if the name is not empty
`rmsfile`	""	out file with rms, saved if the name is not empty
maskfile	""	out file with pixel mask with 0/1-for bad/good pixels, saved if the name is not empty
`hotpixfile`	""	out file with pixel bit-words: 0/1/2/4/8 for good/hot/saturated/cold/cold-rms, saved if the name is not empty
`maxfile`	""	out file with maximal value per pixel over events
`ftype`	`txt`	out file type: `txt` (default), `metatxt, bin`
`thr_rms_min`	0.	minimal threshold on rms (in ADU); If rms lower than this threshold - pixel is cold-rms.
`thr_rms_ADU`	10000.	maximal threshold on rms (in ADU); =0 - use auto-evaluated threshold. If rms exceeds this threshold - pixel is hot.
`thr_min_ADU`	-100000.	threshold on minimal intensity (in ADU); if ave exceeds this threshold - pixel is good
`thr_max_ADU`	100000.	threshold on maximal intensity (in ADU); if ave exceeds this threshold - pixel is bad
`evts_stage1`	1000000	number of events before stage 1
`evts_stage2`	0	additional number of events before stage 2
`gate_width1`	0	gate_width for stage 1
`gate_width2`	0	gate_width for stage 2
`print_bits`	0	module verbosity: • =0 - print nothing • +1 - input pars • +2 - event record • +4 - beginning of 3 stages • +8 - processed statistics at the end of each stage • +16 - records for saved files • +32 - summary with keywords for parser • +64 - statistics of bad pixels • +128 - mean and rms in 3 iterations of the threshold auto-evaluation algorithm and evaluated threshold • +256 - warning if ndarray shape is not defined yet

If all file names are empty (by default), the files with pre-defined names "arr-ave-<exp>-r<run>.dat" and "arr-rms-<exp>-r<run>.dat" will be saved for averaged and rms arrays, respectively. Otherwise, the files with specified names will be saved.
Default parameters are set for regular single-stage averaging without any constrains.
See also Example for Module ImgAlgos::NDArrAverage.

Module ImgAlgos::NDArrCalib

Functionality

- NDArrCalib uses the `source` and `key_in` parameters to get any `ndarray<const T,NDIM>` object from the event store, where `T` stands for `int16_t, uint16_t, int, float, uint8_t,` or `double, 1NDim5`,

- automaticly gets parameters from calibration store for types `pedestals, common_mode, pixel_status, pixel_gain, and pixel_rms,`
- gets parameters from user-defined files `fname_bkgd` and `fname_mask`, if their names are specified,
- the specified by the `do_...` parameter corrections are applied to raw data `ndarray<const T,NDIM>` as follows:

    1. subtracts **pedestals**,
    2. subtracts **common mode**,
    3. subtracts normalized **background**,
    4. apply **gain** factors,
    5. apply hot/bad **pixel_status** mask,
    6. apply **mask** (1/0 = good/bad pixels),
    7. apply **threshold** as a common low level,
    8. apply per-pixel **threshold** as N*RMS,

- and saves the corrected ndarray`<const TOUT,NDim>` in the event with key `key_out`, where `TOUT` is controlled by the parameter `outtype, w` hich can be set to `double`(default), `float`, `int,` and `int16`.
- In ImgAlgos V00-02-01 implemented detectors: CsPad, CsPad2x2, Pnccd, Princeton, Andor, Opal1000, Opal4000

Control on corrections

```
A_cor = A_raw
        (1) - pedestal             | if do_peds==true and pedestals are available in calib store
        (2) - common mode          | if do_cmod==true and common_mode parameters are available in calib store
        (3) - N*background         | if do_bkgd==true and the file name is specified in the parameter
fname_bkgd and bkgd_ind_* are set
        (4) * gain                 | if do_gain==true and pixel_gain are available in calib store
        (5) apply bad pixel status | if do_stat==true and pixel_status are available in calib store
        (6) apply mask             | if do_mask==true and the file name is specified in the parameter
fname_mask. Parameter masked_value is used to substitute masked values.
        (7) apply N*RMS threshold  | if do_nrms==true and pixel_rms are available in calib store, parameters
threshold_nrms and below_thre_value are set
        (8) apply common threshold | if do_thre==true. Parameter below_thre_value is used to substitute below
threshold values.
```

Configuration parameters

parameter	default value	description
`source`	DetInfo(:Camera)	source of data
`key_in`		key for input `ndarray<const T,NDIM>`
`key_out`	calibrated	output key for calibrated image saved in event
`outtype`	double	output ndarray data type can be set to `double`(default), `float`, `int,` and `int16`.
`do_peds`	false	true: pedestals subtracted if available in calib store
`do_cmod`	false	true: common mode correction is evaluated and applied [Ref.]
`do_stat`	false	true: bad/hot pixels in pixel_status are masked
`do_mask`	false	true: mask is applied if the file `fname_mask` is available (1/0 = good/bad pixels)
`do_bkgd`	false	true: normalized background is subtracted if the file `fname_bkgd` is available
`do_gain`	false	true: `pixel_gain` correction is applied if available in calib store
`do_nrms`	false	true: per-pixel threshold is applied if `pixel_rms` is available in calib store
`do_thre`	false	true: low level `threshold` in ADU is applied
`fname_bkgd`		input file name for background, applied if the file name is specified
`fname_mask`		input file name for mask, applied if the file name is specified
`masked_value`	0.	intensity value (in ADU) substituted for masked pixels
`threshold_nrms`	3.	threshold as a number of sigmas to `pixel_rms` parameters
`threshold`	0.	common low level threshold in ADU
`below_thre_value`	0.	intensity substituted for pixels below threshold

bkgd_ind_min	0	minimal index in flatten ndarray, which is used for background normalization
bkgd_ind_max	100	maximal index in flatten ndarray, which is used for background normalization
bkgd_ind_inc	2	index increment in flatten ndarray, which is used for background normalization
print_bits	0	verbosity: • =0 - print nothing • +1 - input pars • +2 - calibration parameters • +4 - common mode algorithm parameters • +8 - ndarray parameters; type, ndim, shape, etc. • +16- time stamp for each event • +32 - first 10 elements of the raw image data • +64 - first 10 elements of the calibrated image data

To add implementation for any other new detector "Det":

```
        1) add file pdscalibdata/DetBaseV1.h

        2) in PSCalib/src/GenericCalibPars.cpp add
             #include "pdscalibdata/DetBaseV1.h"
             ...
             template class PSCalib::GenericCalibPars<pdscalibdata::Opal4000BaseV1>;

        3) in PSCalib/include/CalibParsStore.h add
             #include "pdscalibdata/DetBaseV1.h"
             ...
             return new PSCalib::GenericCalibPars<pdscalibdata::DetBaseV1>(calibdir, type_group, src, runnum,
prbits);
```

Doxygen documentation for interface: CalibParsStore, GenericCalibPars

See also Example for Module ImgAlgos::NDArrCalib,

Test of the NDArrCalib module for pnCCD.

Module ImgAlgos::NDArrDropletFinder

Finds "droplets" (wide peaks) in data ndarray and saves their list in output ndarray. This is a re-implementation of algorithm ImgPeakFinder for ndarray of 2-d segments with functionality as follows.
1. Selects pixels in the windows defined by the list of parameters: `segment, rowmin, rowmax, colmin, colmax` with amplitudes above the `threshold_low`.
2. Optionally smears image for selected pixels using 2-d matrix of weights in the range `-smear_radius` to `+smear_radius` in both dimensions around smeared pixel. The matrix of weights is defined by the 2-d Gaussian function of width `sigma`. If `sigma=0` smearing is not applied.
3. Finds peaks as pixels with absolute-maximal amplitude above the `threshold_high` in the center of the matrix covering the range `-peak_radius` to `+peak_radius` in both dimensions.
4. Puts the ndarray of found peaks in the event store with key `key_droplets`. Each row of this ndarray has parameters from `struct Droplet`, containing segment number, row, column of the droplet center, peak pixel amplitude, the total amplitude inside `peak_radius` the region, and the number of pixels in this region above `threshold_low`:

```
  struct Droplet{
    unsigned seg;
    double   row;
    double   col;
    double   ampmax;  // amplitude in the peak maximum
    double   amptot;  // total amplitude in the range of peak_radius
    unsigned npix;    // number of pixels in the range of peak_radius
  };
```

parameter	default value	description

source	"DetInfo()"	source of data
key		key for input data ndarray, default is empty - raw data
key_droplets		key for output list of peaks as std::vector<AlgDroplet::Droplet> (default is empty - do not save)
key_smeared		
threshold_low	10	low threshold on pixel amplitude
threshold_high	100	high threshold on pixel amplitude
sigma	1.5	width of the Gaussian for smearing; =0-no smearing
smear_radius	3	radius in pixel for smearing - radial size of matrix of weights
peak_radius	3	radius in pixel for peak finding - radial size of the region to search for local maximum
low_value	0	value substituted for pixels with intensity below threshold and outside window
windows		list of windows, each window is defined by 5 parameters; `segment-index, rowmin, rowmax, colmin, colmax`, separated by space. Default is empty - process all segments
mask		path to the file with mask, by default (empty) mask is not used
masked_value	0	value substituted for masked pixels (0-masked, non-zero - good pixel)
fname_prefix		Common prefix for saved files. If non-empty - save files with image and list of droplets for each event with found droplets. The file name is formed as <prefix>-r####-e######-<suffix>.txt, where hash stands for number (0-9), suffix may be `raw`, `smeared`, or `peaks`. Default is empty - do not save files.
print_bits	0	module verbosity: • =0 - print nothing, • +1 - input pars in the `beginJob(...)`, • +2 - summary in the `endJob(...)`, • +4 - number of droplets/peaks in the event, • +8 - array of peak parameters in the event, • +16 - print info about saved files (if they are saved) • +64 - info messages from smearing and droplet finding algorithms, • +128 - debugging messages from smearing and droplet finding algorithms, • +256 - details for debugging; messages from windows parser, window parameters accounting for segment limits, • != 0 - all warning messages

Remarks:

- if `print_bits` is not zero - warning messages will be printed.
- Saves table of droplets/peaks as ndarray<float,2> with shape=[ndroplets,6] (see struct Droplet) if the keyword `peaks_droplets` is non-empty.
- A set of threshold parameters should be different in cases when smearing is applied (`sigma` is not zero) or not.

See also Example for Module ImgAlgos::NDArrDropletFinder

Module ImgAlgos::PixCoordsProducer

Functionality

For data `source` in each run loads/updates calibration `geometry` file from the calibration DB, evaluates pixel coordinate, area, mask arrays using class PSCalib::GeometryAccess and saves them as ndarray<const TYPE,1> in the `env.calibStore()` for keys `(TYPE=float) key_out_x, key_out_y, key_out_z, key_out_area(unsigned), key_out_area(int), key_out_ix, key_out_iy, key_gfname(string) and (TYPE=uint8_t) key_fname`.

The main idea of this module is that calibration `geometry` file will be found and loaded (if available) automatically and pixel coordinate, index, and other arrays will be produced and saved in the `env.calibStore()`, if appropriate keys are not empty.

Configuration parameters

parameter	default value	description

source	DetInfo(:Cspad)	source of data
group		group of calibration type, by default will be set from source
key_out_x	x-pix-coords	output key pixel x-coordinate[um] array. If set empty - array is not saved in the env.calibStore.
key_out_y	y-pix-coords	output key pixel y-coordinate[um] array. If set empty - array is not saved in the env.calibStore.
key_out_z	z-pix-coords	output key pixel z-coordinate[um] array. If set empty - array is not saved in the env.calibStore.
key_out_area	""	output key pixel area array, default value is empty - array is not saved in the env.calibStore
key_out_mask	""	output key pixel mask array, default value is empty - array is not saved in the env.calibStore
key_out_ix	""	output key pixel x-coordinate index array, default value is empty - array is not saved in the cenv.alibStore
key_out_iy	""	output key pixel y-coordinate index array, default value is empty - array is not saved in the env.calibStore
key_fname	geometry-calib	path to "geometry" calibration file saved as ndarray<char,1> DEPRICATED! USE key_gfname
key_gfname	geometry-fname	path to "geometry" calibration file saved as std::string
x0_off_pix	0	offset of detector origin x0 in number of pixels before evaluation of indexes, by default offset moves xmin to 0.
y0_off_pix	0	offset of detector origin y0 in number of pixels before evaluation of indexes, by default offset moves ymin to 0.
mask_bits	255	mask control bits. For cspad2x1 sensor mask control bits mean: • =0 - all pixels set to 1 • +1 - mask (set 0) edges • +2 - mask two wide central columns • +4 - mask unbound pixels • +8 - mask unbound pixel neighbours
print_bits	0	verbosity: • =0 - print nothing • +1 - input pars • +2 - calibration file path • +4 - list of geometry objects • +8 - list of geometry objects with children • +16- comments from calibration file • +32 - pixel coordinates

Current version of this module works with CSPAD and CSPAD2x2. It can be extended for other detectors, whenever necessary.

See Example for Module ImgAlgos::PixCoordsProducer

Module ImgAlgos::ImgAverage

This module averages over events the per-pixel data of the image array (ndarray<const double,2>) and saves files for averaged, rms values, and, if requested, the hot pixel mask. Input data can be specified by the source and key parameters. Averaging may have up to three stages, depending on configuration parameters:

- 0-stage: the pixel amplitudes are averaged without any constrains for events from 0 to evts_stage1, the preliminary averaged and rms values are defined for each pixel at the end of this stage.
- 1-stage: starting from event evts_stage1 the pixel data are collected only for abs(amplitude-average0) < gate_width1. At the end of this stage the preliminary averaged and rms values are defined for each pixel.
- 2-stage: starting from the event evts_stage1 + evts_stage2 the pixel data are collected only for abs(amplitude-average1) < gate_width2. At the end of this stage the preliminary averaged and rms values are defined for each pixel and saved in the files specified by the avefile and rmsfile parameters, respectively.
  This 3-stage averaging algorithm eliminates large statistical fluctuations in the pixel amplitude spectrum.

parameter	default value	description
source	DetInfo(:Opal1000)	input source of data
key		key for input data, for example, it might be "calibrated"
sumfile	""	out file with sum of amplitudes, saved if the name is not empty
avefile	""	out file with averaged amplitudes, saved if the name is not empty

`rmsfile`	""	out file with rms, saved if the name is not empty
`hotpix_mask`	""	out file with hot pixel mask, saved if the name is not empty
`hotpix_thr_adu`	10000.	threshold on rms (in ADU); if rms exceeds this threshold - pixel is hot
`evts_stage1`	1000000	number of events before stage 1
`evts_stage2`	0	additional number of events before stage 2
`gate_width1`	0	gate_width for stage 1
`gate_width2`	0	gate_width for stage 2
`print_bits`	0	module verbosity: • =0 - print nothing • +1 - input pars • +2 - event record • +4 - beginning of 3 stages • +8 - processed statistics at the end of each stage • +16 - output in files

If all file names are empty (by default), the files with pre-defined names "img-ave-r####.dat" and "img-rms-r####.dat" (where #### stands for run number) will be saved for averaged and rms images, respectively. Otherwise, the files with specified names will be saved.
Default parameters are set for regular single-stage averaging without any constrains.
See also Example for Module ImgAlgos::ImgAverage.

Module ImgAlgos::ImgMaskEvaluation

This module gets the image data array (ndarray<`const` T,2>), where T stands for double, float, int, uint8_t, or uint16_t, and evaluates two masks:

1. **saturated** mask for pixels, which had an intensity above the saturation-threshold with frequency grater than specified fraction of events.
2. **noise** mask for pixels, which estimated S/N ratio exceeds the S/N-threshold with frequency grater than specified fraction of events.

In the `endJob` this module saves files (if their names are provided) of image size for:

1. **saturated** mask
2. **noise** mask,
3. **combined** mask,
4. **fraction of noisy** events,
5. **fraction of saturated** events.

The S/N is estimated by averaging over neigbouring pixels.

parameter	default value	description
`source`	"DetInfo(:Opal1000")	input source of data
`key`		key for input data, for example, it might be "calibrated"
`file_mask_satu`	"img-mask-satu.dat"	out file with **saturated** mask
`file_mask_nois`	"img-mask-nois.dat"	out file with **noise** mask
`file_mask_comb`	"img-mask-comb.dat"	out file with **combined** mask
`file_frac_satu`	"img-frac-satu.dat"	out file with **fraction of saturated** events
`file_frac_nois`	"img-frac-nois.dat"	out file with **fraction of noisy** events
`thre_satu`	1000000	saturation-threshold (in ADU if the gain correction was not applied to image)
`frac_satu`	0	allowed fraction of saturated events
`dr_SoN_ave`	1	radial size of the area for S/N evaluation
`thre_SoN`	3	noise-threshold (in ADU if the gain correction was not applied to image)
`frac_nois`	0.5	allowed fraction of noisy events

print_bits	0	module verbosity: • =0 - print nothing • +1 - input pars • +2 - event record • +4 - vector of indexes for SoN evaluation • +8 - processed statistics at the end of job • +16 - output in files

See also Example for Module ImgAlgos::ImgMaskEvaluation.

Module ImgAlgos::ImgCalib

ImgCalib gets the raw image from data and process it as follows:

1. subtracts **pedestals**,
2. subtracts normalized **background**,
3. apply **gain** factors,
4. apply **mask**, and
5. apply **threshold** as a constant low level,
6. apply **threshold** as N*RMS,
   and saves the corrected image in the event.

Functionality:

- ImgCalib uses the `source` and `key_in` parameters to get the input raw image (as `ndarray<const T,2>` object), where T stands for int16_t (from V00-03-46), uint16_t, int, float, uint8_t, or double,
- gets the calibration parameters from files `fname_peds`, `fname_bkgd`, `fname_gain`, `fname_mask`, and `fname_rms`, if their names are specified,
- the specified by the file name corrections are applied per-pixel to raw data image as follows:

```
A_cor = A_raw
        (1) - pedestal              | if the file name is specified in the parameter "fname_peds"
        (2) - N*background          | if the file name is specified in the parameter "fname_bkgd"
        (3) * gain                  | if the file name is specified in the parameter "fname_gain"
        (4) apply mask              | if the file name is specified in the parameter "fname_mask"
        (5) apply N*RMS threshold   | if the file name is specified in the parameter "fname_nrms"
        (6) apply threshold         | if the "do_threshold" = true
```

- corrected image is saved in the event with key `key_out` as `double` type.

Details:

- All files with input calibration parameters should have the same shape as image and formatted as regular text file containing 2d matrix (table) of float (or integer) values, with columns separated by space(s), ' ', and rows(lines) terminated by the '\n'.
- Background normalization is performed in window defined by the parameters `bkgd_row_min`, `bkgd_row_max`, `bkgd_col_min`, and `bkgd_col_max`. Normalization factor `N` is evaluated for pixel amplitudes in this window as:
  `N = sum(A_raw - pedestal) / sum(A_bkgd)`.
- Masking algorithm assumes that good pixels in the `fname_mask` file should be marked by '1' (ones) and bad pixels – by '0' (zeros). Mask is applied as a last correction. The bad masked pixel amplitudes are substituted by the `masked_value`.

parameter	default value	description
source	"DetInfo(:Camera)"	source of data
key_in		key for input image
key_out	"calibrated"	output key for calibrated image saved in event
fname_peds		input file name for pedestals, applied if the file name is specified
fname_bkgd		input file name for background, applied if the file name is specified
fname_gain		input file name for gain, applied if the file name is specified
fname_mask		input file name for mask, applied if the file name is specified
fname_rms		input file name for RMS, applied if the file name is specified
masked_value	0.	amplitude value (in ADU) substituted for masked pixels
threshold_nrms	3.	threshold as a number of sigmas from file `fname_nrms`
threshold	0.	constant low level threshold in ADU

below_thre_value	0.	the amplitude substituted in pixels below threshold
do_threshold	false	if true - low level threshold in ADU is applied
bkgd_row_min	0	the window in image, which is used for background normalization
bkgd_row_max	10	the window in image, which is used for background normalization
bkgd_col_min	0	the window in image, which is used for background normalization
bkgd_col_max	10	the window in image, which is used for background normalization
print_bits	0	verbosity: • =0 - print nothing • +1 - input pars • +2 - time stamp for each event • +4 - detailed info about input files for pedestals, mask, background, and gain • +8 - first 10 elements of the raw image data • +16 - first 10 elements of the calibrated image data

The pedestal, background, gain, mask, and N*RMS corrections are applied if associated file name is specified.
The constant low level threshold is applied if do_threshold is true.
Corrections are not applied by default or with empty file names.

See also Example for Module ImgAlgos::ImgCalib.

Module ImgAlgos::ImgIntForBins

Functionality:

- ImgIntForBins uses the source and key_in parameters to get the input image (as ndarray<const T,2> object), where T stands for uint16_t, int, float, uint8_t, or double,
- gets the pixel-bin indexes from files fname_map_bins, which has a size of image,
- calculates average per pixel intensity for each of number_of_bins bin (from 0 to number_of_bins-1),
- saves the 2-D array of <I>(event, bin) in file fname_int_bins.

parameter	default value	description
source	"DetInfo(:Camera)"	source of data
key_in		key for input image
fname_map_bins		input file name for map of bin indexes
fname_int_bins		output file name for intensity(bin,event)
number_of_bins	10	number of bins for output intensities, if map has bins \>=number_of_bins they will be ignored
print_bits	0	verbosity: • =0 - print nothing • +1 - input pars • +2 - time stamp for each event • +4 - detailed info about input file pars • +8 - the first 100 chars of the output intensity string • +32 - info about open/close output file

Module ImgAlgos::ImgRadialCorrection

This module averages the image pixel amplitude in r-phi bins, normalizes it per single pixel and subtracts the average amplitude from each pixel. Image is obtained from event by its source and inkey values. The output corrected image is saved in the event with outkey keyword. Carthesian to polar coordinate transformation is done with respect to center coordinates xcenter, ycenter. The central region (r<rmin) and edges of the image (r>rmax) pixels can be removed from further consideration by setting rmin and rmax. The number of radial bins is defined as an int(rmax-rmin). The number of angular bins is set by n_phi_bins.

parameter	default value	description
source	"DetInfo()"	input source of data
inkey		key for input data, by default use raw data

outkey	"rad_corrected"	output key for further image processing
xcenter	850	x coordinate of the image center
ycenter	850	y coordinate of the image center
rmin	10	radius minimal image is not processed for r<rmin
rmax	1000	radius maximal image is not processed for r>rmax
n_phi_bins	12	number of angular sectors for the background averaging
event	0	test event for print/save
print_bits	0	module verbosity: • =0 - print nothing, • +1 - input pars, • +2 - summary, • +4 - event ID, • +8 - info about input image type (ndarray<double,2> or Image2D<double>)

See also Example for Module ImgAlgos::ImgRadialCorrection.

Module ImgAlgos::ImgPixAmpFilter

The ImgAlgos::ImgPixAmpFilter is a filter for event selection.
This filter counts the number of image pixels in the specified window with amplitude exceeding the `threshold`. If the number of high-amplitude pixels exceed the `numPixMin`, the event is passed for further analysis.

The algorithm performance was tested for CSPad images. This algorithm consumes up to 15 ms/event on psana0205 for full CSPad (1650x1650) window size. For smaller window consumed time is negligible, comparing to the image reconstruction time, which is ~70 ms/event (for `cspad_mod.CsPadCalib` and `CSPadPixCoords::CSPadImageProducer`) on psana0205.

parameter	default value	description
source	"DetInfo(:Cspad)"	source of data
key	"Image2D"	key for input image data
threshold	10	minimal threshold on pixel amplitude
numPixMin	100	minimal number of pixels with amplitude above the threshold
filterIsOn	true	On/Off the filter
xmin	0	minimal column number
xmax	100000	maximal column number
ymin	0	minimal row number
ymax	100000	maximal row number
print_bits	0	verbosity: • =0 - print nothing, • +1 - input pars, • +2 - summary, • +4 - per event number of pixels above the threshold, • +8 - the same as previous, but for each 100's event, • +16 - event ID

Remarks:

- The default `key` ("Image2D") stands for the `CSPadPixCoords::Image2D<double>` image object. Other `key` names work for the `ndarray<const double,2>` image object.
- The `xmin`, `xmax`, `ymin`, `ymax` (in pixels) defines the window in the image for pixel counting. Default values means the entire image range.
- The `threshold`, `numPixMin`, and the window extents have to be adjusted for particular experiment.

Module ImgAlgos::ImgPeakFinder

This algorithm was motivated by users of amo42112:
1. Select the pixels in the window `xmin`, `xmax`, `ymin`, `ymax`
with amplitudes above the `threshold_low`.
2. Optionally smears image for selected pixels, using 2-d matrix of weights over pixels from `-smear_radius` to `+smear_radius` around each smeared pixel amplitude. The matrix of weights is defined by the 2-d Gaussian function of width `sigma`. If `sigma=0` smearing is not applied.
3. Find peaks as pixels with absolute-maximal amplitude above the `threshold_high` in the center of the matrix `-peak_radius` to `+peak_radius`.
4. Put the vector or ndarray of found peaks in the event with key `peaksKey` or `peaks_nda` respectively. Each entry of this vector has an object of the `struct Peak`, containing `x`, `y` positions, peak pixel amplitude, the total amplitude in the matrix, defined by the `peak_radius`, and the number of pixels in the matrix above `threshold_low`:

```
struct Peak{
   double x;
   double y;
   double ampmax;  // amplitude in the peak maximum
   double amptot;  // total amplitude in the range of {{peak_radius}}
   unsigned npix;  // number of pixels in the range of {{peak_radius}}
} ;
```

parameter	default value	description
`source`	"DetInfo()"	source of data
`key`		key for input image data
`peaksKey`	"peaks"	key for output list of peaks as std::vector<Peak> (if empty - do not save)
`peaks_nda`		key for output list of peaks ndarray<float,2> with shape=[npeaks,5]. Is empty by default (if empty - do not save)
`threshold_low`	10	low threshold on pixel amplitude
`threshold_high`	100	high threshold on pixel amplitude
`sigma`	1.5	width of the Gaussian for smearing; =0-no smearing
`smear_radius`	3	radius in pixel for smearing - radial size of matrix of weights
`peak_radius`	3	radius in pixel for peak finding - radial size of the region to search for local maximum
`xmin`	0	minimal column number
`xmax`	100000	maximal column number
`ymin`	0	minimal row number
`ymax`	100000	maximal row number
`testEvent`	0	event number to save images and print info for test purpose
`finderIsOn`	true	On/Off algorithm
`print_bits`	0	module verbosity: • =0 - print nothing, • +1 - input pars, • +2 - summary, • +4 - number of peaks in the event, • +8 - per event each peak parameters, • +16- info about input image format

Remarks:

- This algorithm consumes ~15 ms/event on psana0101 for full Opal1000 (1024x1024) camera image.
- Smearing algorithm use a "safety margin" which is currently set to 10 pixels (offset from each boarder of the full image size).
- Since V00-03-58 saves table of peaks as ndarray<float,2> with shape=[npeaks,5] if key `peaks_nda` is non-empty.

See also Example for Module ImgAlgos::ImgPeakFinder.

Module ImgAlgos::ImgPeakFilter

This module use results and should work after the ImgAlgos::ImgPeakFinder. It gets the vector of peaks for the `source` and `key`, loops over all founded peaks and counts the number of peaks above the thresholds `threshold_peak` and `threshold_total`. If the `selection_mode` is "SELECTION_ON" and the number of found peaks exceeds the `n_peaks_min` the event is passed for further analysis/processing,
the table of found peaks may be saved in file with prefix defined by the `fname` parameter.

parameter	default value	description
source	"DetInfo()"	source of data
key	"peaks"	key for input list of peaks, should be the same as peaksKey in ImgPeakFinder
selection_mode	"SELECTION_ON"	three possible options: • SELECTION_ON is a normal mode for selector • SELECTION_OFF selector is turned off, all events are passed • SELECTION_INV inversed mode for selector - selected events are discarded
threshold_peak	0	threshold on peak amplitude
threshold_total	0	threshold on total peak intensity (in the matrix around peak)
n_peaks_min	1	minimal number of peaks above all thresholds
fname		file name prefix; by default the prefix is empty and file is not saved
print_bits	0	filter verbosity: • =0 - print nothing • +1 - input pars • +2 - summary • +4 - peaks' info • +8 - event record • +16 - event ID

See also Example for Module ImgAlgos::ImgPeakFilter.

Module ImgAlgos::ImgPeakFinderAB

This module finds peaks in the ndarray<const double,2> image object. Algorithm and the list of parameters are the same as described is section Module ImgAlgos::CSPadArrPeakFinder. The only difference is that the image size is defined by the ndarray<const double,2> object.

See also Example for Module ImgAlgos::ImgPeakFinderAB.

Module ImgAlgos::ImgHitFinder

ImgHitFinder is created by request for amo74213. It works pretty similar to ImgCalib, but the threshold algorithms are extended and background subtraction is removed.
It gets the raw image from data and process it as follows:

1. subtracts **pedestals**,
2. apply **gain** factors,
3. apply **mask**, and
4. apply one of the threshold algorithms
   and saves the corrected image or hit-pixel map in the event.

Functionality:

- ImgHitFinder uses the source and key_in parameters to get the input raw image (as ndarray<const T,2> object), where T stands for uint16_t, int, float, uint8_t, or double,
- gets the calibration parameters from files fname_peds, fname_gain, fname_mask, and fname_thre, if their names are specified,
- the specified by the file name corrections are applied per-pixel to raw data image as follows:

```
A_cor = A_raw
        (1) - pedestal              | if the file name is specified in the parameter "fname_peds"
        (2) * gain                  | if the file name is specified in the parameter "fname_gain"
        (3) apply mask              | if the file name is specified in the parameter "fname_mask"
        (4) apply threshold         | if the file name is specified in the parameter "fname_thre"
```

- corrected image is saved in the event with key key_out as double (or unsigned for pixel map) type.

Details:

- All files with input calibration parameters should have the same shape as image and formatted as regular text file containing 2d matrix (table) of float (or integer) values, with columns separated by space(s), ' ', and rows(lines) terminated by the '\n'.
- Masking algorithm assumes that good pixels in the fname_mask file should be marked by '1' (ones) and bad pixels – by '0' (zeros). The bad masked pixel amplitudes are substituted by the masked_value.

parameter	default value	description
source	"DetInfo(:Camera)"	source of data
key_in		key for input image
key_out	"img-hits"	output key for calibrated image saved in event
fname_peds		input file name for pedestals, applied if the file name is specified
fname_mask		input file name for mask, applied if the file name is specified
fname_gain		input file name for gain, applied if the file name is specified
fname_thre		input file name for threshold, applied if the file name is specified and thre_mode is not zero
masked_value	0.	amplitude value (in ADU) substituted for masked pixels
thre_mode	1	threshold mode: 0 - is not applied; 1 - constant level threshold defined by thre_param in ADU; 2 - apply threshold from file fname_thre multiplied by thre_param; 3 - the same as 2 but local peak finding algorithm is on
thre_param	def	threshold parameter - its meaning depends on thre_mode
thre_below_value	0.	the amplitude substituted in pixels below threshold
thre_above_value	def	the amplitude substituted in pixels below threshold, by default - amplitude above threshold is not substituted
win_row_min	1	the window in image, which is used to make hit map
win_row_max	def	the window in image, which is used to make hit map, by default - use all rows-1
win_col_min	1	the window in image, which is used to make hit map
win_col_max	def	the window in image, which is used to make hit map, by default - use all columns-1
print_bits	0	verbosity: • =0 - print nothing • +1 - input pars • +2 - time stamp for each event • +4 - detailed info about input files for pedestals, mask, background, and gain • +8 - first 10 elements of the raw image data • +16 - first 10 elements of the calibrated image data • +32 - print once input and output image data types

Corrections are not applied by default or with empty file names.

See also Example for Module ImgAlgos::ImgHitFinder.

Module ImgAlgos::ImgSpectra

This module is motivated by the discussion with Josef Frisch, Ryan Coffee, Nick Hartmann. In xppi0412 etc. experiments they need to extract two spectra from Opal1000 camera image for signal and reference, evaluate their relative difference, and find peak position in the differential spectrum.
Module mgAlgos::ImgSpectra works as follows:

- gets the ndarray<const double,2> image object from event,
- selects two spectral band regions and integrates amplitudes for each column (it is assumed that both spectra are oriented along the rows),
- saves two spectral arrays for signal and reference bands and their relative difference as another ndarray<const double,2> object with shape (3,<number-of-columns>) in the event.
- Further analytical work is assumed to be done in the next module ImgAlgos::ImgSpectraProc.

parameter	default value	description
source	"DetInfo(:Opal1000)"	source of data
key_in	"img"	key for input image data

key_out	"spectra"	key for output array with spectra
sig_band_rowc	510	signal band central row at column=0
ref_band_rowc	550	reference ...
sig_band_tilt	0	signal band tilt angle
ref_band_tilt	0	reference ...
sig_band_width	10	signal band width in number op rows
ref_band_width	10	reference ...
print_bits	0	module verbosity: • =0 - print nothing • +1 - input pars • +2 - event record • +4 - image shape • +8 - spectral array

See also Example for Module ImgAlgos::ImgSpectra.

Module ImgAlgos::ImgSpectraProc

Works after module ImgAlgos::ImgSpectra.
This module is designed as an example, in order to show how to get access to spectral array.
In particular, method `ImgSpectraProc::getSpectra(...)` shows how to get pointer to data, and method `ImgSpectraProc::printSpectra(...)` iterates over array and selectively prints its elements.

parameter	default value	description
source	"DetInfo(:Opal1000)"	source of data
key_in	"spectra"	key for input image data
print_bits	0	module verbosity: • =0 - print nothing • +1 - input pars • +2 - event record • +4 - spectral array shape • +8 - spectral array

See also Example for Module ImgAlgos::ImgSpectraProc.

Module ImgAlgos::ImgSaveInFile

⚠ Aka depricated modules: CSPadPixCoords::CSPadImageGetTest, CSPadPixCoords::SaveImageInFile, and ImgAlgos::SaveImageInFile.

Module ImgSaveInFile receives from the event the image object using `source` and `key` parameters and saves it in the `ftype` format with prefix file name `fname` for event(s) specified by the parameters `eventSave` or `saveAll`. Currently implemented file formats: `txt`, `bin`, `tiff`, and `png`.

ImgSaveInFile works after the CSPadImageProducer, CSPadInterpolImageProducer, CameraImageProducer, PnccdImageProducer, etc., which produce image object in formats `CSPadPixCoords::Image2D<T>` or `ndarray<const T,2>`, where the `T` stands for one of the data types, double, float, int, uint8_t, or uint16_t.

parameter	default value	description
source	"CxiDs1.0:Cspad.0"	source of data
key	"Image2D"	input image key
eventSave	0	event number to save the CSPad image file
saveAll	false	save or not all selected events
ftype	"txt"	output file format; currently implemented formats `txt`, `bin`, `png`, `tiff`

fname	"img"	prefix of the output file name. Entire file name is formed as <cspad_image_ev><timestamp>.<ftype>
print_bits	0	verbosity: • =0 - print nothing • +1 - input pars • +2 - info about saved files

⚠ Saving in PNG currently works for uint8_t and uint16_t formats only...

See also Example for Module ImgAlgos::ImgSaveInFile

Module ImgAlgos::ImgVsTimeSplitInFiles

This module is a part of complex algorithm, described in Command Line Interface For Time Correlation Analysis.

This module is designed for parallel image processing for correlation analysis.
Functionality:

- get image for each event as an ndarray<const T,2> object,
- splits it for nfiles_out equal parts,
- saves each part of the image for all events in the job in a separate file with name <fname_prefix>-<fname-common>-b<block-number>.<file_type>,
- saves metadata in the text file with name: <fname_prefix>-<fname-common>-med.txt,
- saves counter number and the time records in file: <fname_prefix>-<fname-common>-time.txt.

parameter	default value	description
source	"DetInfo(:Princeton)"	source of data
key	"img"	input image key
fname_prefix	"my-exp"	output files common prefix
file_type	"bin"	output files type: "bin", "txt"
add_tstamp	true	add time stamp to the output file names
nfiles_out	8	number of output files (or number of parts to split image), it works tested and works for nfiles_out < 1016
ampl_thr	1	pixel amplitude threshold in image amplitude units
ampl_min	1	default pixel amplitude if it is below ampl_min
print_bits	0	verbosity: • =0 - print nothing • +1 - input pars • +2 - event record • +4 - job summary • +8 - details about split and saving • +16 - info about saved files • +32 - the first 10 elements of input data

See also Example for Module ImgAlgos::ImgVsTimeSplitInFiles.

Further processing of the files

- <fname_prefix>-<fname-common>-b<block-number>.<file_type>
- <fname_prefix>-<fname-common>-med.txt
- <fname_prefix>-<fname-common>-time.txt
  is implemented in stand-alone c++ module

```
ImgAlgos/app/corana.cpp (or ImgAlgos/test/corana.cpp)
```

⚠ Note, the application in the test directory is compiled and run by the commands:
scons test
<path>/corana -f <fname_data> -t <fname_tau>\ -h -l <logfile>\ -b <basedir>\

where

- <fname_data> is one of the data files: <fname_prefix>-<fname-common>-b<block-number>.<file_type>, which needs to be available;
- <fname_tau> is a file with a list of indexes of tau for evaluation of correlations. By default or if the file is missing, the list of indexes will be generated automatically, and for book-keeping is saved in the file <fname_prefix>-<fname-common>-tau.txt;
- <basedir> is a directory for all data files, which is current by default;
- <logfile> is an output log-file, or standard output by default.

Module ImgAlgos::ImgTimeStampList

This module is reduced from ImgAlgos::ImgVsTimeSplitInFiles.
Functionality is restricted to:

- saves counter number and the time records in file fname.
- print summary parameters for parser, for example:

```
ImgTimeStampList: Summary for parser
BATCH_RUN_NUMBER              0020
BATCH_NUMBER_OF_EVENTS        75
BATCH_FRAME_TIME_INTERVAL_AVE 8.086934
BATCH_FRAME_TIME_INTERVAL_RMS 0.120381
BATCH_FRAME_TIME_INDEX_MAX          74
```

parameter	default value	description
fname	"tstamp-list.txt"	output files common prefix
print_bits	0	verbosity: • =0 - print nothing • +1 - input pars • +2 - event record • +4 - job summary • +8 - print summary parameters for parser

Module ImgAlgos::UsdUsbEncoderFilter

ImgAlgos::UsdUsbEncoderFilter (ImgAlgos > V00-03-43) module is created by request of Thomas Kroll for experiment with mobile rack in SACLA .

This psana module contains an example of how to get UsdUsb::DataV1 object and time-stamps from different data sources for event synchronization purpose.

Functionality:

To work with time-code objects it uses helper class TimeCode. It loads input timing information from file defined by the ifname and store it in the std::vector<TimeCode> member object. For each event the TimeCode object is defined from data; time stamp from PSEvt::EventId, and unique code from UsdUsb::DataV1. TheTimeCode object from data is compared with information loaded from the input file. If the TimeCode object from data is consistent with one of the records in the file event is passed for further processing, otherwise discarded.

parameter	default value	description
source	"DetInfo(:USDUSB)"	source of data
mode	1	filter mode: 0-off, 1-on, -1-on in inverted decision mode
ifname		input file name, is empty by default
ofname		output file name, is empty by default
bitmask	63	bitmask on UsdUsb code, by default use 6 bits

print_bits	0	verbosity: • =0 - print nothing • +1 - input pars • +2 - records from input file • +4 - open/close info for output file • +8 - filter summary • +16 - content of the UsdUsb::ConfigV1 object • +32 - content of the UsdUsb::DataV1 object

Input (ifname) and output (ofname) files have any number of records of the same format. Each record consists of four integer numbers:

- timestamp sec (uint32_t),
- timestamp nsec (uint32_t),
- UsdUsb 6-bit code (uint8_t),
- user defined counter (unsigned).

Example of the input/output file content:

```
1373280273 293197261  51     1
1373280273 301532011  52     2
1373280273 309867156  54     3
1373280273 318206221  58     4
1373280273 326553079  61     5
...
```

See also Example for Module ImgAlgos::UsdUsbEncoderFilter.

Module ImgAlgos::ImgIntMonCorr

This module is intended for CorAna project.
ImgIntMonCorr gets the image and intensity monitor data, evaluate the normalization factor, applies this factor to the image intensity, and saves the corrected image in the event.

Functionality:

- ImgIntMonCorr uses the source and key_in parameters to get the input image (as ndarray<const double,2> object),
- gets and process the intensity monitors' data in accordance with configuration from file fname_imon_cfg,
- intensity normalized image is saved in the event with key key_out. The type of output data is the same as the type of input data.

parameter	default value	description
source	"DetInfo(:Camera)"	source of data
key_in	"calibrated"	key for input image
key_out	"imon_corrected"	key for output calibrated image saved in event
fname_imon_cfg		input file name with intensity monitors' configuration
print_bits	0	verbosity: • =0 - print nothing • +1 - input pars • +2 - time stamp for each event • +4 - intensity monitor data • +8 - first 10 elements of the input image data • +16 - first 10 elements of the output image data • +32 - normalization factor

fname_imon_cfg file content per line: source name, short name, on/off bits for 4-channels, normalization and selection, minimal, maximal and averaged intensities:

```
BldInfo(FEEGasDetEnergy)         FEEGasDetEnergy  1 1 1 1  0 1   3.000000  7.500000  5.250000
BldInfo(XCS-IPM-02)              XCS-IPM-02       1 1 1 0  1 0  10.000000 13.000000 11.500000
BldInfo(XCS-IPM-mono)            XCS-IPM-mono     1 1 1 0  0 1  14.500000 16.000000 15.250000
DetInfo(XcsBeamline.1:Ipimb.4)   Ipimb.4          1 1 1 1  0 0  -1.000000 -1.000000  1.000000
DetInfo(XcsBeamline.1:Ipimb.5)   Ipimb.5          1 1 1 1  0 0  -1.000000 -1.000000  1.000000
```

Module ImgAlgos::IntensityMonitorsData

This module is intended for `CorAna` project.
It gets the 5 intensity monitors data (4 channels for each) and saves them in the text or binary file `file_data`. Comments (or header) for this file is saved separately in `file_header`. It also prints the summary parameters for parser, for example:

```
IntensityMonitorsData: Summary for parser
BATCH_RUN_NUMBER              0020
BATCH_NUMBER_OF_EVENTS        75
```

parameter	default value	description
`imon1`	"BldInfo(FEEGasDetEnergy)"	source name for intensity monitor
`imon2`	"BldInfo(XCS-IPM-02)"	source name for intensity monitor
`imon3`	"BldInfo(XCS-IPM-mono)"	source name for intensity monitor
`imon4`	"DetInfo(XcsBeamline.1:Ipimb.4)"	source name for intensity monitor
`imon5`	"DetInfo(XcsBeamline.1:Ipimb.5)"	source name for intensity monitor
`file_type`	"txt"	file type may be "txt" or "bin"
`file_data`	"intensity-monitor-data.txt"	file name for data from intensity monitors
`file_header`	"intensity-monitor-comments.txt"	file name for comments
`print_bits`	0	verbosity: • =0 - print nothing • +1 - input pars • +2 - event record • +4 - job summary • +8 - summary parameters for parser • +16 - all available data for all sources • +32 - information about opened/closed files

Module ImgAlgos::CSPadArrSaveInFile

This module saves the CSPad data array formatted as [5920=4*8*185][388] in output file for each passes event.

parameter	default value	description
`source`	"DetInfo(:Cspad)"	input source of data
`key`		key for input data, for example, it might be "calibrated"
`outfile`	"cspad-arr"	out file name prefix for saved array
`print_bits`	0	module verbosity: • =0 - print nothing • +1 - input pars • +2 - event ID • +4 - time stamp • +8 - saved file names

Module ImgAlgos::CSPadArrAverage

This module averages the CSPad data array and saves two files for averaged and rms values in CSPad format [5920=4*8*185][388]. In contrast to the cspad_mod.CsPadPedestals, the input data can be specified with a `key`, that allows to average CSPad array for already pre-processed data, for example `"calibrated"`. This feature can be used to evaluate the averaged signal or background event. Implemented algorithm of averaging allows to eliminate large statistical fluctuations in the pixel amplitude spectrum. In advanced case averaging may have up to three stages, depending on configuration parameters:

- 0-stage: the 1st portion of events from 0 to `evts_stage1` is averaged without any constrains, the preliminary averaged and rms values are defined for each pixel
  at the end of this stage.
- 1-stage: starting from the event `evts_stage1` data are collected only for `abs(amplitude-average0) < gate_width1`. At the end of this stage the preliminary averaged and rms values are defined for each pixel.
- 2-stage: starting from the event `evts_stage1 + evts_stage2` data are collected only for `abs(amplitude-average1) < gate_width2`. At the end of this stage the preliminary averaged and rms values are defined for each pixel and saved in the files specified by the `avefile` and `rmsfile` parameters, respectively.

This type of averaging algorithm may be useful for pedestal defenition in case of large amplitude fluctuations.

parameter	default value	description
`source`	"DetInfo(:Cspad)"	input source of data
`key`		key for input data, for example, it might be "calibrated"
`avefile`	"cspad-ave.dat"	out file with averaged amplitudes
`rmsfile`	"cspad-rms.dat"	out file with rms
`evts_stage1`	1<<31U	number of events before stage 1
`evts_stage2`	100	additional number of events before stage 2
`gate_width1`	0	gate_width for stage 1
`gate_width2`	0	gate_width for stage 2
`print_bits`	0	module verbosity: • =0 - print nothing • +1 - input pars • +2 - beginning of 3 stages • +4 - processed statistics at the end of each stage • +8 - output in files • +16 - event ID

Default version of the configuration parameters works the same way as cspad_mod.CsPadPedestals. In this case module gets raw events and stage 0 continues for entire input data sample.

See also Example for Module ImgAlgos::CSPadArrAverage.

Module ImgAlgos::CSPadCommonModeCorrection

Alternative to the `cspad_mod.CsPadCalib` algorithm for the common mode correction.
Takes the CSPad data array with subtracted pedestals, evaluate the average amplitude for each 2x1 section for amplitudes below the threshold, and subtract it from all pixel amplitudes. This algorithm consumes about 30 ms/event on psana0205.

parameter	default value	description
`source`	"DetInfo(:Cspad)"	source of data for CSPad
`inputKey`	"calibrated"	key for input data, by default expects data from `cspad_mod.CsPadCalib`
`outputKey`	"cm_subtracted"	output key for the next data processing stage
`ampthr`	30	threshold to exclude the large pixel amplitudes in average

Module ImgAlgos::CSPadBkgdSubtract

This module uses the CSPad array, specified by the configuration parameters `source` and `inputKey`, subtracts the background, defined in the file `bkgd_fname`, and saves the resulting array in the event with `outputKey`. The subtracted background array is normalized on the sum of pixel amplitudes in the quad section `norm_sector`, which can be set from 0 to 7.

The shape of the CSPad array in the file `bkgd_fname` is [4*8*185][388] for all 2x1 sections.
The shape of the CSPad array in the event, specified by the `source` and `inputKey` or `outputKey`,
is [number_of_sections*188][388] depends on number of available in DAQ 2x1 sections, provided by the masks in CSPad configuration, for example:

```
shared_ptr<Psana::CsPad::ConfigV2> config2 = env.configStore().get(m_str_src);
unsigned mask = config2->roiMask(quad_number); // should be in the range from 0 to 255
```

parameter	default value	description
`source`	"DetInfo(:Cspad)"	input source of data
`inputKey`		key for input data, by default use raw data
`outputKey`	"bkgd_subtracted"	output key for the next data processing stage
`bkgd_fname`	"cspad_background.dat"	file with CSPad array [4*8*185][388] of averaged background
`norm_sector`	0	CSPad sector in quad from 0 to 7
`print_bits`	0	module verbosity: • =0 - print nothing • +1 - input pars • +2 - event ID • +4 - normalization factor for each event • +8 - part of the background array

See also Example for Module ImgAlgos::CSPadBkgdSubtract.

Module ImgAlgos::CSPadMaskApply

This module uses the CSPad array, defined by the configuration parameters `source` and `inkey`, apply the mask from file `mask_fnname` and saves the masked data with key `outkey`. For masked pixels the amplitude will be replaced by the value from `masked_amp`.

The file `mask_fnname` has the same structure as files for pedestals and background with dimensions [4*8*185][388]. Masked pixels are indicated by 0-th in this file. This file can be generated, for example, from the averaged background file, using amplitude threshold. This can be done with auxiliary python script `MakePixelMask.py` as explained in Example for Module ImgAlgos::CSPadMaskApply.

parameter	default value	description
`source`	"DetInfo(:Cspad)"	input source of data
`inkey`		key for input data, by default use raw data
`outkey`	"bkgd_subtracted"	output key for the next data processing stage
`mask_fname`	"cspad_mask.dat"	file with CSPad mask array [4*8*185][388]
`masked_amp`	0	this amplitude will replace the amplitude in each masked pixel
`mask_control_bits`	1	control bits for applied mask: • =0 - do not apply any mask • +1 - apply mask from file • +2 - mask two long edges of 2x1 • +4 - mask two short edges of 2x1 • +8 - mask two short rows in the middle of 2x1 (rows with wide pixels)
`print_bits`	0	module verbosity: • =0 - print nothing • +1 - input pars (in beginJob) • +2 - event ID (in event) • +4 - mask statistics (in beginJob) • +8 - part of the mask array (in beginJob)

Module ImgAlgos::CSPadArrNoise

This module works on CSPad data array shaped as [5920=4*8*185][388], uses the "median algorithm" to evaluate the signal and noise for each pixel, evaluates S/N ratio for each pixel, counts the fraction of events where `S/N > SoNThr`, and writes the same shape arrays for pixel mask and status information in the `maskfile` and `statusfile`, respectively. The `statusfile` contains for each pixel the fraction of events where `S/N > SoNThr`. This module presents a part of features implemented in the module ImgAlgos::CSPadArrPeakFinder.

parameter	default value	description
`source`	"DetInfo(:Cspad)"	input source of data
`key`		key for input data, for example, it might be "calibrated"
`statusfile`	"cspad-pix-status.dat"	out file with pixel status: fraction of noisy images
`maskfile`	"cspad-pix-mask.dat"	out file with pixel mask
`rmin`	3	radial parameter of the area for median algorithm
`dr`	1	radial band width of the area for median algorithm
`SoNThr`	3	S/N threshold for each pixel to be considered as noisy
`frac_noisy_img`	0.1	fraction of noisy images above which pixel is masked in the `maskfile`
`print_bits`	0	module verbosity: • =0 - print nothing • +1 - input pars • +2 - empty • +4 - processed statistics • +8 - output in files • +16 - event ID • +32 - event time stamp • +32 - vector of indexes and map of indexes for the median algorithm

See also Example for Module ImgAlgos::CSPadArrNoise.

Module ImgAlgos::CSPadArrPeakFinder

Module `ImgAlgos::CSPadArrPeakFinder` is a psana-based implementation of the "median algorithm" for peak finding in CSPad data array shaped as [5920=4*8*185][388]. This algorithm was first implemented in myana/Cheetah by Anton Barty and Co. The "median algorithm" assumes that the amplitude level of background and noise for each pixel can be estimated as a mean and RMS of the surrounding pixels, located in the ring with parameters `rmin` and `dr` around the pixel in question. The threshold `SoNThr_noise` on signal over noise (S/N) ratio allow to asset the pixel amplitude as a large noise fluctuation. Statistics of pixels above the S/N threshold accumulated over many images can be used to form the noisy-pixel mask. For example, if the fraction of images where pixel exceeds the S/N threshold grater than certain value (`frac_noisy_imgs`=0.9), the pixel is considered as noisy. The permanent bad pixel mask (see module ImgAlgos::CSPadMaskApply) and dynamically evaluated noisy pixel mask are used to get rid of bad pixels and improve the image quality. Healthy pixels with S/N above threshold (`SoNThr_signal` about 3-5) are treated as potential signals. Using recursive flood-filling algorithm the groups of connected signal pixels can be found and considered as a candidate for a diffraction peaks. Peak finding algorithm uses the amplitude, S/N thresholds, and limits on number of pixels in the connected region (parameters `peak_amp_tot_thr`, `peak_SoN_thr`, `peak_npix_min`, and `peak_npix_max`) in order to define the peak. Finally, the event is selected or discarded depending on number of found peaks and total amplitude threshold, defined by the parameters `event_npeak_min`, `event_npeak_max`, and `event_amp_tot_thr`, respectively.

Description of implemented algorithm:

- in the constructor and `beginJob(...)`method:
  - enter input parameters,
  - (re)set the initial mask of noisy pixels from file `hot_pix_mask_file` (if its name is specified in the configuration file),
  - do necessary initialization of work arrays.
- in the `event(...)`method the main part of "median algorithm" is implemented:
  - fill [4][8][185][388] per-pixel arrays:
    - `m_stat` - number of events with `|S/N| > SoNThr`,
    - `m_signal` - signal amplitude, or 0(zero) for masked pixels,
    - `m_proc_status` - sets 255 for `S/N > SoNThr` or 0(zero) for masked pixels.
  - use arrays `m_proc_status` and `m_signal`to find peaks:
    - iterate over [185][388] 2x1 pixels and find the connected regions (using recursive flood-filling algorithm)
    - create vector of peaks `v_peaks` of `struct Peak`, using `peak_npix_min`, `peak_npix_max`, and `peak_amp_tot_thr` parameters,
  - loop over `v_peaks`, count total amplitude and the number of peaks in the event.
  - decide if the event selected or not based on `event_npeak_min`, `event_amp_tot_thr`, and `selection_mode` parameters.
  - periodically dynamically re-generate the mask, based on `m_stat` array and `frac_noisy_imgs` parameter. When to start and for how many events to update the mask is defined by the `nevents_mask_update` and `nevents_mask_accum` parameters, respectively.
  - save `m_signal` in file for **selected events**, depending on `out_file_bits` parameter.
  - put the vector with peaks `v_peaks` in the evt with key=`key_peaks_out`.
- in the `endJob(...)` method, depending on bit status in `out_file_bits`:

- save current hot-pixel mask in the file `hot_pix_mask_out_file`
- save current fraction of events with noisy/signal pixels in the file `frac_noisy_evts_file`

parameter	default value	description
`source`	"DetInfo(:Cspad)"	input source of data
`key`		key for input data, for example, it might be "calibrated"
`key_signal_out`		key for output signal array. If the string non-empty, the array is added in the datagram for each event (before selection).
`key_peaks_out`	"peaks"	key for vector of found peaks in the selected event
`key_peaks_nda`		key for ndarray<const float,2>, shape=[npeaks,12] of found peaks in the selected event
`hot_pix_mask_inp_file`	"cspad-pix-mask-in.dat"	in read the pixel mask from file
`hot_pix_mask_out_file`	"cspad-pix-mask-out.dat"	out write current pixel mask in the file
`frac_noisy_evts_file`	"cspad-pix-frac-out.dat"	out file with per-pixel fraction of noisy images
`evt_file_out`	"./cspad-ev-"	out file with signal CSPad array. Time stamp is added.
`rmin`	3	radial parameter of the area for median algorithm
`dr`	1	radial band width of the area for median algorithm
`SoNThr_noise`	3	S/N threshold for each pixel to be considered as noisy
`SoNThr_signal`	5	S/N threshold for each pixel to be considered as noisy
`frac_noisy_imgs`	0.9	fraction of noisy images above which pixel will be masked
`peak_npix_min`	4	minimal number of connected pixels for the good peak
`peak_npix_max`	25	maximal number of connected pixels for the good peak
`peak_amp_tot_thr`	0.	threshold on total signal amplitude of the group of connected pixels, if =0:OFF
`peak_SoN_thr`	7.	threshold on peak S/N (S and N are sums over connected pixels)
`event_npeak_min`	10	threshold on minimal number of good peaks for the event selection
`event_npeak_max`	10000	threshold on maximal number of good peaks for the event selection
`event_amp_tot_thr`	0.	threshold on total signal amplitude in ADU of all good peaks for the event selection, if =0:OFF
`nevents_mask_update`	0	number of skipped events before each mask re-evaluation cycle
`nevents_mask_accum`	50	number of events for the mask re-evaluation
`selection_mode`	SELECTION_ON	selection mode, other allowed values are: SELECTION_OFF, or SELECTION_INV
`out_file_bits`	0	control on writing of files: • =0 - save nothing • +1 - save the `hot_pix_mask_out_file` file with current mask array in the `endJob(...)` method • +2 - save the `frac_noisy_evts_file` file with current fraction of noisy events array in the `endJob(...)` method • +4 - save the CSPad signal array for selected events in the file with name like <evt_file_out>~~<counter>~~<run>-<time stamp>.txt • +8 - save the vector of found peaks in file <evt_file_out>~~<counter>~~<run>-<time-stamp>-peaks.txt

print_bits	0	module verbosity: • =0 - summary at the endJob • +1 - input pars (once in beginJob) • +2 - initial and current mask statistics (in constructor and when mask is updated) • +4 - event selection parameters (for each event); event no., time-stamp, mode, number of peaks, and amp_tot • +8 - output in files (for selected events) • +16 - start/stop to collect data for mask re-evaluation (driven by the mode counters) • +32 - event time stamp (for each event) • +64 - vector of indexes and map of indexes for the median algorithm (once in beginJob) • +128 - peak parameters **before selection** (for each event) • +256 - peak parameters saved in file (**for selected events**) • +512 - selection statistics (N<5: for each event; N<50: for each 10-th; N<500: for each 100-th, then for each 1000-th) • +1024 - event time stamp (for selected events) • +2048 - peak finding from connected pixels; this is printed for connected region of signal pixels if npix>peak_npix_min-2 (otherwise too much junk output)

See also Example for Module ImgAlgos::CSPadArrPeakFinder.

Module ImgAlgos::CSPadArrPeakAnalysis

This module is intended for analysis of the results obtained in the peak finding algorithm implemented in the ImgAlgos::CSPadArrPeakFinder module.

1. It gets the vector of peaks defined by the key parameter and prints it.
2. fills ROOT-style histograms and ntuples, and
3. saves histograms and ntuples in file defined by the fname_root parameter - NOT AVAILABLE SINCE V00-03-50 - get rid of root...

parameter	default value	description
source	"DetInfo(:Cspad)"	input source of data
key	"peaks"	key for input data for peaks found in event
fname_root	"file.root"	name of the output file with root histograms and ntuples - NOT AVAILABLE SINCE V00-03-50 - get rid of root...
print_bits	0	module verbosity: • =0 - print nothing • +1 - input pars (once in beginJob) • +2 - summary, number of processed and collected events (in endJob) • +4 - peak parameters (in event) • +8 - event ID (in event)

See also Example for Module ImgAlgos::CSPadArrPeakAnalysis.

Package pyimgalgos

This package contains python modules which work with both frameworks **pyana** and **psana**. Functionality of these modules resembles modules from C++ package **ImgAlgos**. The difference between two frameworks at code level is explained in Migration from pyana to psana.

Module pyimgalgos.cspad_arr_producer

This module gets data from evt store for CSAPD or CSPAD2x2 depending on unique detector name in parameter source, produce numpy array of full scale shape (4,8,185,388) or (185, 388, 2) of specified in dtype type, and saves it in the evt store with unique name key_out. In case of missing 2x1 sections, their pixel amplitudes substituted by the value form val_miss.

parameter	default value	type	description
source	*-*|Cspad-*	string	address of Detector-Id|Device-Id
data_type	int	string	output array data type. Implemented types: int, int8, int16, int32, uint8, uint16, uint32, float, double.
key_out	cspad_array	string	unique keyword for output array identification
val_miss	0	float	intensity value substituted for missing 2x1 sections in data

print_bits	1	int	module verbosity: • =0 - print nothing • +1 - input pars (in __init__) • +2 - calibration parameters • +4 - configuration parameters • +8 - part of the output 2-d array (in event) • +16 - image shape • +32 - warning about wrong array shape • +64 - path to calibration types

See also Examples for package pyimgalgos.

Module pyimgalgos.cspad_image_producer

This module gets from evt store the numpy array identified by key_in of **full scale shape** (4,8,185,388) or (185, 388, 2) for CSPAD or CSPAD2x2, respectively, and produces 2-d image numpy array, taking into account geometry calibration parameters specified by the path calib_dir. The output image array is saved in the evt store with unique name, specified by parameter key_out.

parameter	default value	type	description
calib_dir	'' (empty)	string	path to calibration directory for ex.: /reg/d/psdm/mec/meca6113/calib/CsPad2x2::CalibV1/MecTargetChamber.0:Cspad2x2.1/
source	*-*|Cspad-*	string	address of Detector-Id|Device-Id
key_in	cspad_array	string	keyword for input numpy array, shape=(4, 8, 185, 388) - for cspad or (185, 388, 2) - for cspad2x2
key_out	cspad_image	string	unique keyword for output 2-d image numpy array
print_bits	1	int	module verbosity: • =0 - print nothing • +1 - input pars (in __init__) • +2 - calibration parameters (in the 1st event) • +4 - configuration parameters (in the 1st event) • +8 - part of CSPAD array (in event) • +16 - output image shape (in event)

See also Examples for package pyimgalgos.

Module pyimgalgos.image_crop

This module gets from evt store the 2-d image numpy array identified by source and key_in, crop it using range of row and column parameters, and saves cropped 2-d image numpy array in the evt store with unique key_out.

parameter	default value	type	description
source	*-*|Cspad-*	string	address of Detector.Id:Device.Id, or Detector-Id|Device-Id
key_in	mage_in	string	keyword for input 2-d image numpy array
key_out	image_out	string	keyword for output 2-d image numpy array
rowmin	0	int	row minimal to crop image (dafault = 0 - for full size)
rowmax	-1	int	row maximal to crop image (dafault = -1 - for full size)
colmin	0	int	column minimal to crop image (dafault = 0 - for full size)
colmax	-1	int	column maximal to crop image (dafault = -1 - for full size)
print_bits	1	int	module verbosity: • =0 - print nothing • +1 - input pars (in __init__) • +2 - input image parameters (in the 1st event)

Module pyimgalgos.image_save_in_file

This module gets from evt store 2-d image numpy array for specified unique name in key_in and saves it in the file with name given by parameter ofname. File extension defines the output file format. Experiment, run, and event numbers are added to the name of the output file. For example, for ofname = image.tiff files will be created with names image-<experinent>-r####-ev######.tiff, where symbols # stands for numbers.

parameter	default value	type	description
source	*-*|Cspad-*	string	address of Detector-Id|Device-Id
key_in	image	string	unique keyword for input 2-d image numpy array
ofname	/roi-img	string	output file name (type is selected by extension) supported formats: txt, tiff, gif, pdf, eps, png, jpg, jpeg, npy (default), npz
mode	0	int	0-save one event per event, >0-length of the ring buffer (or round robin) for event browser
delay_sec	0	int	additional sleep time in sec between events for event browser
print_bits	1	int	module verbosity: • =0 - print nothing • +1 - input pars (in __init__) • +4 - info about saved file • +8 - part of the image array (in event) • +16-info about saving of files in the ring buffer

In the event browser mode, mode>0, this module saves images in the files with names from the ring buffer. That images can be seen by the application plims, see command options:

> *% plims -h*

See also Examples for package pyimgalgos.

Module pyimgalgos.tahometer

Is intended to print records showing job performance current and integrated from the beginning of job:

```
pyimgalgos.tahometer: run:0049  evt:000005  t[sec]:     2.575  dt[sec]:     2.575  n/t[1/sec]:     1.942  dn/dt
[1/sec]:     1.942
```

parameter	default value	type	description
dn	100	int	interval in number of events to print current statistics
print_bits	1	int	module verbosity: • =0 - print nothing • +1 - input pars (once in __init__) • +2 - current event statistics (in event and endJob)

See also Examples for package pyimgalgos.

Module pyimgalgos.ex_peaks_nda

This module is an example of how to get from the evt store the 2-d numpy array of shape=[Npeaks,12] with peak info produced by ImgAlgos.CSPadArrPeakFinder.

The numpy array is identified by source and key_in. If the peak numpy array is available in the event it will be printed.

parameter	default value	type	description
source	DetInfo(:Cspad.)	string	address of Detector.Id:Device.Id, or Detector-Id|Device-Id
key_in	peaks_nda	string	keyword for input 2-d numpy array
print_bits	1	int	module verbosity: • =0 - print nothing • +1 - input pars (in __init__)

See also Example for Module ImgAlgos::CSPadArrPeakAnalysis.

Package Translator

The translator package include the H5Output module which translates xtc to hdf5. For more information see the page Outdated: The XTC to HDF5 Translator

Package psana_test

The psana_test package includes the psana module dump, some Python library code for testing, and a command line tool for providing a line oriented dump of xtc files. The psana_test package is primarily for psana developers to do software testing, however the dump module and xtclinedump can be generally useful to users.

module dump

The dump module will take a standard psana datasource and dump all the event, config, and epics data found. The entire contents of large arrays are not printed. However a checksum over all the array data is, as well as the min, 25th percentile, median, 75th percentile, and max over the data. The dump module does not serve as a good example of how to retrieve and work with objects from the event store – see the psana_examples package for this.

Running psana_test dump

An example of running the module is

```
psana -n 2 -m psana_test.dump exp=xpptut13:run=179
```

This dumps the first two events of run 179 of the xpp tutorial data.

Understanding psana_test dump output

Below we annotate the output that psana_test.dump can produce. All annotations are preceded by a #

```
==================
=== begin job ====
# first the epics aliases are printed during begin job.
Epics Aliases: total = 240
  Be_xpos
  Be_ypos
  Be_zpos
  ...
# next the epics pv, as they appear during beginJob.
# This corresponds to the xtc configure transition.
# At this point, these are ctrl pvs.
Epics PV
  pvName=HX2:DVD:GCC:01:PMON  pvid=106 dbrtype=34 isCtrl=1 pvName=HX2:DVD:GCC:01:PMON numElements=1 status=0
severity=0 units=T upper_disp_limit=1.0000e-02 lower_disp_limit=0.0000e+00 upper_alarm_limit=0.0000e+00
upper_warning_limit=0.0000e+00 lower_warning_limit=0.0000e+00 lower_alarm_limit=0.0000e+00 upper_ctrl_limit=1.
0000e-02 lower_ctrl_limit=0.0000e+00 data=5.0000e-09
  ...
# After epics, we get the content of the psana env configStore.
# The dump module is getting all keys from the configStore(),
# then retrieving each object. Objects that have a xtc type id, or are
# an numpy array will be printed.

# For each object, we first get a string describing the even key:

type=psana.ControlData.ConfigV2, src=ProcInfo(0.0.0.0, pid=7670)

  # then we get the data of the object.
  # most all data for a Psana object is obtained through accessor methods.
  # methods that return unsigned ints print in hex.
  # methods that return signed ints print in decimal.
  # methods that return floats print in scientific format with 4 decimals of precision.

  npvControls: 0x1
  npvMonitors: 0x0
  npvLabels: 0x0
  events: 0x1E0
  uses_duration: 0x0
  uses_events: 0x1
```

```
# methods that return a compound type, like duration() that returns the compound type time
# are printed as follows:
  duration:
    nanoseconds: 0x0
    seconds: 0x0

# some methods return a python list. Each element in the list is printed separately:
  pvControls[0]:
    name: las_lensh
    index: 0xFFFFFFFF
    value: 0.0000e+00
    array: 0x0
  ...

# epics config is not dumped, see the EPICS alias list for how Psana stores this data
type=psana.Epics.ConfigV1, src=DetInfo(EpicsArch.0:NoDevice.0)
  epicsConfig not dumped

# cspad config is an example which has methods that return ndarrays
type=psana.CsPad.ConfigV4, src=DetInfo(XppGon.0:Cspad.0)
  ...
  numSect: 0x20
  # some methods return a list of simple types, these are printed in one line
  roiMask [0]=0xFF [1]=0xFF [2]=0xFF [3]=0xFF
  numAsicsStored [0]=0x10 [1]=0x10 [2]=0x10 [3]=0x10
  ...
  quads[0]:
    ...
    dp:
      # for an ndarray, we print the type, dimensions, adler32 checksum, and quartile
      # statistics (min, 25th percentile, median, 75th percentile, and max):
      pots: ndarray_uint8_1: dim=[ 80 ] adler32=0x231B31F5 min=0x0 25th=0x3F median=0xB0 75th=0xFF max=0xFF
    gm:
      gainMap: ndarray_uint16_2: dim=[ 185 x 194 ] adler32=0x18730001 min=0x0 25th=0x0 median=0x0 75th=0x0
max=0x0
   ...

============================================================
=== beginrun 0 ===            # typically, there is nothing new in beginrun
============================================================
=== begincalibcycle run=0 step=0 ===

# having dumped the entire initial contents of epics and the config store,
# the dump module will now only print changes to epics or the config.
# That is it remembers how each epics pv and config object printed the last
# time it saw it. With each new transition, it looks at all the epics pv and
# config objects. If any change, they are dumped.

# The control data changed in the calib cycle:

type=psana.ControlData.ConfigV2, src=ProcInfo(0.0.0.0, pid=7670)
  npvControls: 0x1
  npvMonitors: 0x0
  npvLabels: 0x0
  events: 0x1E0
  uses_duration: 0x0
  uses_events: 0x1
  duration:
    nanoseconds: 0x0
    seconds: 0x0
  pvControls[0]:
    name: las_lensh
    index: 0xFFFFFFFF
    value: -4.9997e-01
    array: 0x0

# next we see event data, printing the following:
============================================================
=== event: run=0 step=0 event=0 seconds= 1362889345 nanoseconds= 770371931 fiducials= 19593
# at this point, all epics pv's are replaced with TIME pv's, not the stamp.sec, stamp.nsec below:
```

```
Epics PV
  pvName=HX2:DVD:GCC:01:PMON  pvid=106 dbrtype=20 isTime=1 numElements=1 status=0 severity=0 stamp.
sec=731737344 stamp.nsec=134374000 data=5.2000e-09
  ...
# while most epics pv's have one value, there are some with more than one. If a EPICS pv has less than 20
values, they are all printed, otherwise the typical ndarray summary is printed.

# now we get into regular event data
type=psana.EvrData.DataV3, src=DetInfo(NoDetector.0:Evr.0)
  numFifoEvents: 0x2
  fifoEvents[0]:
    timestampHigh: 0x4C89
    timestampLow: 0x32A6
    eventCode: 0x29
  fifoEvents[1]:
    timestampHigh: 0x4C89
    timestampLow: 0x2E4C
    eventCode: 0x8C
type=psana.CsPad.DataV2, src=DetInfo(XppGon.0:Cspad.0)
  quads[0]:
    seq_count: 0x1
    ticks: 0x329D
    fiducials: 0x4C89
    sb_temp: ndarray_uint16_1: dim=[ 4 ] adler32=0x4FC00AC min=0x7 25th=0x8 median=0x9 75th=0x291 max=0x291
    frame_type: 0x4
    data: ndarray_int16_3: dim=[ 8 x 185 x 388 ] adler32=0xAD5ACF7F min=0 25th=1281 median=1346 75th=1475
max=16383
    virtual_channel: 0x0
    lane: 0x0
    tid: 0x0
    acq_count: 0x85
    op_code: 0x85
    quad: 0x0
    sectionMask: 0xFF
  quads[1]:
    seq_count: 0x1
    ticks: 0x329D
    fiducials: 0x4C89
    sb_temp: ndarray_uint16_1: dim=[ 4 ] adler32=0x2DB006A min=0x7 25th=0xA median=0xA 75th=0x34B max=0x34B
    frame_type: 0x4
    data: ndarray_int16_3: dim=[ 8 x 185 x 388 ] adler32=0xD28441BE min=0 25th=1316 median=1374 75th=1504
max=16383
    virtual_channel: 0x0
    lane: 0x0
    tid: 0x0
    acq_count: 0x85
    op_code: 0x85
    quad: 0x1
    sectionMask: 0xFF
```

src aliases

If a source alias has been defined, it will show up when the event key is printed:

```
type=psana.Pimax.FrameV1, src=DetInfo(AmoEndstation.0:Pimax.0) alias=pimax
```

Options

Several options allow you to control the output of psana_test.dump

The most useful are

```
include = term1 term2
exclude = term1 term2
```

These are used to filter the key strings. For example, running

```
psana -m psana_test.dump -o psana_test.dump.include=10k exp=mob30114:run=145
```

Would only dump event keys that had 10k in them, effectively giving you only Epix::Config10KV1 and psana.Epix.ElementV1 since these are the only types coming from the source DetInfo(NoDetector.0:Epix10k.0).

Other options one could set are:

```
epics = False       do not print epics
aliases = False     do not print the EPICS alias list
dump_aliases=True   follow EPICS aliases to print the EPICS pv's they point to
dump_sml=True       dump the small data type (if found, psana should automatically replace these proxies)
regress_dump=True   do not print the DAQ assigned pvId when printing EPICS
dump_beginjob_evt=False   do not dump begin job data
output_file = filename   write output to filename
config = False      do not print the contents of the configStore, only regular event data
counter = False     do not print the counter string that labels event numbers and calib cycle numbers
header = False
indent = 4          change the indent from the default of 2 to 4
```

Library Usage

Two functions are provided in the Python psana_test package that allow Python scripts to turn Psana objects into strings. A Python script could include the following function to build dictionaries describing the state of the event, configStore, and epicsStore:

```
from psana_test import obj2str, epicsPvToStr

def getPsanaState(event, configStore, epicsStore):
    evtDict = {}
    cfgDict = {}
    epicsDict = {}
    for key in event.keys():
        if key.type() is None: continue
        obj = event.get(key.type(), key.src(), key.key())
        if (obj is None): continue
        if not hasattr(obj,'TypeId'):  continue
        evtDict[str(key)]=obj2str(obj)
    for key in configStore.keys():
        if key.type() is None: continue
        obj = configStore.get(key.type(), key.src())
        if (obj is None): continue
        if not hasattr(obj,'TypeId'):  continue
        cfgDict[str(key)]=obj2str(obj)
    for pvName in epicsStore.pvNames():
        pv = epicsStore.getPV(pvName)
        if not pv: continue
        epicsDict[pvName] = epicsPvToStr(pv)
    return evtDict, cfgDict, epicsDict
```

xtclinedump

xtclinedump is a command line tool to dump xtc and datagram header information in a line oriented style. By keeping the output for each header to one line, it makes it easy to use grep to filter the output. The command can be run by

```
xtclinedump dg xtcfile.xtc
or
xtclinedump xtc xtcfile.xtc
```

The first just dumps datagram headers, the latter dumps xtc headers. There are some additional options, how much of the xtc payloads to print, and if you want parsed output for epics. A help string is available by typing xtclinedump with no arguments. Except for the non-default epics argument, xtclinedump does no parsing of the xtc payloads, it simply prints the first few bytes in hex. For reading through payloads, the intel architechture uses little endian, so 0x00040000 = 1024,

References

- Psana User Manual - Old
- Psana Reference Manual - Old
- psana - Module Examples
- Migration from pyana to psana
- psana - Migration from pyana

```
dump_sml            dump small data type
```