iCalibrationDB Documentation

Release 1.0

S. Hauf

Jun 16, 2021

Contents

Contents:

CHAPTER 1

Examples & Tests

Examples:

1.1 Example Round-Trip of Calibration Constants

Below is an example of roundtripping example calibration data for an AGIPD detector module

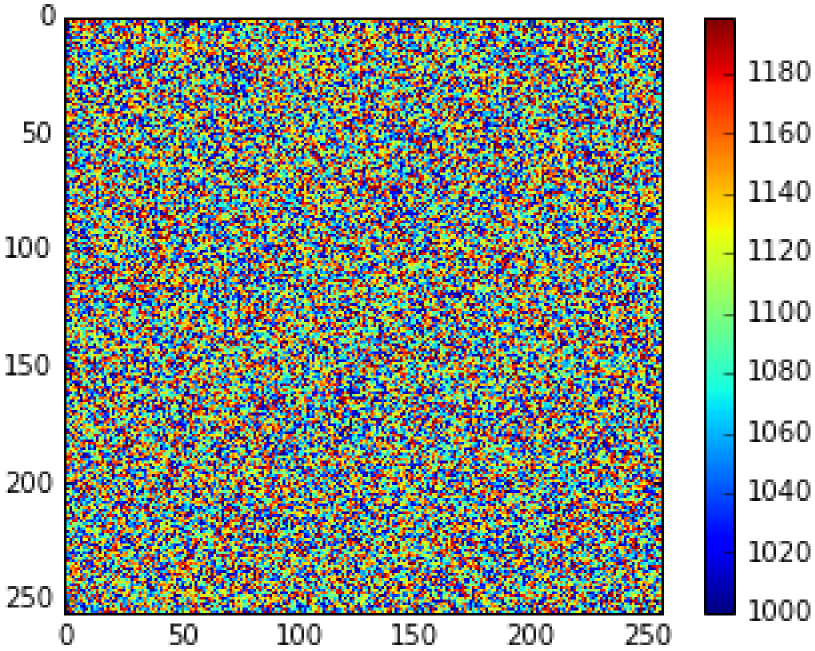
```
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline

from iCalibrationDB import ConstantMetaData, Constants, Conditions, Detectors,␣
↪Versions
```

We first create a fake constant for multi-memory cell data.

```
fake_constant = np.random.randint(1000, 1200, (256,256,32))
fake_constant = fake_constant.astype(np.short)
```

```
fig = plt.figure()
ax = fig.add_subplot(111)
im = ax.imshow(fake_constant[...,0], interpolation="nearest")
cb = fig.colorbar(im)
```



In the following we set up the meta data for this constant. Common constants, operating conditions and versions of these constants are already pre-defined and can be directly used from the library.

For our example we've created an AGIPD offset constant, which was taken at an AGIPD dark condition with 32 memory cells and a bias voltage of 200V. Furthermore we are creating a new version, reflected by using *Versions.Now*. This version is for the *Q1M1* module of the *AGIPD1M1* detector.

```
metadata = ConstantMetaData()

# set the constant
offset = Constants.AGIPD.Offset()
offset.data = fake_constant
metadata.calibration_constant = offset

# set the operating condition
condition = Conditions.Dark.AGIPD(memory_cells=32, bias_voltage=200)
metadata.detector_condition = condition

# specify the a version for this constant
metadata.calibration_constant_version = Versions.Now(device=Detectors.AGIPD1M1.Q1M1)

# this will have auto-assigned a device uuid and device name:
print("Device name is: {}".format(metadata.calibration_constant_version.device_name))
print("Device uuid is: {}".format(metadata.calibration_constant_version.device_uuid))
```

```
Device name is: AGIPD_M001
Device uuid is: 00100001100000
```

As shown above, calibration constants map to detector modules for segmented detectors, or to a single detector (module) for monolithic detectors. These modules are internally identified by a *uuid*, but are also identified by a module name and mapped to their current physical location in a detector instance. The latter may be altered to reflect updates in module usage. Through the combination of name and *uuid* the correct module will still be identified:

```
# these are the same modules, once identified by location, once my name
assert Detectors.AGIPD1M1.Q1M1 is Detectors.AGIPD.M001
assert Detectors.AGIPD1M1.Q1M2 is Detectors.AGIPD.M002

# these are not the same modules:
assert Detectors.AGIPD1M1.Q1M2 is Detectors.AGIPD.M001
```

```
---------------------------------------------------------------------------

AssertionError                            Traceback (most recent call last)

<ipython-input-5-70f3f4000e6d> in <module>()
      4
      5 # these are not the same modules:
----> 6 assert Detectors.AGIPD1M1.Q1M2 is Detectors.AGIPD.M001


AssertionError:
```

For each detector a *repr*esentation is availble to show the current location mapping. You can access it simply by typing the instance name as the last command into a notebook cell or by using *display(Detectors.AGIPD1M1)* explicitly.

```
Detectors.AGIPD1M1
```

```
from IPython.display import display
display(Detectors.FastCCD1)
```

With our meta data complete and having assured ourselves that we are referring to the correct module, we can now send the data to the database. This requires passing the ZMQ address of a running *CalibrationDbRemote* Karabo device, which is accessible to us. It need not be on *localhost*; this is only used in this example.

```
metadata.send("tcp://localhost:5005")
```

```
Converting calibration_constant to dict
Converting detector_condition to dict
Converting calibration_constant_version to dict
Successfully sent constant to database!
```

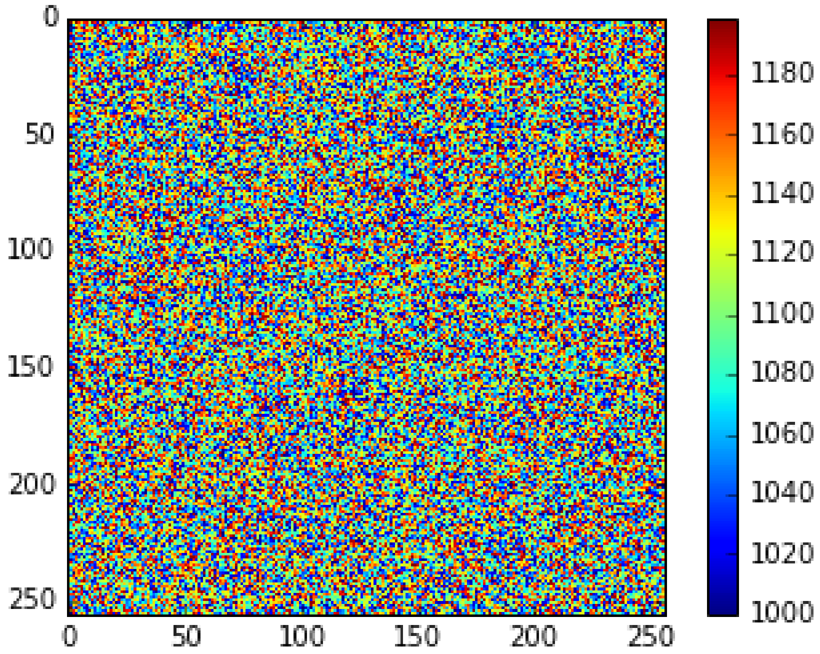
Retrieving data works in a similar fashion. Again we need to pass an address of a *CalibrationDbRemote* device we would like to connect to.

If a constant matching our meta-data is found it is returned and the meta-data is updated to reflect it.

```
metadata.retrieve("tcp://localhost:5005")
```

```
Converting calibration_constant to dict
Converting detector_condition to dict
Converting calibration_constant_version to dict
Successfully retrieved constant from database!
Updating self...
```

```
retrieved_constant = metadata.calibration_constant.data
fig = plt.figure()
ax = fig.add_subplot(111)
im = ax.imshow(retrieved_constant[...,0], interpolation="nearest")
cb = fig.colorbar(im)
```



Since we basically round-tripped the calibration constant in this example the two should be identical. Here we explicitely verify that.

```
assert np.allclose(retrieved_constant, fake_constant)
```

1.2 Current Detector Mappings

This notebook shows the current module to detector mappings, module names and UUIDs currently in use.

```
from IPython.display import display

from iCalibrationDB import Detectors, DetectorSpec, DetectorInstance
```

```
sorted_detectors = sorted([k for k in vars(Detectors).keys()])

for key in sorted_detectors:
    detector = vars(Detectors)[key]
    if isinstance(detector, (DetectorSpec, DetectorInstance)):
        display(detector)
```

CHAPTER 2

iCalibrationDB

2.1 iCalibrationDB package

2.1.1 Subpackages

iCalibrationDB.detector_instances package

Submodules

class iCalibrationDB.detector_instances.agipd.**AGIPDInstance**
Bases: *iCalibrationDB.detectors.DetectorInstance*

class iCalibrationDB.detector_instances.dssc.**DSSCInstance**
Bases: *iCalibrationDB.detectors.DetectorInstance*

class iCalibrationDB.detector_instances.lpd.**LPDInstance**
Bases: *iCalibrationDB.detectors.DetectorInstance*

Module contents

class iCalibrationDB.detector_instances.**Detectors**
Bases: object

AGIPD = <iCalibrationDB.detector_instances.agipd._AGIPD object>

AGIPD1M1 = <iCalibrationDB.detector_instances.agipd._AGIPD1M1 object>

AGIPD1M2 = <iCalibrationDB.detector_instances.agipd._AGIPD1M2 object>

AGIPD500K = <iCalibrationDB.detector_instances.agipd._AGIPD500K object>

DSSC = <iCalibrationDB.detector_instances.dssc._DSSC object>

DSSC1M1 = <iCalibrationDB.detector_instances.dssc._DSSC1M1 object>

Jungfrau_M035

Jungfrau_M039

Jungfrau_M086

Jungfrau_M125

Jungfrau_M203

Jungfrau_M221

Jungfrau_M228

Jungfrau_M233

Jungfrau_M242

Jungfrau_M260

Jungfrau_M263

Jungfrau_M266

Jungfrau_M267

Jungfrau_M273

Jungfrau_M275

Jungfrau_M276

Jungfrau_M285

Jungfrau_M288

LPD = <iCalibrationDB.detector_instances.lpd._LPD object>

LPD1M1 = <iCalibrationDB.detector_instances.lpd._LPD1M1 object>

PnCCD1

ePix100_M15

ePix100_M16

ePix100_M17

ePix100_M18

ePix100_burn1

ePix10K_M40

ePix10K_M41

ePix10K_M43

fastCCD1

pnCCD_M205_M206

iCalibrationDB.tests package

Submodules

class iCalibrationDB.tests.test_calibration_constant.**TestCalibrationConstant**(*methodName='runT*

Bases: unittest.case.TestCase

test_defaults()

test_to_dict()

test_typing()

class iCalibrationDB.tests.test_constant_version.**TestConstantVersion**(*methodName='runTest'*)
Bases: unittest.case.TestCase

test_defaults()

test_time_assignments()

test_to_dict()

class iCalibrationDB.tests.test_detector_condition.**TestDetectorCondition**(*methodName='runTest'*)
Bases: unittest.case.TestCase

test_defaults()

test_setting_parameters()

test_to_dict()

class iCalibrationDB.tests.test_known_constants.**TestKnownConstants**(*methodName='runTest'*)
Bases: unittest.case.TestCase

base_constants = (<class 'iCalibrationDB.known_constants.BadPixels'>, <class 'iCalibra

test_base_constants()

test_detector_constants()

class iCalibrationDB.tests.test_operating_condition.**TestOperatingCondition**(*methodName='runTest*
Bases: unittest.case.TestCase

test_defaults()

test_to_dict()

test_typing()

Module contents

2.1.2 Submodules

class iCalibrationDB.calibration_constant.**CalibrationConstant**
Bases: *iCalibrationDB.util.DictConvertable*

A calibration constant bound to a detector type

auto_approve
If the constant is auto-approved for good quality

data
The constant data itself.

Should be a numpy array, usually of dimensions (x, y, memory cell)

description
A description for the constant

device_type_name
The device type the constant refers to.

Expects an object of type *Detector*.

mandatory = ['flg_auto_approve', 'description', 'calibration_name', 'detector_type_name

name
The name of the constant

class iCalibrationDB.constant_version.**ConstantVersion**
Bases: *iCalibrationDB.util.DictConvertable*

Calibration constant versions reflect evolution over time

begin_at
When the constant was produced/injected

Expects a datetime object, e.g. *datatime.now()*

begin_validity_at
When the validity of the constant begins at.

Expects a datetime object, e.g. *datatime.now()*

ccv_id
Id number of the calibration constant version

description
A description of this constant version

device_name
Name of the device producing the constant

end_validity_at
When the validity of the constant ends at.

Expects a datetime object, e.g. *datatime.now()*

file_name
File name the constant is stored at - will be auto-filled

good_quality
Flag indicating if the constant is of good quality (True)

karabo_da
Name of the Karabo DA to identify the detector producing the constant

karabo_id
Name of the Karabo ID to identify the detector producing the constant

mandatory = ['karabo_id', 'file_name', 'flg_good_quality', 'begin_validity_at', 'end_va

raw_data_location
Location of the raw_data. Will be auto-set.

report_path
Path of the report. Will be auto-set.

retrieve_optionals = ['file_name']

class iCalibrationDB.detector_condition.**DetectorCondition**
Bases: *iCalibrationDB.util.DictConvertable*

Detector condition's group operating conditions and additional meta-data.

available
Internal parameter, should be left at the default (True).

description
A description for this detector condition.

mandatory = ['name', 'flg_available', 'parameters']

name
The name of the detector condition, will be used in data base

parameters
The operating parameters of the detector at this condition.

Expects a list of *OperatingConditions*.

class iCalibrationDB.detectors.**DetectorInstance**
Bases: object

class iCalibrationDB.detectors.**DetectorModule**(*uuid=0*, *version=0*)
Bases: object

detector_type = None

detector_uuid = None

device_name

module_uuid = None

owner = None

type_name

version = None

class iCalibrationDB.detectors.**DetectorSpec**
Bases: object

detector_type = None

class iCalibrationDB.detectors.**DetectorTypes**
Bases: enum.Enum

An enumeration.

AGIPD = 'AGIPD-Type'

DSSC = 'DSSC-Type'

LPD = 'LPD-Type'

ePix100 = 'ePix100-Type'

ePix10K = 'ePix10K-Type'

fastCCD = 'fastCCD-Type'

jungfrau = 'jungfrau-Type'

pnCCD = 'pnCCD-Type'

iCalibrationDB.detectors.**make_detector_instance**(*detector*)

class iCalibrationDB.known_constants.**BadPixels**
Bases: *iCalibrationDB.calibration_constant.CalibrationConstant*

Detector bad pixel map

class iCalibrationDB.known_constants.**Constants**

Bases: `object`

Predefined constants are provided in this class.

They are organized by detector type, and then constant name. This class is filled automatically at module load time. It is thus statically empty on purpose.

Special contants for a given detector are defined explicitly

class AGIPD

Bases: `object`

class BadPixels

Bases: *iCalibrationDB.known_constants.BadPixels*

class BadPixelsCI

Bases: *iCalibrationDB.calibration_constant.CalibrationConstant*

Bad pixels derived from CI runs

class BadPixelsDark

Bases: *iCalibrationDB.calibration_constant.CalibrationConstant*

Bad pixels derived from dark runs

class BadPixelsFF

Bases: *iCalibrationDB.calibration_constant.CalibrationConstant*

Bad pixels derived from FF runs

class BadPixelsPC

Bases: *iCalibrationDB.calibration_constant.CalibrationConstant*

Bad pixels derived from PC runs

class Noise

Bases: *iCalibrationDB.known_constants.Noise*

class Offset

Bases: *iCalibrationDB.known_constants.Offset*

class RelativeGain

Bases: *iCalibrationDB.known_constants.RelativeGain*

class SlopesCI

Bases: *iCalibrationDB.calibration_constant.CalibrationConstant*

Gain slopes derived from CI runs

class SlopesFF

Bases: *iCalibrationDB.calibration_constant.CalibrationConstant*

Gain slopes derived from FF runs

class SlopesPC

Bases: *iCalibrationDB.calibration_constant.CalibrationConstant*

Gain slopes derived from PC runs

class ThresholdsCI

Bases: *iCalibrationDB.calibration_constant.CalibrationConstant*

Gain thresholds derived from CI run

class ThresholdsDark

Bases: *iCalibrationDB.calibration_constant.CalibrationConstant*

Gain thresholds derived from dark images

class ThresholdsPC

Bases: *iCalibrationDB.calibration_constant.CalibrationConstant*

Gain thresholds derived from PC runs

CCD()

class DSSC

Bases: object

class BadPixels

Bases: *iCalibrationDB.known_constants.BadPixels*

class Noise

Bases: *iCalibrationDB.known_constants.Noise*

class Offset

Bases: *iCalibrationDB.known_constants.Offset*

class RelativeGain

Bases: *iCalibrationDB.known_constants.RelativeGain*

class LPD

Bases: object

class BadPixels

Bases: *iCalibrationDB.known_constants.BadPixels*

class BadPixelsCI

Bases: *iCalibrationDB.calibration_constant.CalibrationConstant*

Bad pixels derived from CI runs

class BadPixelsDark

Bases: *iCalibrationDB.calibration_constant.CalibrationConstant*

Bad pixels derived from dark runs

class BadPixelsFF

Bases: *iCalibrationDB.calibration_constant.CalibrationConstant*

Bad pixels derived from FF runs

class FFMap

Bases: *iCalibrationDB.calibration_constant.CalibrationConstant*

Flatfield map

class GainAmpMap

Bases: *iCalibrationDB.calibration_constant.CalibrationConstant*

Gain amplification factor map

class Noise

Bases: *iCalibrationDB.known_constants.Noise*

class Offset

Bases: *iCalibrationDB.known_constants.Offset*

class RelativeGain
Bases: *iCalibrationDB.calibration_constant.CalibrationConstant*

Detector relative gain.

class SlopesCI
Bases: *iCalibrationDB.calibration_constant.CalibrationConstant*

Gain slopes derived from CI runs

class SlopesFF
Bases: *iCalibrationDB.calibration_constant.CalibrationConstant*

Gain slopes derived from FF runs

class ePix100
Bases: `object`

class BadPixels
Bases: *iCalibrationDB.known_constants.BadPixels*

class BadPixelsDark
Bases: *iCalibrationDB.calibration_constant.CalibrationConstant*

Detector Bad pixels

class BadPixelsIlluminated
Bases: *iCalibrationDB.calibration_constant.CalibrationConstant*

Detector Bad pixels Illuminated

class Noise
Bases: *iCalibrationDB.calibration_constant.CalibrationConstant*

Detector noise

class Offset
Bases: *iCalibrationDB.calibration_constant.CalibrationConstant*

Detector offset/pedestal

class RelativeGain
Bases: *iCalibrationDB.calibration_constant.CalibrationConstant*

Detector relative gain

class ePix10K
Bases: `object`

class BadPixels
Bases: *iCalibrationDB.known_constants.BadPixels*

class BadPixelsDark
Bases: *iCalibrationDB.calibration_constant.CalibrationConstant*

Detector Bad pixels

class BadPixelsIlluminated
Bases: *iCalibrationDB.calibration_constant.CalibrationConstant*

Detector Bad pixels Illuminated

class Noise
Bases: *iCalibrationDB.calibration_constant.CalibrationConstant*

Detector noise

class Offset
Bases: *iCalibrationDB.calibration_constant.CalibrationConstant*

Detector offset/pedestal

class RelativeGain
Bases: *iCalibrationDB.calibration_constant.CalibrationConstant*

Detector relative gain

class fastCCD
Bases: object

class BadPixels
Bases: *iCalibrationDB.known_constants.BadPixels*

class Noise
Bases: *iCalibrationDB.known_constants.Noise*

class Offset
Bases: *iCalibrationDB.known_constants.Offset*

class RelativeGain
Bases: *iCalibrationDB.known_constants.RelativeGain*

class jungfrau
Bases: object

class BadPixels
Bases: *iCalibrationDB.known_constants.BadPixels*

class BadPixelsDark
Bases: *iCalibrationDB.calibration_constant.CalibrationConstant*

Detector Bad pixels

class BadPixelsFF
Bases: *iCalibrationDB.calibration_constant.CalibrationConstant*

Bad pixels derived from FF runs

class Noise
Bases: *iCalibrationDB.calibration_constant.CalibrationConstant*

Detector noise

class Offset
Bases: *iCalibrationDB.calibration_constant.CalibrationConstant*

Detector offset/pedestal

class RelativeGain
Bases: *iCalibrationDB.calibration_constant.CalibrationConstant*

Detector relative gain

class pnCCD
Bases: object

class BadPixels
Bases: *iCalibrationDB.known_constants.BadPixels*

class Noise
Bases: *iCalibrationDB.known_constants.Noise*

class Offset

Bases: *iCalibrationDB.known_constants.Offset*

class RelativeGain

Bases: *iCalibrationDB.known_constants.RelativeGain*

class iCalibrationDB.known_constants.**Noise**

Bases: *iCalibrationDB.calibration_constant.CalibrationConstant*

Detector noise

class iCalibrationDB.known_constants.**Offset**

Bases: *iCalibrationDB.calibration_constant.CalibrationConstant*

Detector offset/pedestal

class iCalibrationDB.known_constants.**RelativeGain**

Bases: *iCalibrationDB.calibration_constant.CalibrationConstant*

Detector relative gain

iCalibrationDB.known_constants.**cls**

alias of *iCalibrationDB.known_constants.BadPixels*

iCalibrationDB.known_constants.**constant**

alias of *iCalibrationDB.known_constants.BadPixels*

iCalibrationDB.known_constants.**dcls**

alias of *iCalibrationDB.known_constants.Constants.ePix10K*

iCalibrationDB.known_constants.**meta_init**(*cls*, *detector*, *orig_init*)

class iCalibrationDB.known_detector_conditions.**AGIPDCondition**(*memory_cells*, *bias_voltage*, *pixels_x=512*, *pixels_y=128*, *acquisition_rate=None*, *gain_setting=None*, *gain_mode=None*)

Bases: *iCalibrationDB.known_detector_conditions.BaseDetectorCondition*

Basic dark AGIPD detector condition.

class iCalibrationDB.known_detector_conditions.**AGIPDIlluminatedCondition**(*memory_cells*, *bias_voltage*, *photon_energy*, *pixels_x=512*, *pixels_y=128*, *beam_energy=None*, *acquisition_rate=None*, *gain_setting=None*, *gain_mode=None*)

Bases: *iCalibrationDB.known_detector_conditions.IlluminatedCondition*, *iCalibrationDB.known_detector_conditions.AGIPDCondition*

Illuminated AGIPD detector condition.

class iCalibrationDB.known_detector_conditions.**BaseDetectorCondition**(*memory_cells=1*, *bias_voltage=None*, *pixels_x=None*, *pixels_y=None*)

Bases: *iCalibrationDB.detector_condition.DetectorCondition*

A basic operating condition valid for all semi-conductor detectors

add_operating_condition(*condition*)

Add an operating condition to the detector condition parameter list

class iCalibrationDB.known_detector_conditions.**CCDCondition**(*bias_voltage*, *integration_time*, *gain_setting*, *pixels_x=1024*, *pixels_y=512*, *temperature=291*, *freq_threshold=None*)

Bases: *iCalibrationDB.known_detector_conditions.BaseDetectorCondition*, *iCalibrationDB.known_detector_conditions.CCDMixinCondition*

Basic CCDs detector condition.

class iCalibrationDB.known_detector_conditions.**CCDIlluminatedCondition**(*bias_voltage*, *photon_energy*, *integration_time*, *gain_setting*, *pixels_x=1024*, *pixels_y=512*, *beam_energy=None*, *temperature=291*)

Bases: *iCalibrationDB.known_detector_conditions.IlluminatedCondition*, *iCalibrationDB.known_detector_conditions.CCDMixinCondition*

Illuminated CCDs detector condition.

class iCalibrationDB.known_detector_conditions.**CCDMixinCondition**

Bases: object

class iCalibrationDB.known_detector_conditions.**Conditions**

Bases: object

Predefined detector conditions are grouped in this class.

class Dark

Bases: object

Conditions for non-illuminated detector

AGIPD
alias of *AGIPDCondition*

CCD
alias of *CCDCondition*

DSSC
alias of *DSSCCondition*

LPD
alias of *LPDCondition*

ePix100
alias of *EPix100Condition*

ePix10K
alias of *EPix10KCondition*

jungfrau
alias of *JungfrauCondition*

class Illuminated
Bases: `object`

Conditions for an illuminated detector, which is additionally specified by photon energy in keV and optionally, beam energy in μJ

AGIPD
alias of *AGIPDIlluminatedCondition*

CCD
alias of *CCDIlluminatedCondition*

DSSC
alias of *DSSCCondition*

LPD
alias of *LPDIlluminatedCondition*

ePix100
alias of *EPix100IlluminatedCondition*

ePix10K
alias of *EPix10KIlluminatedCondition*

jungfrau
alias of *JungfrauIlluminatedCondition*

class iCalibrationDB.known_detector_conditions.**DSSCCondition**(*memory_cells*, *bias_voltage*, *pixels_x=512*, *pixels_y=128*, *pulseid_checksum=None*, *acquisition_rate=None*, *target_gain=None*, *encoded_gain=None*)

Bases: *iCalibrationDB.known_detector_conditions.BaseDetectorCondition*

Basic DSSC detector condition.

class iCalibrationDB.known_detector_conditions.**EPix100Condition**(*bias_voltage*, *integration_time*, *in_vacuum*, *pixels_x=708*, *pixels_y=768*, *temperature=288*)

Bases: *iCalibrationDB.known_detector_conditions.BaseDetectorCondition*, *iCalibrationDB.known_detector_conditions.JungfrauMixinCondition*

Basic xPix100 detector condition.

class iCalibrationDB.known_detector_conditions.**EPix100IlluminatedCondition**(*bias_voltage*, *photon_energy*, *integration_time*, *in_vacuum*, *pixels_x=708*, *pixels_y=768*, *beam_energy=None*, *temperature=288*)

Bases: *iCalibrationDB.known_detector_conditions.IlluminatedCondition*, *iCalibrationDB.known_detector_conditions.JungfrauMixinCondition*

Illuminated xPix100 detector condition.

class iCalibrationDB.known_detector_conditions.**EPix10KCondition**(*bias_voltage*, *integration_time*, *in_vacuum*, *pixels_x=356*, *pixels_y=384*, *temperature=253*, *gain_setting=0*)

Bases: *iCalibrationDB.known_detector_conditions.BaseDetectorCondition*, *iCalibrationDB.known_detector_conditions.JungfrauMixinCondition*

Basic xPix100 detector condition.

class iCalibrationDB.known_detector_conditions.**EPix10KIlluminatedCondition**(*bias_voltage*, *photon_energy*, *integration_time*, *in_vacuum*, *pixels_x=356*, *pixels_y=384*, *beam_energy=None*, *temperature=253*, *gain_setting=0*)

Bases: *iCalibrationDB.known_detector_conditions.IlluminatedCondition*, *iCalibrationDB.known_detector_conditions.JungfrauMixinCondition*

Illuminated xPix100 detector condition.

class iCalibrationDB.known_detector_conditions.**IlluminatedCondition**(*photon_energy=None*, *beam_energy=None*, *memory_cells=1*, *bias_voltage=None*, *pixels_x=None*, *pixels_y=None*)

Bases: *iCalibrationDB.known_detector_conditions.BaseDetectorCondition*

A basic operating condition valid for detectors illuminated with photons.

class iCalibrationDB.known_detector_conditions.**JungfrauCondition**(*memory_cells*, *bias_voltage*, *integration_time*, *pixels_x=1024*, *pixels_y=512*, *temperature=291*, *gain_setting=0*)

Bases: *iCalibrationDB.known_detector_conditions.BaseDetectorCondition*, *iCalibrationDB.known_detector_conditions.JungfrauMixinCondition*

Basic Jungfrau detector condition.

class iCalibrationDB.known_detector_conditions.**JungfrauIlluminatedCondition**(*memory_cells*, *bias_voltage*, *photon_energy*, *integration_time*, *pixels_x=1024*, *pixels_y=512*, *beam_energy=None*, *temperature=291*, *gain_setting=0*)

Bases: *iCalibrationDB.known_detector_conditions.IlluminatedCondition*, *iCalibrationDB.known_detector_conditions.JungfrauMixinCondition*

Illuminated Jungfrau detector condition.

class iCalibrationDB.known_detector_conditions.**JungfrauMixinCondition**

Bases: object

class iCalibrationDB.known_detector_conditions.**LPDCondition**(*memory_cells*, *bias_voltage*, *pixels_x=256*, *pixels_y=256*, *capacitor='5pF'*)

Bases: *iCalibrationDB.known_detector_conditions.BaseDetectorCondition*

Basic LPD detector condition.

class iCalibrationDB.known_detector_conditions.**LPDIlluminatedCondition**(*memory_cells*, *bias_voltage*, *photon_energy*, *pixels_x=256*, *pixels_y=256*, *beam_energy=None*, *capacitor='5pf'*, *category=0*)

Bases: *iCalibrationDB.known_detector_conditions.IlluminatedCondition*

Illuminated LPD detector condition.

class iCalibrationDB.known_versions.**ConvenientVersion**

Bases: *iCalibrationDB.constant_version.ConstantVersion*

class iCalibrationDB.known_versions.**FromFile**(*file_path*, *end=None*)

Bases: *iCalibrationDB.known_versions.ConvenientVersion*

class iCalibrationDB.known_versions.**Now**(*end=None*)

Bases: *iCalibrationDB.known_versions.ConvenientVersion*

class iCalibrationDB.known_versions.**Timespan**(*start*, *end=None*)

Bases: *iCalibrationDB.known_versions.ConvenientVersion*

class iCalibrationDB.known_versions.**Versions**

Bases: object

class FromFile(*file_path*, *end=None*)

Bases: *iCalibrationDB.known_versions.ConvenientVersion*

class Now(*end=None*)

Bases: *iCalibrationDB.known_versions.ConvenientVersion*

class Timespan(*start*, *end=None*)

Bases: *iCalibrationDB.known_versions.ConvenientVersion*

class iCalibrationDB.meta_data.**ConstantMetaData**

Bases: *iCalibrationDB.util.DictConvertable*

calibration_constant

The calibration constant this meta data refers to.

An object of type *CalibrationConstant* is expected. A selection of predefined constants is available in the *known_constants* module.

calibration_constant_version

The version of the constant this meta data refers to.

An object of type *ConstantVersion* is expected. A selection of predefined versions is available in the *known_versions* module.

calibration_hash_schema_version

Version of the injection Hash schema to use.

The default version is usually appropriate.

detector_condition

The detector operating condition the constant is valid for/ is requested for.

An object of type *DetectorCondition* is expected. A selection of predefined operating conditions is available in the

get_from_version_info(*ccv*)

Return a constant from a dictionary descriptor returned by ‘retrieve(. . . , version_info=True)

Parameters ccv – calibration constant version

mandatory = ['detector_condition', 'calibration_constant_version', 'calibration_consta

producing_device

(Karabo) device producing this constant. Defaults to “interactive”.

retrieve(*receiver: str*, *when: Optional[str] = None*, *silent: Optional[bool] = True*, *timeout: Optional[int] = 30000*, *meta_only: Optional[bool] = False*, *version_info: Optional[bool] = False*, *strategy: Optional[str] = 'pdu_closest_by_time'*)

Retrieve a constant and its meta data from the database at *receiver*.

Receiver should be ZMQ address of of the form *tcp://localhost:5050*. The *when* parameter defaults to *None*: in this case the most current constant version will be requested. To request a version for a specific time, set *when* an iso-formatted time string, e.g. using *datetime.now().isoformat()*.

If a constant is successfully returned, the requesting *ConstantMetaData* object updates itself to reflect the parameter conditions of the constant.

The constant itself can then be found at *self.calibration_constant.data*.

If meta_only is set, then no constant data is transferred. Instead constants are read from a .h5 file by the client. The variable meta_only is overwritten by the environmental variable *CAL_DB_METAONLY*, if it is defined.

if version_info flag is True, then function returns a list of meta-data for constant-versions. The meta_only flag in this case is not used. :param receiver: ZMQ address, e. g. *tcp://localhost:5050* :param timeout: Timeout for zmq request :param strategy: Default pdu_closest_by_time, options:

a) **pdu_closest_by_time: use physical detector unit to** retrieve CCV closest to measured-at

b) **detector_closest_by_time: use karabo-id and karabo-da to** retrieve CCV closest to measured-time

c) **pdu_prior_in_time: use PDU to retrieve CCV prior on time** only to measured-at

Raises A *RuntimeError* if database communication fails or no matching constant is found.

`retrieve_from_version_info`(*ccv*)

Retrieve a constant and meta data from a dictionary descriptor returned by ‘retrieve(…, version_info=True)

Parameters **`ccv`** – calibration constant version

`retrieve_pdus_for_detector`(*receiver: str*, *karabo_id: str*, *snapshot_at: Optional[str] = ”*, *timeout: Optional[int] = 30000*) → List[dict]

Retrieve physical detector units corresponding to a detector identifier

Parameters

- **`receiver`** – ZMQ address, e. g. *tcp://localhost:5050*
- **`karabo_id`** – The karabo id which is used as a detector identifier in CalCat
- **`snapshot_at`** – CalCat database snapshot
- **`timeout`** – Timeout for zmq request

Returns

`send`(*receiver: str*, *silent: Optional[bool] = True*, *timeout: Optional[int] = 30000*)

Send a constant and its meta data to the database at *receiver*.

Receiver should be ZMQ address of of the form *tcp://localhost:5050*.

Raises A *RuntimeError* if database communication fails or no matching conditions are found.

`update_flg_good_quality`(*receiver*, *cvv_id*, *flg_good_quality*, *silent=True*, *timeout=30000*)

Update *flg_good_quality* of the calibration constant version.

Parameters

- **`receiver`** – ZMQ address, e. g. *tcp://localhost:5050*
- **`cvv_id`** – calibration constant version if

- **`flg_good_quality`** – flg_good_quality to be set
- **`silent`** – Set to False to print information
- **`timeout`** – Timeout for zmq request

Returns response message

Raises A *RuntimeError* if database communication fails or *update_keys* are not set.

class `iCalibrationDB.operating_condition.`**`OperatingCondition`**

Bases: *`iCalibrationDB.util.DictConvertable`*

A class describing a detector operating condition parameter.

It is mandatory to give a *name* and *value*.

Optionally, acceptable upper and lower deviations, and whether these are to be evaluated logarithmically can be specified.

`available`

Internal parameter, should be kept at its default (True).

`description`

A description of the property

`logarithmic`

If set to *True*, deviations are evaluated as decades of a logarithm

`lower_deviation`

Absolute deviation to lower values that is acceptable

`mandatory = ['flg_available', 'flg_logarithmic', 'parameter_name', 'value']`

`name`

The property described by this condition. May contain spaces

`upper_deviation`

Absolute deviation to lower higher that is acceptable

`value`

Value of the variable. Needs to be convertable to *float*.

class `iCalibrationDB.settings.`**`Settings`**

Bases: `object`

`base_path = '/gpfs/exfel/d/cal/caldb_store/'`

class `iCalibrationDB.util.`**`DictConvertable`**

Bases: `object`

A class whose properties are convertible to a *dict*.

Overwrite the *mandatory* list in derived classes to specify mandatory properties.

Overwrite the *retrieve_optionals* in derived classes to specify which parameters out of mandatory are optional for retrieve queries.

`mandatory = []`

`retrieve_optionals = []`

`to_dict`(*mode='send'*, *silent=True*)

Convert properties of class to to *dict*.

`iCalibrationDB.util.`**`float_from_integer`**(*integer*) `iCalibrationDB.util.`**`integer_from_float`**`(`*double*`)`

2.1.3 Module contents

- Available calibration constants

CHAPTER 3

Indices and tables

- genindex
- modindex
- search

Python Module Index

Index