DQuality Documentation

Release 0.0.2

Argonne National Laboratory

Sep 27, 2017

Contents

DQuality is an open-sourced Python package to perform data integrity and quality assessment at the Advanced Photon Source.

This guide is maintained on GitHub.

Data Quality assurance is an essential task to provide consistency, accuracy, reliability and reproducibility of scientific results *[C1]*, *[C2]*, *[C3]*, *[C4]*.

DQuality is a python toolbox to asses the quality of raw and processed data collected at the Advanced Photon Source (APS).

DQuality verifies that all experiment components EPICS Process Variables (PVs) like motor positions, storage ring parameters etc. are valid and their values are within a predefined range.

DQuality verifies that the raw data files saved in the Data Exchange hdf5 file format contain a valid data and meta-data structure and that existing dependencies between the data and meta-data structure are correct.

DQuality provides live monitoring of the above functionalities for all raw data created in the active directory.

CHAPTER 1

Install

This section covers the basics of how to download and install DQuality.

Contents:

Pre-requisites

Each of the verifier requires a parameter configuration file. The configuration files must define schemas and verifier's properties and be located in the user account home directory under a .dquality/default folder.

- `~/.dquality/default/dqconfig.ini`

The schemas should be saved under "*default/schemas*" folder and contain the following files:

- `~/.dquality/default/schemas/pvs.json` containing the list of Process variable (PV) of your beamline PVs with their acceptable value range.
- `~/.dquality/default/schemas/tags.json` containing the list valid HDF file tags, attributes and array dimentions.
- `~/.dquality/default/schemas/dependencies.json` containing the list of valid relation among data sets in the same HDF file.
- `~/.dquality/default/schemas/limits.json` containing the threshold values for the quality check calculations.

Different instruments generating data that require different sets of configuration files can be configured as ~/.dquality/instrument1, ~/.dquality/instrument2 etc. with the same schemas subfolder structure.

Installing from source

Clone the DQuality from GitHub repository:

```
git clone https://github.com/bfrosik/data-quality.git DQuality
```

then:

```
cd DQuality
python setup.py install
```

Installing from Conda/Binstar (coming soon)

First you must have Conda installed, then open a terminal or a command prompt window and run:

```
conda install -c ....
```

Updating the installation (coming soon)

Data Management is an active project, so we suggest you update your installation frequently. To update the installation run in your terminal:

```
conda update -c ....
```

For some more information about using Conda, please refer to the docs.

CHAPTER 2

API reference

DQuality provides the following funtionalities:

- "*PV*": Before data collection start, verify that the experiment setup PVs, i.e. all required setup data, are valid and their values are within a predefined range.
- "*Hdf*": verify the correctness of the data and meta-data structure in an hdf5 file.
- "*Hdf Dependencies*": verify dependencies between the data and meta-data structure in an hdf5 file.
- "*Data*": verify the quality of the data after data is collected in a file. A set of QC functions is provided to assess the image quality against different criteria (mean, dynamic range, structural similarity, multi-scale structural similarity, visual information fidelity, most apparent distortion, etc.) *[C4]*. The resulting "*limit*", related to the quantitive QC functions, defines whether the data is of good or poor quality. The limit values, at first, are set by the research/tests with trial data sets. The QC function "*limit*" range will eventually be learned by implementing a learning mechanism. Any calculated "*result*" quality parameter is stored, in the case of hdf format, in the file itself with a corresponding tag. If the data file format supports only raw data (no meta-data), the quality parameter results are stored in a separate file with a name corresponding to the data file.
- "*Monitor*": monitor the active data collection directory and run "*Data*" on each new file.
- "*Accumulator*": monitor the active data collection directory where each new file is part of the same data set.
- "*Check*": provides a wrapper to "*PV*", "*Hdf*", "*Hdf Dependencies*", "*Data*", "*Monitor*" and "*Accumulator*".

DQuality Modules:

dquality.accumulator

Please make sure the installation *Pre-requisites* are met.

The application monitors given directory for new/modified files of the given pattern. Each of the detected file is verified according to schema configuration and for each of the file several new processes are started, each process performing specific quality calculations.

The results will be sent to an EPICS PV (printed on screen for now).

Functions:

init(config)	This function initializes global variables.
verify(conf, folder, data_type, num_files[, ...])	This is the main function called when the verifier application starts.
directory(directory, patterns)	This method monitors a directory given by the "*directory*" parameter.

dquality.accumulator.**init**(*config*)

This function initializes global variables. It gets values from the configuration file, evaluates and processes the values. If mandatory file or directory is missing, the script logs an error and exits.

Parameters **config** (*str*) – configuration file name, including path

Returns

- **logger** (*Logger*) – logger instance
- **limits** (*dictionary*) – a dictionary containing limit values read from the configured 'limit' file
- **report_file** (*str*) – a report file configured in a given configuration file
- **extensions** (*list*) – a list containing extensions of files to be monitored read from the configuration file

dquality.accumulator.**verify**(*conf*, *folder*, *data_type*, *num_files*, *report_by_files=True*)

This is the main function called when the verifier application starts. It reads the configuration for the directory to monitor, for pattern that represents a file extension to look for, and for a number of files that are expected for the experiment. The number of files configuration parameter is added for experiments that generate multiple files. In some cases the experiment data is collected into a single file, which is organized with data sets.

The function calls directory function that sets up the monitoring and returns notifier. After the monitoring is initialized, it starts a loop that reads the global "*files*" queue and then the global "*results*" queue. If there is any new file, the file is removed from the queue, and the data in the file is validated by a sequence of validation methods. If there is any new result, the result is removed from the queue, corresponding process is terminated, and the result is presented. (currently printed on console, later will be pushed into an EPICS process variable)

The loop is interrupted when all expected processes produced results. The number of expected processes is determined by number of files and number of validation functions.

Parameters

- **conf** (*str*) – configuration file name, including path
- **folder** (*str*) – monitored directory
- **data_type** (*str*) – defines which data type is being evaluated
- **num_files** (*int*) – number of files that will be processed
- **report_by_files** (*boolean*) – this variable directs how to present the bad indexes in a report. If True, the indexes are related to the files, and a filename is included in the report. Otherwise, the report contains a list of bad indexes.

Returns **bad_indexes** (*dict*) – a dictionary or list containing bad indexes

dquality.accumulator.**directory**(*directory*, *patterns*)

This method monitors a directory given by the "*directory*" parameter. It creates a notifier object. The notifier is registered to await the "*CLOSE_WRITE*" event on a new file that matches the "*pattern*" parameter. If there is no such event, it yields control on timeout, defaulted to 1 second. It returns the created notifier.

Parameters

- **file** (*str*) – File Name including path
- **patterns** (*list*) – A list of strings representing file extension

Returns *None*

dquality.check

This module contains command line interface to data-quality. To use Please make sure the installation *Pre-requisites* are met.

Functions:

hdf(conf, fname)	HDF file structure verifier.
pv(conf)	PV verifier.
monitor_quality	
monitor(conf, fname, num_files)	Data quality monitor verifier.
dquality	
dependency	

dquality.check.**hdf**(*conf*, *fname*)
HDF file structure verifier.

Parameters

- **conf** (*str*) – configuration file name, including path
- **file** (*str*) – File Name to verify including path

Returns *boolean*

dquality.check.**pv**(*conf*)
PV verifier.

Parameters **conf** (*str*) – configuration file name, including path

Returns *boolean*

dquality.check.**monitor**(*conf*, *fname*, *num_files*)
Data quality monitor verifier.

Parameters

- **conf** (*str*) – configuration file name including path
- **folder** (*str*) – folder name to monitor
- **num_files** (*int*) – expected number of files. This script will exit after detecting and processing given number of files.

Returns *None*

dquality.check.**accumulator**(*conf*, *fname*, *dtype*, *num_files*, *report_by_file*)
Data Quality monitor.

Parameters

- **conf** (*str*) – configuration file name, including path
- **folder** (*str*) – monitored directory
- **data_type** (*str*) – defines which data type is being evaluated
- **num_files** (*int*) – number of files that will be processed
- **report_by_files** (*boolean*) – this variable directs how to present the bad indexes in a report. If True, the indexes are related to the files, and a filename is included in the report. Otherwise, the report contains a list of bad indexes.

Returns **bad_indexes** (*dict*) – a dictionary or list containing bad indexes

dquality.check.**data**(*conf*, *fname*)

Data Quality verifier.

Parameters

- **conf** (*str*) – name of the configuration file including path. If contains only directory, 'dq-config_test.ini' will a default file name.
- **file** (*str*) – file name to do the quality checks on

Returns **bad_indexes** (*Dict*)

dquality.check.**hdf_dependency**(*conf*, *fname*)

Dependency verifier.

Parameters

- **conf** (*str*) – configuration file name, including path
- **file** (*str*) – File Name to verify including path

Returns *boolean*

dquality.data

Please make sure the installation *Pre-requisites* are met.

This module verifies a configured hd5 file. The data is verified against configured "limits" file. The limits are applied by processes performing specific quality calculations.

The results is a detail report of calculated values. The indexes of slices that did not pass quality check are returned back.

Functions:

init(config)	This function initializes global variables.
process_data(data_type, aggregateq, fp, ...)	This method creates and starts a new handler process.
verify_file(logger, file, data_tags, limits, ...)	This method creates and starts a new handler process.
verify(conf, file)	This invokes sequentially data verification processes for data_dark, data_white, and data that is contained in configured hd5 file.

dquality.data.**init**(*config*)

This function initializes global variables. It gets values from the configuration file, evaluates and processes the

values. If mandatory file or directory is missing, the script logs an error and exits.

Parameters **config** (*str*) – configuration file name, including path

Returns

- **logger** (*Logger*) – logger instance
- **data_tags** (*dict*) – a dictionary od data_type/hdf tag
- **limits** (*dictionary*) – a dictionary containing limit values read from the configured 'limit' file
- **report_file** (*str*) – a report file configured in a given configuration file

dquality.data.**process_data**(*data_type*, *aggregateq*, *fp*, *data_tag*, *limits*)

This method creates and starts a new handler process. The handler is initialized with data queue, the data type, and a result queue. The data type can be 'data_dark', 'data_white' or 'data'. After starting the process the function enqueues queue slice by slice into data, until all data is queued. The last enqueued element is end of the data marker.

Parameters

- **data_type** (*str*) – string indicating what type of data is processed.
- **aggregateq** (*Queue*) – result queue; used by handler to enqueue results
- **fp** (*file pointer*) – a pointer to the file that is verifier
- **data_tag** (*str*) – a data tag to look for
- **limits** (*dict*) – a dictionary of limits values

Returns *None*

dquality.data.**verify_file**(*logger*, *file*, *data_tags*, *limits*, *report_file*)

This method creates and starts a new handler process. The handler is initialized with data queue, the data type, and a result queue. The data type can be 'data_dark', 'data_white' or 'data'. After starting the process the function enqueues queue slice by slice into data, until all data is queued. The last enqueued element is end of the data marker.

Parameters

- **logger** (*Logger*) – Logger instance.
- **file** (*str*) – a filename including path that will be verified
- **data_tags** (*dict*) – a dictionary od data_type/hdf tag
- **limits** (*dict*) – a dictionary of limits values
- **report_file** (*str*) – a name of a report file

Returns **bad_indexes** (*dict*) – a dictionary of bad indexes per data type

dquality.data.**verify**(*conf*, *file*)

This invokes sequentially data verification processes for data_dark, data_white, and data that is contained in configured hd5 file. The calculated values are check against limits, configured in a file. Each process gets two queues parameters; one queue to get the data, and second queue to pass back the results. The function awaits results from the response queues in the matching sequence to how the processes started. The results are retrieved as json object. The indexes of slices that did not pass quality check are reported to the calling function in form of dictionary.

Parameters

- **conf** (*str*) – name of the configuration file including path. If contains only directory, 'dq-config_test.ini' will a default file name.
- **file** (*str*) – file name to do the quality checks on

Returns **bad_indexes** (*Dict*) – A dictionary containing indexes of slices that did not pass quality check. The key is a type of data. (i.e. data_dark, data_white,data)

dquality.hdf

Please make sure the installation *Pre-requisites* are met.

This file contains verification functions related to the file structure. It reads configuration parameters "*schema_type*" and "*schema*" to determine first which kind of file verification is requested, and a schema that defines mandatory parameters. If any of the parameters is not configured, it is assumed no file structure verification is requested.

Functions:

init(config)	This function initializes global variables.
report_items(list, text1, text2, logger)	This function takes a list and strings.
verify(conf, file)	This is the main function called when the structureverifier application starts.
structure(file, required_tags, logger)	This method is used when a file of hdf type is given.
tags(file, required_tags, logger)	This method is used when a file of hdf type is given.

dquality.hdf.**init**(*config*)

This function initializes global variables. It gets values from the configuration file, evaluates and processes the values. If mandatory file or directory is missing, the script logs an error and exits.

Parameters **config** (*str*) – configuration file name, including path

Returns

- **logger** (*Logger*) – logger instance
- **tags** (*dictionary*) – a dictionary containing tag and attributes values read from the configured 'schema' file

dquality.hdf.**report_items**(*list*, *text1*, *text2*, *logger*)

This function takes a list and strings. If the list is not empty it prints the two string parameters as a title, and prints formatted output for each item in a list.

Parameters

- **list** (*list*) – A list of items
- **text1** (*str*) – A title that will be printed if the list is not empty
- **text2** (*str*) – An optional part of title that will be printed if the list is not empty
- **logger** (*Logger*) – a Logger instance

Returns *None*

dquality.hdf.**verify**(*conf*, *file*)

This is the main function called when the structureverifier application starts. It reads the configuration file for "*verification_type*" to verify "*hdf_structure*" or "*hdf_tags*".

Parameters

- **conf** (*str*) – configuration file name, including path
- **file** (*str*) – File Name to verify including path

Returns *boolean*

`dquality.hdf.`**`tags`**(*file*, *required_tags*, *logger*)

This method is used when a file of hdf type is given. All tags from the hdf file are added in the filetags list. Then the schema is evaluated for tags. With each tag discovered it checks whether there is matching tag in the filetags list. If a tag is missing, the function exits with False. Otherwise, it will return True.

Parameters

- **file** (*str*) – File Name including path
- **schema** (*str*) – Schema file name
- **logger** (*Logger*) – a Logger instance

Returns

- *True if verified*
- *False if not verified*

`dquality.hdf.`**`structure`**(*file*, *required_tags*, *logger*)

This method is used when a file of hdf type is given. All tags and array dimensions are verified against a schema. (see `schemas/tags.json` example file).

Parameters

- **file** (*str*) – File Name including path
- **schema** (*str*) – Schema file name
- **logger** (*Logger*) – a Logger instance

Returns *None*

dquality.hdf_dependency

Please make sure the installation *Pre-requisites* are met.

This module verifies that each of the PVs listed in the configuration file exist and their values are set within the predefined range.

The results will be reported in a file (printed on screen for now). An error will be reported back to UI via PV.

Functions:

init(config)	This function initializes global variables.
verify(conf, file)	This function reads the json "*dependencies*" file from the `dqconfig.ini` file.
verify_list(file, list, relation, logger)	This function takes an hd5 file, a list of tags (can be extended) and a relation between the list members.
find_value(tag, dset)	This function takes tag parameter and a corresponding dataset from the hd5 file.

dquality.hdf_dependency.**init**(*config*)

This function initializes global variables. It gets values from the configuration file, evaluates and processes the values. If mandatory file or directory is missing, the script logs an error and exits.

Parameters **config** (*str*) – configuration file name, including path

Returns

- **logger** (*Logger*) – logger instance
- **dep** (*dictionary*) – a dictionary containing dependency values read from the configured 'dependencies' file

dquality.hdf_dependency.**verify**(*conf*, *file*)

This function reads the json "*dependencies*" file from the `dqconfig.ini` file. This file contains dictionary with keys of relations between tags. The value is a list of lists. The relation applies to the tags in inner list respectively. For example if the relation is "*equal*", all tags in each inner list must be equal,The outer list hold the lists that apply the relation. A first element in a inner list is an "*anchor*" element, so all elements are compared to it. This is important for the "*less_than*" type of relation, when the order of parameters is important.

The allowed keys are:

•"*less_than*" - the PV value must be less than attribute value

•"*less_or_equal*" - the PV value must be less than or equal attribute value

•"*equal*" - the PV value must be equal to attribute value

•"*greater_or_equal*" - the PV value must be greater than or equal attribute value

•"*greater_than*" - the PV value must be greater than attribute value

The tag in a "*dependencies*" file can be an hd5 tag, or can have appended keyword "*dim*" and an numeric value indicating axis. If the tag is simple hd5 tag, the verifier compares the value of this tag. If it has the "*dim*" keyword appended, the verifier compares a specified dimension.

Any vakue that does not agree with the configured relation is reported (printed for now). The function returns True if no error was found and False otherwise.

Parameters

- **conf** (*str*) – configuration file name, including path
- **file** (*str*) – File Name to verify including path

Returns *boolean*

dquality.hdf_dependency.**verify_list**(*file*, *list*, *relation*, *logger*)

This function takes an hd5 file, a list of tags (can be extended) and a relation between the list members. First the method creates a tags dictionary from the list of tags. The key is a simple hd5 tag, and the value is a newly created TagValue instance that contains extended tag (or simple tag if simple tag was in the list), and a placeholder for the value. The first tag from the list is retained as an anchor.

The function travers through all tags in the given file. If the tag name is found in the tags dictionary, the `` `find_value` `` method is called to retrieve a defined valueu referenced by the tag. The value is then added to the TagValue instance for this tag.

When all tags are processed the function iterates over the tags dictionary to find if the relation between anchor value and other values can be verified. If any relation is not true, a report is printed for this tag, and the function will return `` `False` ``. Otherwise the function returns `` `True` ``.

Parameters

- **file** (*file*) – an hd5 file to be verified

- **list** (*list*) – list of extended or simple hd5 tags
- **relation** (*str*) – a string specifying the relation between tags in the list
- **logger** (*Logger*) – a Logger instance

Returns *boolean*

dquality.hdf_dependency.**find_value**(*tag*, *dset*)

This function takes tag parameter and a corresponding dataset from the hd5 file. The tag can be a simple hd5 member tag or extended tag. This function assumes that the extended tag is a string containing hd5 tag, a word indicating the tag category (i.e. "*dim*" meaning dimension), and a parameter (i.e. numeric index). In the case of a simple hd5 tag the function returns a value of this member. In the second case the function returns decoded value, in the case of "dim" category, it returns the indexed dimension.

Parameters

- **tag** (*str*) – a simple hd5 tag or extended
- **dset** (*dataset*) – hd5 dataset corresonding to the tag parameter

Returns *value of decoded tag*

dquality.monitor

Please make sure the installation *Pre-requisites* are met.

The application monitors given directory for new/modified files of the given pattern. Each of the detected file is verified according to schema configuration and for each of the file several new processes are started, each process performing specific quality calculations.

The results will be sent to an EPICS PV (printed on screen for now).

Functions:

init(config)	This function initializes global variables.
verify(conf, folder, num_files)	This is the main function called when the verifier application starts.
directory(directory, patterns)	This method monitors a directory given by the "*directory*" parameter.

dquality.monitor.**init**(*config*)

This function initializes global variables. It gets values from the configuration file, evaluates and processes the values. If mandatory file or directory is missing, the script logs an error and exits.

Parameters **config** (*str*) – configuration file name, including path

Returns

- **logger** (*Logger*) – logger instance
- **data_tags** (*dict*) – a dictionary od data_type/hdf tag
- **limits** (*dictionary*) – a dictionary containing limit values read from the configured 'limit' file
- **report_file** (*str*) – a report file configured in a given configuration file

- **extensions** (*list*) – a list containing extensions of files to be monitored read from the configuration file

dquality.monitor.**verify**(*conf*, *folder*, *num_files*)

This is the main function called when the verifier application starts. The configuration and the directory to monitor are passed as parameters, as well as number of files that will be accepted. If the files to monitor for should have certain extension, a list of acceptable file extensions can be defined in a configuration file.

The function calls directory function that sets up the monitoring and returns notifier. After the monitoring is initialized, it starts a loop that reads the global "*files*" queue. If there is any new file, the file is removed and validate with data.verify function.

The loop is interrupted when all expected files produced results.

Parameters

- **conf** (*str*) – configuration file name including path
- **folder** (*str*) – folder name to monitor
- **num_files** (*int*) – expected number of files. This script will exit after detecting and processing given number of files.

Returns *None*

dquality.monitor.**directory**(*directory*, *patterns*)

This method monitors a directory given by the "*directory*" parameter. It creates a notifier object. The notifier is registered to await the "*CLOSE_WRITE*" event on a new file that matches the "*pattern*" parameter. If there is no such event, it yields control on timeout, defaulted to 1 second. It returns the created notifier.

Parameters

- **directory** (*str*) – Directory to monitor
- **patterns** (*list*) – A list of strings representing file extension. Closing matching files will generate event.

Returns *None*

dquality.pv

Please make sure the installation *Pre-requisites* are met.

This module verifies that each of the PVs listed in the configuration file exist and their values are set within the predefined range.

The results will be reported in a file (printed on screen for now). An error will be reported back to UI via PV.

Functions:

init(config)	This function initializes global variables.
verify(conf)	This function reads the `schemas/pvs.json` as set in the `dqconfig.ini` file.
read(pv_str)	This function returns a Process Variable (PV) value or None if the PV does not exist.
state(value, limit)	This function takes boolean "*value*" parameter and string "limit" parameter that can be either "*True*" or "*False*".

dquality.pv.**init**(*config*)

This function initializes global variables. It gets values from the configuration file, evaluates and processes the values. If mandatory file or directory is missing, the script logs an error and exits.

Parameters **config** (*str*) – configuration file name, including path

Returns

- **logger** (*Logger*) – logger instance
- **pvs** (*dictionary*) – a dictionary containing pvs values and attributes read from the configured 'pv_file' file

dquality.pv.**verify**(*conf*)

This function reads the `schemas/pvs.json` as set in the `dqconfig.ini` file. This file contains dictionary with keys of mandatory process variables. The values is a dictionary of attributes, each attribute being either description, or a verification operation. The verification operation attribute has an operation as a key, and the value is the limit of the PV. The allowed keys are:

•"*less_than*" - the PV value must be less than attribute value

•"*less_or_equal*" - the PV value must be less than or equal attribute value

•"*equal*" - the PV value must be equal to attribute value

•"*greater_or_equal*" - the PV value must be greater than or equal attribute value

•"*greater_than*" - the PV value must be greater than attribute value

•"*state*" - to support boolean PVs. The defined value must be "True" or "False".

Any missing PV (i.e. it can't be read) is an error that is reported (printed for now). Any PV value that is out of limit is an error that is reported (printed for now). The function returns True if no error was found and False otherwise.

Parameters **conf** (*str*) – configuration file name, including path

Returns *boolean*

dquality.pv.**read**(*pv_str*)

This function returns a Process Variable (PV) value or None if the PV does not exist.

Parameters **value** (*str*) – name of the PV

Returns *PV value*

dquality.pv.**state**(*value*, *limit*)

This function takes boolean "*value*" parameter and string "limit" parameter that can be either "*True*" or "*False*". The limit is converted to string and compared with the value. The function returns True if the boolean values are equal, False otherwise.

Parameters

- **value** (*numeric*) – value
- **limit** (*str*) – limit value

Returns *boolean*

CHAPTER 3

Examples

This section contains various DQuality examples. Please make sure the installation *Pre-requisites* are met.

File Accumulator

DQuality Folder Accumulator example. (Download file: `accumulator_check.py`)

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-

# ##########################################################################
# Copyright (c) 2015, UChicago Argonne, LLC. All rights reserved.          #
#                                                                          #
# Copyright 2015. UChicago Argonne, LLC. This software was produced        #
# under U.S. Government contract DE-AC02-06CH11357 for Argonne National    #
# Laboratory (ANL), which is operated by UChicago Argonne, LLC for the     #
# U.S. Department of Energy. The U.S. Government has rights to use,        #
# reproduce, and distribute this software.  NEITHER THE GOVERNMENT NOR     #
# UChicago Argonne, LLC MAKES ANY WARRANTY, EXPRESS OR IMPLIED, OR         #
# ASSUMES ANY LIABILITY FOR THE USE OF THIS SOFTWARE.  If software is      #
# modified to produce derivative works, such modified software should      #
# be clearly marked, so as not to confuse it with the version available    #
# from ANL.                                                                #
#                                                                          #
# Additionally, redistribution and use in source and binary forms, with    #
# or without modification, are permitted provided that the following       #
# conditions are met:                                                      #
#                                                                          #
#     * Redistributions of source code must retain the above copyright     #
#       notice, this list of conditions and the following disclaimer.      #
#                                                                          #
#     * Redistributions in binary form must reproduce the above copyright  #
#       notice, this list of conditions and the following disclaimer in    #
#       the documentation and/or other materials provided with the         #
```

```
#       distribution.                                                        #
#                                                                            #
#     * Neither the name of UChicago Argonne, LLC, Argonne National          #
#       Laboratory, ANL, the U.S. Government, nor the names of its           #
#       contributors may be used to endorse or promote products derived      #
#       from this software without specific prior written permission.        #
#                                                                            #
# THIS SOFTWARE IS PROVIDED BY UChicago Argonne, LLC AND CONTRIBUTORS        #
# "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT          #
# LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS          #
# FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL UChicago        #
# Argonne, LLC OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT,           #
# INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING,       #
# BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES;           #
# LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER           #
# CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT         #
# LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN          #
# ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE            #
# POSSIBILITY OF SUCH DAMAGE.                                                #
# ############################################################################
import sys
import os
import argparse
from dquality.check import accumulator as acc
from os.path import expanduser

def main(arg):

    parser = argparse.ArgumentParser()
    parser.add_argument("instrument", help="instrument name, name should have a␣
↪matching directory in the .dquality folder")
    parser.add_argument("fname", help="folder name to monitor for files")
    parser.add_argument("type", help="data type to be verified (i.e. data_dark, data_
↪white, or data)")
    parser.add_argument("numfiles", help="number of files to monitor for")
    parser.add_argument("repbyfile", help="boolean value defining how the bad indexes␣
↪should be reported.")

    args = parser.parse_args()
    instrument = args.instrument
    fname = args.fname
    dtype = args.type
    num_files = args.numfiles
    report_by_file = args.repbyfile

    home = expanduser("~")
    conf = os.path.join(home, ".dquality", instrument)

    bad_indexes = acc(conf, fname, dtype, num_files, report_by_file)
    return bad_indexes


if __name__ == "__main__":
    main(sys.argv[1:])
```

Data Quality

DQuality Data Quality example. (Download file: `data_check.py`)

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-

# ######################################################################
# Copyright (c) 2015, UChicago Argonne, LLC. All rights reserved.      #
#                                                                      #
# Copyright 2015. UChicago Argonne, LLC. This software was produced    #
# under U.S. Government contract DE-AC02-06CH11357 for Argonne National   #
# Laboratory (ANL), which is operated by UChicago Argonne, LLC for the #
# U.S. Department of Energy. The U.S. Government has rights to use,    #
# reproduce, and distribute this software.  NEITHER THE GOVERNMENT NOR #
# UChicago Argonne, LLC MAKES ANY WARRANTY, EXPRESS OR IMPLIED, OR     #
# ASSUMES ANY LIABILITY FOR THE USE OF THIS SOFTWARE.  If software is  #
# modified to produce derivative works, such modified software should  #
# be clearly marked, so as not to confuse it with the version available   #
# from ANL.                                                            #
#                                                                      #
# Additionally, redistribution and use in source and binary forms, with   #
# or without modification, are permitted provided that the following   #
# conditions are met:                                                  #
#                                                                      #
#     * Redistributions of source code must retain the above copyright    #
#       notice, this list of conditions and the following disclaimer.     #
#                                                                      #
#     * Redistributions in binary form must reproduce the above copyright #
#       notice, this list of conditions and the following disclaimer in   #
#       the documentation and/or other materials provided with the        #
#       distribution.                                                  #
#                                                                      #
#     * Neither the name of UChicago Argonne, LLC, Argonne National       #
#       Laboratory, ANL, the U.S. Government, nor the names of its        #
#       contributors may be used to endorse or promote products derived   #
#       from this software without specific prior written permission.     #
#                                                                      #
# THIS SOFTWARE IS PROVIDED BY UChicago Argonne, LLC AND CONTRIBUTORS     #
# "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT       #
# LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS       #
# FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL UChicago     #
# Argonne, LLC OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT,        #
# INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING,    #
# BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES;        #
# LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER        #
# CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT      #
# LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN       #
# ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE         #
# POSSIBILITY OF SUCH DAMAGE.                                          #
# ######################################################################
"""
Please make sure the installation :ref:`pre-requisite-reference-label` are met.

This script is specific for beamline 32id.

This example takes two mandatory parameters:
instrument: a string defining the detector that will be used. User can enter one of␣
↪these choices:
```

```
'nanotomo', 'microtomo'.
The instrument determines a configuration file that will be used.
file: a file to be verified for dependencies according to schema.

This script calls quality_check verifier.

"""
import sys
import os
import argparse
from dquality.check import data as dquality_check
from os.path import expanduser

def main(arg):

    parser = argparse.ArgumentParser()
    parser.add_argument("instrument", help="instrument name, name should have a␣
↪matching directory in the .dquality folder")
    parser.add_argument("fname", help="file name to do the tag dependencies checks on
↪")

    args = parser.parse_args()
    instrument = args.instrument
    fname = args.fname

    home = expanduser("~")
    conf = os.path.join(home, ".dquality", instrument)

    bad_indexes = dquality_check(conf, fname)
    return bad_indexes


if __name__ == "__main__":
    main(sys.argv[1:])
```

HDF File

This module requires the "*schema*" section of the `dqconfig.ini` file to be set. If "*schema*" is not configured, the function returns True, as no verification is needed.

In case the "*verification_type*" of the `dqconfig.ini` file is set, the follow up logic determines, what type of verification should be applied.

Currently, for the HDF schema, both "*hdf_structure*" and "*hdf_tags*" verification types are supported.

DQuality HDF file check example. (Download file: `hdf_check.py`)

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-

# ############################################################################
# Copyright (c) 2015, UChicago Argonne, LLC. All rights reserved.          #
#                                                                          #
# Copyright 2015. UChicago Argonne, LLC. This software was produced        #
# under U.S. Government contract DE-AC02-06CH11357 for Argonne National    #
# Laboratory (ANL), which is operated by UChicago Argonne, LLC for the     #
```

```
# U.S. Department of Energy. The U.S. Government has rights to use,        #
# reproduce, and distribute this software.  NEITHER THE GOVERNMENT NOR     #
# UChicago Argonne, LLC MAKES ANY WARRANTY, EXPRESS OR IMPLIED, OR         #
# ASSUMES ANY LIABILITY FOR THE USE OF THIS SOFTWARE.  If software is      #
# modified to produce derivative works, such modified software should      #
# be clearly marked, so as not to confuse it with the version available    #
# from ANL.                                                                #
#                                                                          #
# Additionally, redistribution and use in source and binary forms, with    #
# or without modification, are permitted provided that the following       #
# conditions are met:                                                      #
#                                                                          #
#     * Redistributions of source code must retain the above copyright     #
#       notice, this list of conditions and the following disclaimer.      #
#                                                                          #
#     * Redistributions in binary form must reproduce the above copyright  #
#       notice, this list of conditions and the following disclaimer in    #
#       the documentation and/or other materials provided with the         #
#       distribution.                                                      #
#                                                                          #
#     * Neither the name of UChicago Argonne, LLC, Argonne National        #
#       Laboratory, ANL, the U.S. Government, nor the names of its         #
#       contributors may be used to endorse or promote products derived    #
#       from this software without specific prior written permission.      #
#                                                                          #
# THIS SOFTWARE IS PROVIDED BY UChicago Argonne, LLC AND CONTRIBUTORS      #
# "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT        #
# LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS        #
# FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL UChicago      #
# Argonne, LLC OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT,         #
# INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING,     #
# BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES;         #
# LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER         #
# CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT       #
# LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN        #
# ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE          #
# POSSIBILITY OF SUCH DAMAGE.                                              #
# ##########################################################################
import sys
import os
import argparse
from dquality.check import hdf as hdf_check
from os.path import expanduser

def main(arg):

    parser = argparse.ArgumentParser()
    parser.add_argument("instrument", help="instrument name, name should have a␣
↪matching directory in the .dquality folder")
    parser.add_argument("fname", help="file name to do the tag dependencies checks on
↪")

    args = parser.parse_args()
    instrument = args.instrument
    fname = args.fname

    home = expanduser("~")
    conf = os.path.join(home, ".dquality", instrument)
```

```
    bad_indexes = hdf_check(conf, fname)
    return bad_indexes


if __name__ == "__main__":
    main(sys.argv[1:])
```

Dependency Check

DQuality Data Dependency check example. (Download file: `hdf_dependency_check.py`)

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-

# ##########################################################################
# Copyright (c) 2015, UChicago Argonne, LLC. All rights reserved.          #
#                                                                          #
# Copyright 2015. UChicago Argonne, LLC. This software was produced        #
# under U.S. Government contract DE-AC02-06CH11357 for Argonne National    #
# Laboratory (ANL), which is operated by UChicago Argonne, LLC for the     #
# U.S. Department of Energy. The U.S. Government has rights to use,        #
# reproduce, and distribute this software.  NEITHER THE GOVERNMENT NOR     #
# UChicago Argonne, LLC MAKES ANY WARRANTY, EXPRESS OR IMPLIED, OR         #
# ASSUMES ANY LIABILITY FOR THE USE OF THIS SOFTWARE.  If software is      #
# modified to produce derivative works, such modified software should      #
# be clearly marked, so as not to confuse it with the version available    #
# from ANL.                                                                #
#                                                                          #
# Additionally, redistribution and use in source and binary forms, with    #
# or without modification, are permitted provided that the following       #
# conditions are met:                                                      #
#                                                                          #
#     * Redistributions of source code must retain the above copyright     #
#       notice, this list of conditions and the following disclaimer.      #
#                                                                          #
#     * Redistributions in binary form must reproduce the above copyright  #
#       notice, this list of conditions and the following disclaimer in    #
#       the documentation and/or other materials provided with the         #
#       distribution.                                                      #
#                                                                          #
#     * Neither the name of UChicago Argonne, LLC, Argonne National        #
#       Laboratory, ANL, the U.S. Government, nor the names of its         #
#       contributors may be used to endorse or promote products derived    #
#       from this software without specific prior written permission.      #
#                                                                          #
# THIS SOFTWARE IS PROVIDED BY UChicago Argonne, LLC AND CONTRIBUTORS      #
# "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT        #
# LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS        #
# FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL UChicago      #
# Argonne, LLC OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT,         #
# INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING,     #
# BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES;         #
# LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER         #
# CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT       #
```

```
# LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN       #
# ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE         #
# POSSIBILITY OF SUCH DAMAGE.                                             #
# ########################################################################
"""
Please make sure the installation :ref:`pre-requisite-reference-label` are met.

This script is specific for beamline 32id.

This example takes two mandatory parameters:
instrument: a string defining the detector that will be used. User can enter one of these choices:
'nanotomo', 'microtomo'.
The instrument determines a configuration file that will be used.
file: a file to be verified for dependencies according to schema.

This script calls dependency_check verifier.

"""
import sys
import os
import argparse
from dquality.check import hdf_dependency as dependency_check
from os.path import expanduser

def main(arg):

    parser = argparse.ArgumentParser()
    parser.add_argument("instrument", help="instrument name, name should have a matching directory in the .dquality folder")
    parser.add_argument("fname", help="file name to do the tag dependencies checks on")

    args = parser.parse_args()
    instrument = args.instrument
    fname = args.fname

    home = expanduser("~")
    conf = os.path.join(home, ".dquality", instrument)

    bad_indexes = dependency_check(conf, fname)
    return bad_indexes


if __name__ == "__main__":
    main(sys.argv[1:])
```

File Monitor

DQuality Folder Monitoring example. (Download file: `monitor_check.py`)

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-

# ########################################################################
```

```
# Copyright (c) 2015, UChicago Argonne, LLC. All rights reserved.          #
#                                                                          #
# Copyright 2015. UChicago Argonne, LLC. This software was produced        #
# under U.S. Government contract DE-AC02-06CH11357 for Argonne National    #
# Laboratory (ANL), which is operated by UChicago Argonne, LLC for the     #
# U.S. Department of Energy. The U.S. Government has rights to use,        #
# reproduce, and distribute this software.  NEITHER THE GOVERNMENT NOR     #
# UChicago Argonne, LLC MAKES ANY WARRANTY, EXPRESS OR IMPLIED, OR         #
# ASSUMES ANY LIABILITY FOR THE USE OF THIS SOFTWARE.  If software is      #
# modified to produce derivative works, such modified software should      #
# be clearly marked, so as not to confuse it with the version available    #
# from ANL.                                                                #
#                                                                          #
# Additionally, redistribution and use in source and binary forms, with    #
# or without modification, are permitted provided that the following       #
# conditions are met:                                                      #
#                                                                          #
#     * Redistributions of source code must retain the above copyright     #
#       notice, this list of conditions and the following disclaimer.      #
#                                                                          #
#     * Redistributions in binary form must reproduce the above copyright  #
#       notice, this list of conditions and the following disclaimer in    #
#       the documentation and/or other materials provided with the         #
#       distribution.                                                      #
#                                                                          #
#     * Neither the name of UChicago Argonne, LLC, Argonne National        #
#       Laboratory, ANL, the U.S. Government, nor the names of its         #
#       contributors may be used to endorse or promote products derived    #
#       from this software without specific prior written permission.      #
#                                                                          #
# THIS SOFTWARE IS PROVIDED BY UChicago Argonne, LLC AND CONTRIBUTORS      #
# "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT        #
# LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS        #
# FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL UChicago      #
# Argonne, LLC OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT,         #
# INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING,     #
# BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES;         #
# LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER         #
# CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT       #
# LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN        #
# ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE          #
# POSSIBILITY OF SUCH DAMAGE.                                              #
# ##########################################################################
import sys
import os
import argparse
from dquality.check import monitor as monitor_check
from os.path import expanduser

def main(arg):

    parser = argparse.ArgumentParser()
    parser.add_argument("instrument", help="instrument name, name should have a␣
↪matching directory in the .dquality folder")
    parser.add_argument("fname", help="folder name to monitor for files")
    parser.add_argument("numfiles", help="number of files to monitor for")

    args = parser.parse_args()
```

```
    instrument = args.instrument
    fname = args.fname
    num_files = args.numfiles

    home = expanduser("~")
    conf = os.path.join(home, ".dquality", instrument)

    bad_indexes = monitor_check(conf, fname, num_files)
    return bad_indexes


if __name__ == "__main__":
    main(sys.argv[1:])
```

PV Check

DQuality PV check example. (Download file: `pv_check.py`)

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-

# ##########################################################################
# Copyright (c) 2015, UChicago Argonne, LLC. All rights reserved.          #
#                                                                          #
# Copyright 2015. UChicago Argonne, LLC. This software was produced        #
# under U.S. Government contract DE-AC02-06CH11357 for Argonne National    #
# Laboratory (ANL), which is operated by UChicago Argonne, LLC for the     #
# U.S. Department of Energy. The U.S. Government has rights to use,        #
# reproduce, and distribute this software.  NEITHER THE GOVERNMENT NOR     #
# UChicago Argonne, LLC MAKES ANY WARRANTY, EXPRESS OR IMPLIED, OR         #
# ASSUMES ANY LIABILITY FOR THE USE OF THIS SOFTWARE.  If software is      #
# modified to produce derivative works, such modified software should      #
# be clearly marked, so as not to confuse it with the version available    #
# from ANL.                                                                #
#                                                                          #
# Additionally, redistribution and use in source and binary forms, with    #
# or without modification, are permitted provided that the following       #
# conditions are met:                                                      #
#                                                                          #
#     * Redistributions of source code must retain the above copyright     #
#       notice, this list of conditions and the following disclaimer.      #
#                                                                          #
#     * Redistributions in binary form must reproduce the above copyright  #
#       notice, this list of conditions and the following disclaimer in    #
#       the documentation and/or other materials provided with the         #
#       distribution.                                                      #
#                                                                          #
#     * Neither the name of UChicago Argonne, LLC, Argonne National        #
#       Laboratory, ANL, the U.S. Government, nor the names of its         #
#       contributors may be used to endorse or promote products derived    #
#       from this software without specific prior written permission.      #
#                                                                          #
# THIS SOFTWARE IS PROVIDED BY UChicago Argonne, LLC AND CONTRIBUTORS      #
# "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT        #
# LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS        #
```

```
# FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL UChicago      #
# Argonne, LLC OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT,         #
# INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING,     #
# BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES;         #
# LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER         #
# CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT       #
# LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN        #
# ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE          #
# POSSIBILITY OF SUCH DAMAGE.                                              #
# ##########################################################################
"""
Please make sure the installation :ref:`pre-requisite-reference-label` are met.

This script is specific for beamline 32id.

This example takes two mandatory parameters:
instrument: a string defining the detector that will be used. User can enter one of␣
↪these choices:
'nanotomo', 'microtomo'.
The instrument determines a configuration file that will be used.
file: a file to be verified for dependencies according to schema.

This script calls hdf_check verifier.

"""
import sys
import os
import argparse
from dquality.check import pv as pv_check
from os.path import expanduser

def main(arg):

    parser = argparse.ArgumentParser()
    parser.add_argument("instrument", help="instrument name, name should have a␣
↪matching directory in the .dquality folder")

    args = parser.parse_args()
    instrument = args.instrument

    home = expanduser("~")
    conf = os.path.join(home, ".dquality", instrument)

    bad_indexes = pv_check(conf)
    return bad_indexes


if __name__ == "__main__":
    main(sys.argv[1:])
```

CHAPTER 4

Frequently asked questions

Here's a list of questions.

Questions

How can I report bugs?

The easiest way to report bugs or get help is to open an issue on GitHub. Simply go to the project GitHub page, click on Issues in the right menu tab and submit your report or question.

CHAPTER 5

Credits

References

CHAPTER 6

Howto

This file provides instructions how to use the framework to add more features.

How to add a statistical quality check.

Statistical quality check uses previously stored results to evaluate the current result. Follow the steps below. - add your method to dquality/common/qualitychecks.py file. The signature should be: function_name(result, aggregate, results, all_limits). Look at the example statistical method "validate_stat_mean" for the meaning of the parameters.

- define quality id in a dquality/common/constants.py file (for example STAT_MEAN = 3) that is unique in respect to other quality checks constants.
- update function_mapper dictionary in the dquality/handler.py file. If the statistical method uses mean values, it should be added to the list with the const.QUALITYCHECK_MEAN, if the method uses standard deviation values, it should be added to the list with the const.QUALITYCHECK_STD.
- add import "from dquality.common.qualitychecks import validate_stat_mean" in dquality/handler.py file.

CHAPTER 7

Indices and tables

- genindex
- modindex
- search

Bibliography

[C1] Data Quality Control. http://cbs.fas.harvard.edu/science/core-facilities/neuroimaging/information-investigators/qc. Accessed: 2016-03-02.

[C2] MRI Quality Control. http://cbs.fas.harvard.edu/usr/mcmains/CBS_MRI_Quality_Control_Workshop.pdf. Accessed: 2016-03-02.

[C3] L. W. Goldman. Principles of CT: radiation dose and image quality. *J Nucl Med Technol*, 35(4):213–225, Dec 2007.

[C4] Pedram Mohammadi, Abbas Ebrahimi-Moghadam, and Shahram Shirani. Subjective and objective quality assessment of image: A survey. *CoRR*, 2014. URL: http://arxiv.org/abs/1406.7799.

Python Module Index

d

Index