

Detector Calibration Store

Content

- [Content](#)
- [Purpose](#)
- [Requirements](#)
- [Architecture](#)
 - [Calibration directory and files](#)
 - [Repository](#)
 - [Experiment calibration directory](#)
 - [Local calibration directory](#)
 - [Direct path to calibration file](#)
 - [Structure of calibration directory](#)
 - [Calibration file structure](#)
 - [Calibration data hierarchical scheme](#)
 - [Schema features](#)
 - [Metadata in dictpars](#)
 - [Detector metadata](#)
 - [Calibration parameters' metadata](#)
 - [n-d array metadata](#)
- [API](#)
 - [Parameters](#)
 - [Initialization](#)
 - [UI access methods](#)
 - [Base class DCBase](#)
 - [Class DCStore](#)
 - [Class DCType](#)
 - [Class DCRange](#)
 - [Class DCVersion](#)
- [TBD](#)
- [2016-10-27 Mtg with cpo](#)
- [References](#)

Purpose

- short-term goal - support calibration data for detectors moving across LCLS experiments.
- long-term goal - calibration store for LCLS-II.

Hardware configuration of modern detector can be associated with unique index hardwired on detector controller chip. This index will be used to recognize detector and extract hardly retrievable calibration parameters for geometry, gain factors, masks etc. in case of transition of the detector from one experiment to another.

Requirements

- **Instrument / Experiment independent** - detector can be moved between experiments, hatches, and instruments.
- **Portability** - calibration data for particular detector should be portable as a self consistent file or (part of) calibration directory.
- **Interfaces** - API, Command Line (CL), GUI for central management; management can ONLY be done through the API/CL/GUI in order to prevent adding/removing unknown files without history.
- **Time stamp** (sec) - is THE ONLY value for validity range check; the same rules are applied to the time stamp like to the run number in current calibration system.
- **File name** - the <detname> in the calibration store is <dettype>-<detid>.
- **Aliases** to the calibration file names and specific data can be used to simplify access.
- **Versions** - support versions of calibration data.
- **Links to predecessor and successor** - support access to predecessor and successor if this info is available.

Architecture

To accommodate requirements Detector Calibration Store (DCS) suppose to be implemented as a combination of file system (FS) and data base (DB) files (hdf5 for uniformity). Functionality of the DCS from bottom to top level can be listed as follows.

- All calibration constants for particular detector id (<detid> - index of the detector version) will be kept in a single hdf5 file named as <dettype>-<detid>.h5
 - Schema of this file contains a few levels which account for constant types, time stamp ranges, versions of constants.
 - Beside main functionality each level contains dictionary of parameters and dictionary of history records.
- These files are grouped in the directory for detector type; ex.: cspad, cspad2x2, pnccd, etc.
 - The same directory contains files with aliases <aliases1>.als which map human readable detector name-and-version with appropriate <dettype>-<detid>.h5 file.
- All detector type folders in regular case are collected under calib directory, although it is assumed that calibration files can be used directly.
- The calib directory may be nested in three locations;

- The *repository* and *experimental workspace* have predictable location in the file system, *local work-space* may have an arbitrary path. Direct file access pick up calibration file from any place.

- Two-levels of interaction with DCS:
 - interaction with calibration data in particular calibration file,
 - add/get/remove constants for type, time stamp, version, etc.
 - add/remove/edit metadata.
 - data exchange between files between *repository*, *experimental* and *local work-spaces*.
 - file difference
 - transfer constants for type, time stamp, version, etc.
 - use web-service mechanism
- Access Control Lists (ACL) depends on file location and is assumed to be
 - *repository* - limited to dedicated persons
 - *experimental work-spaces* - members of experimental group
 - *local work-spaces* - all

Repository

instrument and experiment-independent path to calibration directory and file

- /reg/g/psdm/detector/calib/
 <dettype>/<dettype>-<detid>.h5

experiment-dependent path to calibration directory and file

- /reg/d/psdm/<INS>/<experiment>/calib/
 <detttype>/<detttype>-<detid>.h5

local calib path to calibration directory and file

- `<path>/calib/`
`<dettype>/<dettype>-<detid>.h5`

direct path to calibration file

- `<path>/<dettype>-<detid>.h5`

Everything in lower case:

```

<path>/calib/                                # top calib directory
    <dettype>/                                # detector type folders
    cspad2x2/
    pnccd/
    epix100a/
    cspad/
        <files>                                # level of files
        <aliases1>.als                        # file with a map of aliases to detector names
        ...
        <aliasesN>.als                        # one of these files can be used to map "sources" to detector
names
        <dettype>-<detid>.h5                  # files with detector-dependent calibration data
        <dettype>-<detid>.h5

```

- The `calib` directory is a top level DCS directory.
- Detector type `<dettype>` folders are used to organize files under the `calib` directory. In case of direct access to the hdf5 file the detector type is duplicated in the name of the file.
- Unique part of the detector name `<detid>` is used to assign calibration file to the particular detector hardware configuration version. If the detector hardware configuration is changing a new calibration file is created with new `<detid>`.
- Information about predecessor and successor (if available) can be saved under the root level of the calibration file.
- Current sources (ex: `'Camp.0:pnCCD.1'`) could be presented in the dictionary of aliases.

Calibration file structure

Calibration data hierarchical scheme

Access to calibration data is based on time stamp. Time stamp internally is presented in [Unix time](#) sec which can be easily converted forth and back to human readable [format](#) `YYYY-MM-DDTHH:MM:SS+HH:MM`

The schema (structure) of the DCS files `<dettype>-<detid>.h5`

DCStore	DCType	DCRange	DCVersion
/			# top-root-level contains info for the
unique detector version			
dettype (str)			# detector type name, ex: cspad, pnccd, etc
detid (str)			# unique detector id, ex: 01234
tscfile (float)			# time stamp of creation of this structure
predecessor (str)			# name of the previous detector if
available or None			
successor (str)			# name of the next detector if available or
None			
dictpars (dict)			# dictionary of parameters associated with
this version of detector			
history (dict)			# dictionary of history records paired as
(tstamp:record)			
<ctype>/			# folder for calibration type, ex.:
pedestals, rms, mask, etc			
ctype (str)			# calibration type name
dictpars (dict)			# dictionary of parameters associated with
calibration type			
history (dict)			# dictionary of history records paired as
(tstamp:record)			
<tstamp-range>/			# folder for time stamp validity range
<tstamp>[-<tstamp-end>]/			# folder for validity range. If <tstamp-
end> is not specified - then valid to the end			
tsbegin (float)			# time stamp for the beginning of the
validity range			
tsend (float)			# time stamp for the end of the validity
range			
defaultv			# reference to the default calibration,
ex: <vers-tstamp2>			
dictpars (dict)			# dictionary of parameters associated with
validity range			
history (dict)			# dictionary of history records paired as
(tstamp:record)			
<vers-tstamp1>/			# folder for version created on tstamp1
tsvers (float)			# tstamp1 of this version production
calib (ndarray)			# calibration data
history (dict)			# dictionary of history records paired as
(tstamp:record)			
dictpars (dict)			# dictionary of parameters,
			# ex: array size, number of dimensions,
shape, data type, experiment, run, comments, author			
<vers-tstamp2>/			# folder for version created on tstamp2
tsvers (float)			
calib (ndarray)			
history (dict)			
dictpars (dict)			
<tstamp2>[-<tstamp2-end>]/			# folder for the next validity range.
pedestals/			# folder for the next calibration type,
pedestals			
rms/			# folder for the next calibration type, rms

Schema features

- Detector name consists of a common part <dettype> and unique part <detid>.
- Alias to the detector name should be kept in separate dictionary outside the file scheme.
- Each detector may have optional predecessor, successor, and other parameters kept in the dictionary of parameters.
- Calibration type folders contain info about calibrations of particular type, ex: pedestals, rms, status, mask, background, etc.
- Each calibration type contains a set of time stamp ranges defining calibration validity range. If the second time stamp of the range is missing it is considered as infinity (by the end).
- The time stamp range folder contains tstamp_begin, tstamp_end (int) values, dictionary of parameters associated with this folder, reference to the default calibration, and folders with calibration versions, distinguished by there production time stamp.
- The number of calibration versions for each time stamp range is unlimited. Default calibration is defined by the reference defaultv

Metadata in dictpars

Below are the lists of metadata fields which potentially can be used to define detector configuration, calibration parameters etc.

Detector metadata

Field name	Description	More details, example
dettype	detector type	CSPAD, CSPAD2X2, EPIX100A, etc.
detname	detector unique name	(if any) ex.: Camera1
detalias	alias name	if it is hard to memorize the entire name, ex.: 'cspad1'
detidx	detector index	integer number which codes the hardware version
detidxalias	symbolic alias of the index	can be used if it is hard to memorize the index integer number
detcompidx:001	list of component indexes	just in case if we are going to retrieve calibration parameters for separate components
detidxprev	detector index for previous version	detector index for previous version (if available) for the purpose of old calibration search
detidxnext	detector index for next version	detector index for next version (if available) for the purpose of new calibration search
dettsec	time-stamp	time stamp associated with beginning of the validity range for new configuration
detcom:001	comments for this hardware version	as it says { key:comment }
detpar:001	other parameters	just in case if something is forgotten in this table

Calibration parameters' metadata

Field name	Description	More details, example
calibtype	calibration type	ex.: geometry, pixel_status, pixel_gain, pedestals, common_mode, etc.
tsec	time stamp	beginning of the validity range
exp	original experiment	(if available) where calibration constants were obtained
runnum	original run number	(if available) where calibration constants were obtained
runbegin	begin run number	(if available) for validity range
runend	end run number	(if available) for validity range
source	original DAQ data source	data source from DAQ, ex.: 'CxiDs2.0:Cspad.0'
srcalias	data source alias	ex.: 'cspad'
calibvers	version par	in order to access using symbolic name or some alias
calibversalias	version alias	if it is hard to memorize version par
com:001	comments for this version	as it says { key:comment }
par:001	other parameters	just in case if something is forgotten in this table

In case of numpy array their metadata are stored with an object.

Text file needs in n-d array metadata

n-d array metadata

Field name	Description	More details, example
DTYPE	(str) data type	int, float, double, etc.
NDIMS	(int) number of dimensions (N)	ex.: 3
DIM:1	(int) size of dim.1	ex.: 185
DIM:2	(int) size of dim.2	ex.: 388
...
DIM:N	(int) size of dim.N	ex.: 2

API

Parameters

dettype (str) - detector type: cspad, cspad2x2, pnccd, fccd, opal, epix100a, etc.

ctype (str) - calibration type: pedestals, status, rms, mask, gain, bkgd, common_mode, geometry

detid (str) - detector unique id number, ex: 123456

detname (str) - detector unique name, combination of <dettype>-<detid>

detalias (str) - detector alias name assigned to the detname

cpath (str) - path to the calibration directory (ex: cpath='<path>/calib') or direct hdf5 file name (ex: cpath='<path>/<detname>-<detid>.h5').
Default calibration directory CPATH_DEF='/reg/g/psdm/detector/calib'.

version (int) - time stamp of the version creation

versind (int) - consecutive version index assigned/mapped to the version production time stamp.

Initialization

```
# Import
from PSCalib.DCStore import DCStore          # inport DCStore (Detector Calibration Store) object

# Initialization
REPO = '/reg/g/psdm/detector/calib'          # default calib repository

cdir = env.calibDir()                         # '/reg/d/psdm/<INS>/<experiment>/calib' - accept current directory
path = cdir + <dettype>

fname = '[path/]pnccd-12345678'               # standard name includes detector type, dash, and n-digit id number
fname = '[path/]Camera1'                     # alias

cs = DCStore(fname)                           # creates a DCStore object.
""" get calibration store object
    Input parameters:
        fname [str] - file name/alias of the detector
    """
```

- If path is missing - use repository.
- If calibration file is not found - throw raise IOError('File %s is not available' % fname)

UI access methods

```
ctype = pedestals # status, rms, mask, gain, bkgd, common_mode, geometry, etc
tsp = tstamp parameter to identify constants, which can be retrieved from evt.run() - run number, evt
vers = None # for default or versind or version time stamp.

# generic access method:
obj = cs.get(ctype, tsp, vers=None)
```

Base class DCBase

Is reserved to support common methods of all project classes. For now it stands for manipulations with dictpars but not limited to.

```

o = DCBase()

# access methods
dictpars = o.dictpars()           # returns (dictpars) dictionary of dictpars associated with each object
par = o.par(k)                    # returns par value for key k
log = o.history(fmt)              # returns (str) history records preceded by the time stamp (default fmt='%Y-%
m-%dT%H:%M:%S%Z') as a text
d = o.histdict()                  # returns (dict) history dictionary associated with current object

# management methods
o.set_dictpars(d)                 # set (d) dictionary of parameters for object
o.add_par(k,v)                    # add (k,v) par to the dictionary of parameters for object
o.del_dictpars()                  # delete all parameters from the dictionary
o.del_par(k)                      # delete par with key k
o.set_history(d)                  # set (dict) as a history dictionary of the current object
o.add_history(rec, ts)            # add (str) record with (float) time stamp to the history dictionary (ts:
rec). If ts is None - call current time is used as a key.

```

Class DCStore

```

cs = DCStore(cpath, detname)      # (str) path to calib directory or file, (str) detector name

# access methods
nda = cs.get(ctype, tsp, vers)    # (str) ctype - calibration type
                                     # (...) tsp - parameter to get time stamp (evt, runnum, ts_sec)
                                     # (int) vers - version of calibration, None - use default
tscfile = cs.tscfile()           # (int) time stamp of the file creation
dettype = cs.dettype()           # (str) detector type
detid = cs.detid()               # (str) detector id
detname = cs.detname()           # (str) detector name of self object
predecessor = cs.predecessor()   # (str) detname of predecessor or None
successor = cs.successor()       # (str) detname of successor or None
ctypes = cs.ctypes()             # (list) calibration types in the file
cto = cs.ctypeobj(ctype)         # (DCType ~ h5py.Group) calibration type object

# management methods
cs.set_tscfile(ts)               # set (int) time stamp of the file creation
cs.set_dettype(dettype)          # set (str) detector type
cs.set_detid(detid)              # set (str) detector id
cs.set_detname(detname)          # set (str) detector name of self object
cs.set_predecessor(pred)         # set (str) detname of predecessor or None
cs.set_successor(succ)           # set (str) detname of successor or None
cs.add_ctype(ctype)              # add (str) calibration type to the DCStore object
cs.del_ctype(ctype)              # delete ctype (str) from the DCStore object
cs.save(path)                    # save current calibration in the file specified by path, if path is Null -
update current file.

```

Class DCType

```

cto = DCType(dettype)            # (str) detector type

# access methods
ctype = cto.ctype()              # (str) of ctype name
tsranges = cto.ranges()          # (list) of time ranges for ctype
tsro = cto.rangeobj(tsrange)     # (DCRange ~ h5py.Group) time stamp validity range object

# management methods
cto.add_range(tsr)               # add (str) of time ranges for ctype
cto.del_range(tsr)               # delete range from the DCType object

```

Class DCRange

```

tsro = DCRange(tsrange)           # (str) time stamp validity range
tsro = DCRange(tsbegin, tsend)    # (int,int) time stamp validity range

# access methods
tsbegin    = tsro.begin()         # (int) time stamp beginning validity range
tsend      = tsro.end()           # (int) time stamp ending validity range
versions   = tsro.versions()      # (list of float) versions of calibrations
versodef   = tsro.versdef()       # (DCVersion ~ h5py.Group) reference to the default version in the time-
range object
verso      = tsro.versobj(vers)   # (DCVersion ~ h5py.Group) specified version in the time-range object

# management methods
tsro.set_begin(tsbegin)           # set (int) time stamp beginning validity range
tsro.set_end(tsend)               # set (int) time stamp ending validity range
tsro.add_version(vers)            # set (DCVersion ~ h5py.Group) versions of calibrations
tsro.set_versdef(vers)            # set (DCVersion ~ h5py.Group) versions of calibrations
tsro.del_version(vers)            # delete version

```

Class DCVersion

```

verso = DCVersion(vers)           # (str) version name

# access methods
tsvers    = verso.tsprod()        # (int) time stamp of the version production
calibdata = verso.calib()         # (np.array) calibration array

# management methods
verso.set_tsprod(tsprod)          # set (int) time stamp of the version production
verso.add_calib(nda)              # set (np.array) calibration array

```

TBD

Open questions

- logic for default version
 - always last added constants
 - set specified version
 - what to do with default if new version is added?
- when constants are updated (file open to write) they are not available... Lock to resolve.

Data flow

- who produces and supply constants
- who is allowed to submit constants
- who is allowed to access constants
- ACL inside API or using OS
- ACL for all or particular detector/type/ etc.

2016-10-27 Mtg with cpo

- table of aliases with records: <alias> <source> <begin> <end>
- repository /reg/g/psdm/detector/calib "g" or "d"
- epix100a full name - 6 numbers of hardware versions, firmware version is a 7th number
- time double to two integers or int64 (32bit-sec, 32-bit-nsec)
- geometry needs to be saved as a text
- repository file has geometry and pixel_gain - like constants
- do not search in <experiment>/calib
- access through network for AMI (cpo: TCP with hardwired IP addresses, m:web service/reddis)

References

- [Calibration Store](#)
- [Unix time](#)
- [ISO 8601 time format](#)