

# Devices and Datatypes

## Devices and datatypes used by the LCLS instruments

Here's a brief intro to: How to find the right data in the xtc/HDF5 files? What datatypes come out of which devices?

A limited number of devices are in use in LCLS, but new ones occasionally gets added, so if you see anything missing, let me know, and please see the [Pyana](#) or [Psana](#) for the most up to date information. You can also take a look at (the possibly outdated) [pdsdata](#) reference manual.

Basic code snippets are given below for how to access this data in pyana and psana. For more elaborate examples, see [Pyana user examples](#), [Psana User Examples - Old](#) and the packages `pyana_examples` and `psana_examples`.

---

- [Detector Info](#)
- [BldData](#), beam line data
- [ControlData](#)
- [EPICS process variables](#)
- [EvrData](#), event receiver data

### Detector Info

The following [devices](#) are used in [these detector locations](#) and produce data classified as "DetInfo", detector info.

The "address" of a particular datatype in the XTC files are typically given contain the "detector" and "device" names and ID. For more details, see e.g. [Psana User Manual -> Data Source Address](#) .

## Acqiris waveform digitizer

Data types: AcqConfig, AcqWaveform

>>

Psana code fragments

Include data description for acqiris:

```
#include "psddl_psana/acqiris.ddl.h"
```

Get the configuration object in the beginCalibCycle member function:

```
Source src = configStr("source", "DetInfo(:Acqiris)");
shared_ptr<Psana::Acqiris::ConfigV1> acqConfig = env.configStore().get(src, &m_src);
```

In the event member function, get the waveform for this event:

```
Pds::Src src;
shared_ptr<Psana::Acqiris::DataDescV1> acqData = evt.get(m_src);
if (acqData.get()) {
    // find matching config object
    shared_ptr<Psana::Acqiris::ConfigV1> acqConfig = env.configStore().get(m_src);
    // loop over channels
    int nchan = acqData->data_shape()[0];
    for (int chan = 0; chan < nchan; ++ chan) {
        const Psana::Acqiris::DataDescV1Elem& elem = acqData->data(chan);
        const ndarray<Psana::Acqiris::TimestampV1, 1>& timestamps = elem.timestamp();
        const ndarray<int16_t, 2>& waveforms = elem.waveforms();
    }
}
```

>>

Pyana code fragments

```
from pypdsdata.xtc import TypeId

def __init__( self, addr = "" ):
    self.addr = addr

def beginjob(self, evt, env):
    config = env.getAcqConfig(self.addr)
    self.channels = config.nbrChannels()

def event(self, evt, env):
    for ch in self.channels:
        acqData = evt.getAcqValue(self.addr, ch, env)
        time_wf = acqData.timestamps()
        volt_wf = acqData.waveform()
```

## IPIMB (Intensity Position and Intensity Monitoring Board)

Data types: IpimbConfig, IpmbFexConfig, IpimbData, IpmbFex

>>

Psana code fragments

Include the data description for IPIMBs

```
#include "psddl_psana/ipimb.ddl.h"
```

In the beginCalibCycle member function, get the configuration object for the IPIMB:

```
Source m_src = configStr("source", "DetInfo(:Ipimb)");  
shared_ptr<Psana::Ipimb::ConfigV1> config1 = env.configStore().get(m_src);
```

In the event member function, get the event data:

```
shared_ptr<Psana::Ipimb::DataV1> data1 = evt.get(m_src);
```

>>

Pyana code fragments

```
from pypdsdata.xtc import TypeId  
  
def __init__( self, addr = "" ):  
    self.addr = addr  
  
def beginjob(self, evt, env):  
    config = env.getConfig( xtc.TypeId.Type.Id_IpimbConfig , self.addr )  
  
def event(self, evt, env):  
    ipm_raw = evt.get(xtc.TypeId.Type.Id_IpimbData, self.addr )  
    ipm_fex = evt.get(xtc.TypeId.Type.Id_IpmFex, self.addr )
```

## Encoder delay scanner

Data types: EncoderConfig, EncoderData

>>

Psana code fragments

Include the data description for Encoder

```
#include "psddl_psana/encoder.ddl.h"
```

In the beginCalibCycle member function, get the configuration object:

```
Source m_src = configStr("source", "DetInfo(:Encoder)");  
shared_ptr<Psana::Encoder::ConfigV1> config1 = env.configStore().get(m_src);
```

In the event member function, get the event data:

```
shared_ptr<Psana::Encoder::DataV1> data1 = evt.get(m_src);
```

>>

Pyana code fragments

```
from pypdsdata.xtc import TypeId  
  
def __init__( self, addr = "" ):  
    self.addr = addr  
  
def beginjob(self, evt, env):  
    config = env.getConfig( xtc.TypeId.Type.Id_EncoderConfig , self.addr )  
    self.channel = None  
    try: # only for ConfigV2 and higher  
        self.channel = 0  
        while ( (config._chan_mask & (1<<self.channel==0 ))):  
            self.channel+=1  
    except:  
        pass  
  
def event(self, evt, env):  
    data = evt.get(xtc.TypeId.Type.Id_EncoderData, self.addr )  
    if self.channel is not None:  
        value = encoder.value( self.channel )  
    else :  
        value = encoder.value() )
```

## Opal1000 camera

Data types: Opal1kConfig, Frame

>>

Psana code fragments

Include the data description for Encoder

```
#include "psddl_psana/opal1k.ddl.h"
```

In the beginCalibCycle member function, get the configuration object:

```
Source m_src = configStr("source", "DetInfo(:Opal1000)");  
shared_ptr<Psana::Opal1k::ConfigV1> config1 = env.configStore().get(m_src);
```

In the event member function, get the event data:

```
shared_ptr<Psana::Opal1k::DataV1> data1 = evt.get(m_src);
```

>>

Pyana code fragments

```
from pypdsdata.xtc import TypeId  
  
def __init__ ( self, addr = "" ):  
    self.addr = addr  
  
def beginjob(self, evt, env):  
    config = env.getConfig(TypeId.Type.Id_Opal1kConfig, self.addr )  
  
def event(self, evt, env):  
    frame = evt.get(TypeId.Type.Id_Frame, self.addr )  
    image = frame.data()
```

## Pulnix camera

Data types: TM6740Config, Frame

>>

Psana code fragments

>>

Pyana code fragments

```
from pypdsdata.xtc import TypeId

def __init__( self, addr = "" ):
    self.addr = addr

def beginjob(self, evt, env):
    config = env.getConfig(TypeId.Type.Id_TM6740Config, self.addr )

def event(self, evt, env):
    frame = evt.get(TypeId.Type.Id_Frame, self.addr )
    image = frame.data()
```

## Princeton Instruments camera

Data types: PrincetonConfig, PrincetonFrame

>>

Psana code fragments

```
#include "psddl_psana/princeton.ddl.h"
```

Get the configuration object in beginCalibCycle method (ConfigV1 or ConfigV2 or later depends on when your data was taken):

```
shared_ptr<Psana::Princeton::ConfigV2> config = env.configStore().get("DetInfo(:Princeton)");
```

Get the image in the event method (FrameV1 or FrameV2 or later depends on when your data was taken):

```
shared_ptr<Psana::Princeton::FrameV2> frame = evt.get("DetInfo(:Princeton)");
const ndarray<uint16_t, 2>& data = frame->data();
// data is now a 2d array of 16-bit integers. Access it's elements in the usual array fashion: data[i][j]
```

>>

Pyana code fragments

```
from pypdsdata.xtc import TypeId

def __init__( self, addr = "" ):
    self.addr = addr

def beginjob(self, evt, env):
    config = env.getConfig(TypeId.Type.Id_PrincetonConfig, self.addr )

def event(self, evt, env):
    frame = evt.get(TypeId.Type.Id_PrincetonFrame, self.addr )
    image = frame.data()
```

## FCCD (Fast CCD) camera

Data types: FccdConfig, Frame

>>

Psana code fragments

>>

Pyana code fragments

```
from pypdsdata.xtc import TypeId

def __init__ ( self, addr = "" ):
    self.addr = addr

def beginjob(self, evt, env):
    config = env.getConfig(TypeId.Type.Id_FccdConfig, self.addr )

def event(self, evt, env):
    frame = evt.get(TypeId.Type.Id_Frame, self.addr )
    image = frame.data()

    # convert to 16-bit integer
    image.dtype = np.uint16
```

## PnCCD camera

Data types: pnCCDconfig, pnCCDframe

>>

Psana code fragments

>>

Pyana code fragments

```
from pypdsdata.xtc import TypeId

def __init__ ( self, addr = "" ):
    self.addr = addr

def beginjob(self, evt, env):
    config = env.getConfig(TypeId.Type.Id_pnCCDconfig, self.addr )

def event(self, evt, env):
    frame = evt.get(TypeId.Type.Id_pnCCDframe, self.addr )
    image = frame.data()
```



## CSPad (Cornell-SLAC Pixel Array Detector)

Data types: CspadConfig, CspadElement

>>

Psana code fragments

>>

Pyana code fragments

```
from pypdsdata.xtc import TypeId

def __init__ ( self, addr = "" ):
    self.addr = addr

def beginjob(self, evt, env):
    config = env.getConfig(TypeId.Type.Id_CspadConfig, self.addr )

    self.sections = map(config.sections, range(4))
    # keep this list of sections in use by each quad

def event(self, evt, env):
    elements = evt.get(TypeId.Type.Id_CspadElement, self.addr )
    # elements is a python list of CspadElement objects

    # To fully populate a 4x8x185x388 array (standard used by pedestal files),
    # create array of zeros and fill in what we have
    pixel_array = np.zeros((4,8,185,388), dtype="uint16")
    for e in elements :
        data = e.data()          # 8x185x388
        quad = e.quad()          # integer
        for n,s in enumerate(self.sections[quad]) :
            pixel_array[quad,s] = data[n]
```

To plot, take a look at examples in XtcExplorer for now.

### Mini CSPad (Cornell-SLAC Pixel Array Detector)

Data types: CspadConfig, Cspad2x2Element

>>

Psana code fragments

>>

Pyana code fragments

```
from pypdsdata.xtc import TypeId

def __init__ ( self, addr = "" ):
    self.addr = addr

def beginjob(self, evt, env):
    config = env.getConfig(TypeId.Type.Id_CspadConfig, self.addr )

def event(self, evt, env):
    element = evt.get(TypeId.Type.Id_Cspad2x2Element, self.addr )
    data = element.data()
    # data is now a 3D array: 185x388x2
```

### Timepix camera

Data types: TimepixConfig, TimepixData

>>

Psana code fragments

>>

Pyana code fragments

```
from pypdsdata.xtc import TypeId

def __init__ ( self, addr = "" ):
    self.addr = addr

def beginjob(self, evt, env):
    config = env.getConfig(TypeId.Type.Id_TimepixConfig, self.addr )

def event(self, evt, env):
    frame = evt.get(TypeId.Type.Id_TimepixData, self.addr )
    image = frame.data()
```

BldData, beam line data

## Beam Energy

Data types: BldDataEBeam, BldDataEBeam0

>>

Psana code fragments

>>

Pyana code fragment

```
def __init__ ( self ):
    pass

def beginjob(self, evt, env):
    pass

def event(self, evt, env):
    ebeam = evt.getEBeam()
    if ebeam is not None:
        beamDmgM = ebeam.uDamageMask
        beamChrg = ebeam.fEbeamCharge
        beamEnrg = ebeam.fEbeamL3Energy
        beamPosX = ebeam.fEbeamLTUPosX
        beamPosY = ebeam.fEbeamLTUPosY
        beamAngX = ebeam.fEbeamLTUAngX
        beamAngY = ebeam.fEbeamLTUAngY
```

## Gas Detector

Data types: FEEGasDetector

>>

Psana code fragments

>>

Pyana code fragments

```
def __init__ ( self ):
    pass

def beginjob(self, evt, env):
    pass

def event(self, evt, env):
    fee_energy_array = evt.getFeeGasDet()
    # an array of 4 energy measurements
```

## Phase Cavity

Data types: PhaseCavity

>>

Psana code fragments

```
#include "psddl_psana/bld.ddl.h"

ClassName::ClassName(const std::string& name)
: Module(name)
{
    m_cavSrc = configStr("phaseCavSource", "BldInfo(PhaseCavity)");
}

void
ClassName::event(Event& evt, Env& env)
{
    shared_ptr<Psana::Bld::BldDataPhaseCavity> myPhaseCavity = evt.get(m_cavSrc);
    if (myPhaseCavity.get()) {
        WithMsgLog(name(), info, str) {
            str << "Bld::BldDataPhaseCavity:"
                << "\n  fitTime1=" << myPhaseCavity->fitTime1()
                << "\n  fitTime2=" << myPhaseCavity->fitTime2()
                << "\n  charge1=" << myPhaseCavity->charge1()
                << "\n  charge2=" << myPhaseCavity->charge2();
        }
    }
}
```

>>

Pyana code fragments

```
def __init__ ( self ):
    pass

def beginjob(self, evt, env):
    pass

def event(self, evt, env):
    pc = evt.getPhaseCavity()
    if pc is not None:
        fitTime1 = pc.fFitTime1
        fitTime2 = pc.fFitTime2
        charge1 = pc.fCharge1
        charge2 = pc.fCharge2
```

## Shared IPIMB

Data types: SharedIpimb, BldDataIpm

>>

Psana code fragments

>>

Pyana code fragments

```
def __init__ ( self, addr = "" ):
    self.addr = addr

def beginjob(self, evt, env):
    pass

def event(self, evt, env):
    ipm = evt.get(xtc.TypeId.Type.Id_SharedIpimb, self.addr )
    if ipm is not None:

        # raw counts
        ch0 = ipm.ipimbData.channel0Volts()
        ch1 = ipm.ipimbData.channel1Volts()
        ch2 = ipm.ipimbData.channel2Volts()
        ch3 = ipm.ipimbData.channel3Volts()

        # fex values and derived quantities
        channels = ipm.ipmFexData.channel
        sum = ipm.ipmFexData.sum
        xpos = ipm.ipmFexData.xpos
        ypos = ipm.ipmFexData.ypos
```

## ControlData

ControlPV, MonitorPV

## EPICS process variables

Data types:

>>

Psana code fragments

>>

Pyana code fragments

```
def __init__ ( self, addr = "" ):
    self.addr = addr

def beginjob(self, evt, env):
    pass

def event(self, evt, env):
    pass
```

EvrData, event receiver data

**Event Receiver**

Data types: EvrConfig, EvrData

>>

Psana code fragments

```
#include "psddl_psana/evr.ddl.h"
void MyClass::beginCalibCycle(Event& evt, Env& env)
{
    shared_ptr<Psana::EvrData::ConfigV1> config = env.configStore().get(source);
    if (config.get()){
        // do something with the config object here (see examples in psana_examples/src/DumpEvt.cpp)
    }
}

void MyClass::event(Event& evt, Env& env)
{
    shared_ptr<Psana::EvrData::DataV3> data3 = evt.get(m_evrSrc);
    if (data3.get()) {
        ndarray<Psana::EvrData::FIFOEvent, 1> fifoEvents = data3->fifoEvents();
        for (int i(0); i<data3->numFifoEvents(); i++){
            evtCode = fifoEvents[i].eventCode();
            tsLow = fifoEvents[i].timestampLow();
            tsHigh = fifoEvents[i].timestampHigh();
        }
    }
}
```

>>

Pyana code fragments

```
import pyana
from pypdsdata import xtc

def __init__ ( self, addr = "" ):
    self.addr = addr

def beginjob(self, evt, env):
    pass

def event(self, evt, env):

    data = evt.get(xtc.TypeId.Type.Id_EvrData, "NoDetector-0|Evr-0")
    if data:
        # print info to screen:
        print " numFifoEvents =", data.numFifoEvents()
        for i in range(data.numFifoEvents()):
            f = data.fifoEvent(i)
            print "    fifo event #%d TimestampHigh=%d TimestampLow=%d EventCode=%d" % \
                (i, f.TimestampHigh, f.TimestampLow, f.EventCode)

        # filter on event codes:
        # get a list of all event codes in this event
        event_codes = [ data.fifoEvent(i).EventCode
                        for i in range(0,data.numFifoEvents()) ]
        # skip event if we find code 162
        if 162 in event_codes :
            print "Found EventCode 162, skipping this event!"
            return pyana.Skip
```

