QA-DKRZ Documentation

Release 0.5

hdh

Jul 18, 2017

Contents

Contents:

CHAPTER 1

QA-DKRZ - Quality Assurance Tool for Climate Data

Version 0.5.x

The Quality Assurance tool QA-DKRZ developed at DKRZ checks conformance of meta-data of climate simulations given in *NetCDF* format with conventions and rules of projects. At present, checking of CF Conventions, CORDEX, CMIP5, and CMIP6, the latter with the option to run PrePARE.py (D. Nadeau, LLNL, https://cmor.llnl.gov) is supported. The check results are summarised in json-formatted files.

Note: The QA tool is still in a testing stage.

Getting Started

The recommendated way to install QA-DKRZ is to use the conda package manager.

```
$ conda create -n qa-dkrz -c conda-forge -c h-dh qa-dkrz
```

If a machine is not suited for Linux-64Bits or you would like to work with the sources, then see *Installation* for details.

The success of the installation may be checked either by running

```
$ qa-dkrz --example
```

(creating a directory *example* in the current path. More details in *Results*) or by operating the stand-alone-checker with a NetCDF file of your choice.

```
$ dkrz-cf-checker your-choice.nc
```

Documentation

QA-DKRZ is using Sphinx, and the latest documentation can be found on ReadTheDocs.

Getting Help

Please, direct questions or comments to hollweg@dkrz.de

Mailing list

Join the mailing list ...

Bug tracker

If you have any suggestions, bug reports or annoyances please report them to our issue tracker at https://github.com/IS-ENES-Data/QA-DKRZ/issues

Contributing

The sources of *QA-DKRZ* are available on Github: https://github.com/IS-ENES-Data/QA-DKRZ

You are highly encouraged to participate in the development.

License

Please notice the *Disclaimer of Warranty* (*DoW.txt*) file in the top distribution directory.

CHAPTER 2

Frequently Asked Questions

Questions

Q: Why does ...?

Answer: Just because ...

CHAPTER 3

Glossary

Quality Assurance Quality assurance (QA) is a way of preventing mistakes ... (Wikipedia).

CHAPTER 4

Change History

0.5

release-date TBA

- packaged as conda.
- travis continous integration added.
- published docs as sphinx on ReadTheDocs.

0.4

release-date 2014-mm-dd

- CORDEX checker added ...

CHAPTER 5

User-guide

User Guide

Release 0.5

Date Jul 18, 2017

Overview

What is checked?

The Quality Assurance (QA) tool developed at DKRZ tests the compliance of meta-data of climate simulations given in *NetCDF* format to conventions and rules of projects. Additionally, the QA checks time sequences and the technical state of data (i.e. occurrences of *Inf* or *NaN*, gaps, replicated sub-sets, etc.) for atomic data set entities, i.e. variable and frequency, e.g. *tas* and *mon* for monthly means of near-surface air temperature‘. When atomic data sets are subdivided into several files, then changes between these files in terms of (auxiliary) coordinate variables will be detected as well as gaps or overlapping time ranges. This may also apply to follow-up experiments.

At present, the QA checks data of the projects *CMIP5*, *CMIP6* and *CORDEX* by consulting tables based on requested controlled vocabulary and requirements. When such pre-defined information is given about directory structures, format of file names, variables and attributes, geographical domain, etc., then any deviation from compliance will be annotated in the QA results.

Files with meta-data similar to a project, but a few deviating features from the default tables can also be checked. User-defined tables may disable particular checks and supersede e.g. the default project name.

While former versions took for granted that meta-data of files were valid in terms of the NetCDF Climate and Forecast (CF) Meta Data Conventions, compliance is now verified. The CF check is both embedded in the QA tool itself and provided by a stand-alone tool, which is described below. Also available is a test suite used during the development phase.

There was a discussion during the 2016-ESGF_F2F_Conference (http://esgf.llnl.gov) for CMIP6 that passing a DRS check and the CMOR3 checker (http://cmor.llnl.gov/mydoc_cmip6_validator) is sufficient to enter files in ESGF-CoG nodes. The CMOR3 validator *PrePARE.py* is additionally run by the QA-DKRZ tool; the results are merged into the flow of QA-DKRZ annotations.

After installation, QA-DKRZ runs are started on the command-line.

```
$ /package-path/scripts/qa-DKRZ [opts]
```

Note: The name of the tool has changed over the years as well as the name of the starting script. At first, the acronym QC (Quality Control/Check) was used, but was changed to the more commonly used term QA (Quality Assurance). The former script names qcManager as well as qc_DKRZ, and the underscore replaced by a hyphen work, too. Also, option names containing QC or qc are still valid for backward compatibility.

Available Versions

At present, the QA-DKRZ package is a rolling release and is provided on GitHub

```
$ git clone  https://github.com/IS-ENES-Data/QA-DKRZ
```

Installation

QA-DKRZ may be installed either via the *conda* package manager or from a GitHub repository.

Recommended is the conda package manager installing a ready-to-use package. At present, a 64-bit processor is required.

If you want to work with sources or use a different machine architecture, then the installation from source should be the first choice.

For CMIP6, *QA-DKRZ* binds the CMOR validator *PrePARE.py* (D. Nadeau, LLNL, https://cmor.llnl.gov). This is conducted only by *conda*, e.g.

```
$ conda create -n cmor -c conda-forge -c pcmdi -c uvcdat cmor
```

Details in section Work-flow below.

No matter what the preference is *conda* or from *GitHub*, you are asked for a location of tables and programs during the installation.

By Conda Package Manager

Make sure that you have conda installed. The quick steps to install *miniconda* on Linux 64-bit are:

```
$ wget https://repo.continuum.io/miniconda/Miniconda-latest-Linux-x86_64.sh
$ bash Miniconda-latest-Linux-x86_64.sh
```

Note: The installation is tested on various 64-bit machines.

Please check that the `conda` package manager is in your `PATH`. For example you may set the `PATH` environment variable as following:

```
$ export PATH=$PATH:your-path/your-miniconda/bin
$ conda -h
```

conda provides the QA-DKRZ package with all dependencies included. You may install it to the root environment or to any other with a name of your choice, e.g.

```
$ conda create -n qa-dkrz -c conda-forge -c h-dh qa-dkrz
```

From GitHUB repository

Requirements

The tool requires the BASH shell and a C/C++ compiler (works for AIX, too).

Building the QA tool

The sources are downloaded from GitHub by

```
$ git clone  https://github.com/IS-ENES-Data/QA-DKRZ
```

Any installation is done with the script `install` (a prefix '`./`' could be helpful in some cases) within the `QA-DKRZ` root directory. Please, have also a look at the Work-flow section.

- A file `install_configure` in `QA-DKRZ` binds necessary `lib` and `include` directories. A template `.install_configure` is provided. If not copied to `install_configure` and edited before the first launch, then it is created and the user is asked to edit the file.
- Environmental variables CC, CXX, CFLAGS, and CXXFLAGS are accepted.
- `install` establishes access to libraries, which may be linked (recommended) or built (triggered by option `--build`).
- Proceedings are logged in file `install.log`.
- The executables are compiled for projects named on the command-line.
- When `install` is started the first time for a given project, then corresponding external tables and programs are downloaded to a `QA_HOME` directory; the user is asked for the path.
- The user's home directory contains a config.txt file in directory .qa-dkrz created and updated by user defined control statements.

The full set of options is described by:

```
$ ./install --help
```

Building Libraries

If you decide to use your own set of libraries (accessing provided ones is preferred by respective settings in the install_configure file), then this is accomplished by

```
$ ./install --build [opts]
```

Sources of the following libraries are downloaded and installed:

- zlib-1.2.8 from www.zlib.net,
- hdf5-1.8.9 from www.hdfgroup.org,
- netcdf-4.3.0 from www.unidata.ucar.edu (shared, no FORTRAN, non-parallel),
- udunits package from http://www.unidata.ucar.edu/packages/udunits.

The libraries are built in sub-directory `local/source`. If libraries had been built previously, then the sources are updated and the libraries are rebuilt.

Work-flow

Please note that the work-flow has changed. By now, operational mode is generally distinct from installation or updating.

Operation is conducted (with some reasonable options) by e.g.

```
$ qa-dkrz -f task-file
```

with task-file containing frequently or task-specific options like `PROJECT_DATA`, `QA_RESULTS`, `SELECT`, `EMAIL_SUMMARY`, `NUM_EXEC_THREADS`, `CHECK_MODE` and `PROJECT` or `QA_CONF`. The full set of options used by default is given in file *QA_HOME/tables/projects/PROJECT/PROJECT_qa.conf* with `QA_HOME` chosen during the installation. Additionally, every valid option may be provided on the command-line by prefixing `-e`.

Note that option `RUN_CMOR3_LLNL` must be given for running *PrePARE.py*.

Installation and update tasks are indicated by option `--install=...`, which is additionally provided to the *qa-dkrz* call. Then, all options of *install* have to be given by a comma-separated list; the usual `--` prefix of options may be omitted, e.g.

```
$ qa-dkrz --install=help
```

would display all install options. The `--install` option may be merged with operational options conducted afterwards.

The separation of operation and install/updating is broken by option `--auto-up` for backward-compatibility. Additionally, when working with the GitHub based package, the `--install` option may be replaced by the *QA-DKRZ/install* command (then options with `--` prefix). Note that *conda* does not know the path to *install*.

Operation/installation/update with conda requires *activating* and *deactivating* the corresponding environment. Provided that *conda* is within the path or aliased:

```
$ source activate your-chosen-environment-name
$ qa-dkrz options
$ source deactivate.
```

Command *which qa-dkrz* while the environment is activated reveals the path to qa-dkrz. Given to *PATH* or as *alias* would make activate/deactivate redundant.

There are three stages for updating:

***install [PROJECT]* without any additional option.** Compilation of executables for PROJECT. Only for GitHub based installation; no effect for a conda installed *qa-dkrz*.

install –up Updating once every other number of days, while num has been given by option `--uf=num` or one day be default.

install –force Unconditional update

Information is kept in a configuration file located in *~/.qa-dkrz/config.txt* by default. User may not edit it, but could. If missing, it is restored during the next *qa-dkrz* call.

A corrupt or incomplete installation becoming apparent during an operational run would issue a message for resolving the conflict.

For those who want to have full access to the sources, but do not like to grapple with library dependencies, a hybrid operation of conda and GitHub based sources QA_DKRZ is possible. Just bind the conda installed libs to the file *QA-DKRZ/install_configure* and operate/update from the GitHub based QA-DKRZ.

Configuration

The QA would run basically on the command-line only with the specification of the target(s) to be checked. However, using options and gathering these in conf-files facilitates checks.

Configuration options follow a specific syntax with option names.

```
KEY-WORD                           enable key-word; equivalent to key-word=t
KEY-WORD   =     value[,value ...]   assign key-word a [comma-separated] value;␣
↪overwrite
KEY-WORD   +=    value[,value ...]   same, but appended
```

Partitioning of Data Sets

The specification of a path to a data directory tree by option PROJECT_DATA results in a check of every NetCDF files found within the entire tree. This can be further customised by the key-words SELECT and LOCK, which follow special rules.

```
KEY-WORD                              var1[,var2,...]   specified variables for every␣
↪path; += mode
KEY-WORD      path1[,path2,...] [=]                     all variables within the␣
↪specified path; +=mode
KEY-WORD      path1[,path2,...]   =   [var1[,var2,...]   specified variables within the␣
↪given paths; += mode
KEY-WORD   =   path1[,path2,...]   =   [var1[,var2,...]   same, but overwrite mode
KEY-WORD  +=   path1[,path2,...]   =   [var1[,var2,...]   append to a previous assignment
```

Some options act on other options:

- Option names on the command-line are case-insensitive they have to be prefixed by '-e' '-E' (a blank or _ may follow).
- Highest precedence is for options on the command-line.
- If path has no leading '/', then the search is relative to the path specified by option PROJECT_DATA.
- Special for options `-S arg` or an appended plain string on the command-line: cancellation of previous SELECTions by any configuration file.
- If SELECTions are specified on the command-line (options -S) with an absolute path, i.e. beginning with '/', then PROJECT_DATA specified in any config-file is cancelled.
- All SELECTions refer to atomic variables, i.e. all sub-temporal files, even if a file name is appended to a path.
- LOCKing gets the higher precedence over SELECTion.
- Path and variable indicators are separated by the '=' character, which may be omitted when there is no variable (except the case that the path item contains no '/' character).

- Regular expressions may be applied to both path(s) and variable(s).
- A variable is selected if the beginning of the name is unambiguous, e.g. specifying 'tas' would select all tas, tasmax, and tasmin files. Note that every file name begins with `variable_...` for CMIP5/CORDEX, thus, use `tas_` for this alone.
- Former option names are valid, if they have changed in the meantime.

Configuration Files

A description of the configuration options is given in the repository.

```
QA-DKRZ:
/package-path/tables/projects/"project-name"/"project-name"_qa.conf
```

Configuration files and options may be specified multiply following a given precedence. This facilitates having a file with short-term options (in a file attached to the -f option on the command-line), another one with settings to site-specific demands, which are robust against changes in the repository, and long-term default settings from the repository. All options may be specified on the command-line plus some more (`/package-path/QA-DKRZ/scripts/qa_DKRZ --help`). A sequence of configuration files is defined by `QA_CONF=conf-file` assignments embedded in the configuration files. The precedence of configuration files/options is given below from highest to lowest:

- directly on the command-line
- in the task-file (`-f file`) specified on the command-line.
- QA_CONF assignments embedded (descending starts from the `-f file`).
- site-specific files provided by files located straight in `/package-path/tables`.
- defaults for the entire project:

Experiment Names

QA-DKRZ checks files individually, the results are gathered in containers unique for a specific scope. Albeit most projects have defined a term 'experiment', this is not suitable to provide a realm common to a sub-set of data files. Note that having everything identical, but the model for instance, could cause annotations, i.e. differences for some variables, in the QA results.

For CMIP5/6 and CORDEX, an unambiguous scope is defined by the properties of the so-called Data Reference System (DRS), i.e. the components of the path to the variables. The options `LOG_PATH_INDEX` and `DRS_PATH_BASE` define a unique experiment-name, where the former contains a comma-separated list of indices of the path components and the latter the starting component with index=0, e.g. `DRS_PATH_BASE=output` and `LOG_PATH_INDEX=1,2,3,4,6`. An example is commented in *Results*.

Similar is for building the name of a consistency table, which is used to check consistency between a parent and a child experiment.

Note: If `CT_PATH_INDEX` is not set, then consistency checks are disabled.

Best Practice

Installation

- Quick and easy: `conda create -n qa-dkrz -c conda-forge -c h-dh qa-dkrz`
- Full sources: `git clone https://github.com/IS-ENES-Data/QA-DKRZ`
  - Use access to locally provided libraries via the `install_configure` file
- CMIP6 with PrePARE.py: `conda create -n cmor -c conda-forge -c pcmdi -c uvcdat cmor`
- Verify the success of the installation by running `qa-dkrz --example[=path]`, see *Results*.
- Automatic updating is enabled by option `--auto-up` until `--auto-up=d[isable]` is provided; works for both `qa-dkrz` and `install`.

Before a Run

- There is a default for almost every option in the project tables
- Gather frequently changed options in a file and provided on the command-line (-f); this file could also contain QA_CONF=with-user-defined-name.
- Options on the command-line (but -f) only for testing or rechecking small data sets.
- Use the SELECT option to partition the entire data set, e.g. for CORDEX, `SELECT AFR-44=` (note that in this case '=' is required; may-be with prefixed '.*/' depending on PROJECT_DATA) would check the data of every model available for the specified domain. On the other hand, `SELECT .*/historical=orog` would find any orography file in all historical in the given `PROJECT_DATA=sub-dir-tree`.
- Option `CT_PATH_INDEX` defines the name of a consistency table, which is created and utilised to check consistency across sub-atomic files and across parent experiments, if available, of a given variable. This kind of table is consulted for experiments and versions matching the same name.
- Similarly for logfile names by `LOG_PATH_INDEX`

Operational Mode

- Before starting to check data, please make sure that the configuration was set as desired by `qa-dkrz [opts] -e_show_conf`.
- Check the selected experiments: `qa_DKRZ [opts] -e_show_exp`.
- Try for a single file first and inspect the log-file, i.e. run `qa-dkrz [opts] -e_next`, then resume without -e_next.
- Use `nohup` for long-term execution in the background.
- If the script is run in the foreground, then command-line option '-m' may be helpful by showing the current file name and the number of variables (done/selected) on a status line .
- Have a look at the QA results in directory `QA_RESULTS/check_logs/Annotationi/experiment-name.json.`
- Manual termination of a session: if an immediate break is required, please inquire the process-id (pid), e.g. by `ps -ef` , and execute the command 'kill -TERM pid'. This will close the current session neatly leaving no remnants and being ready for resumption.

Results

Running `qa-dkrz --example[=path]`, where omission of `path` would do this in the current directory, generates QA results.

check_logs All results are gathered in this directory.

Besides the log-files gathering the check results for all files, summarised results are given in two modes:

- JSON formatted files and information about atomic time ranges of variables (new and used by default since 2017).
- Formaly, a less machine readable format, which is still used when new results are written to a directory where files of this format are already located. Also, when option TRADITIONAL_SUMMARY is applied.

Log-files (files: experiment-name.log, YAML format) There is an entry for each checked file; possibly with annotations. The option setting of the QA-DKRZ run is put in front.

Annotations (files: experiment-name.json) (for experiment-name see *Configuration*): Experiment-wide information about check conclusion, DRS structure and annotations if any.

Period (files: experiment-name.period in YAML format, time_range.txt) The time range for each variable. Shorter ranges are marked.

Summary (under construction) Directories: experiment-name, contained files are human readable)

experiment-name annotations.txt

- inform about missing variables, if a project defines a set of required ones.
- file names with a time-range stamp that violates project rules, e.g. overlapping with other files, syntax failure, etc. .
- In fact copies of the tag-files describe below.

tag-files a file for each annotation flag found in the corresponding log-file. Annotations include path, file name and variable or experiment scope, respectively, caption, impact level, and the tag from one of the check-list tables.

cs_table (only when option CHECKSUM is enabled) Check sums of files; for the verification that no old file is sold as new.

data internal usage

session_logs internal usage

tables The tables actually used for the run and if option PT_PATH_INDEX is set, then also for the consistency table generated during a check.

CF Conventions Checker

The CF Conformance checker applies to conventions 1.4 -1.7draft (cfconventions.org).

Note: The cf-checkers takes a few seconds for the installation when it runs for the first time.

See *Installation* on how to install this tool. See the available options:

```
dkrz-cf-checker [opts] files(s)

Purpose: Check for CF Conventions Compliance (http://cfconventions.org).
The checker is part of the QA-DKRZ package (https://github.com/IS-ENES-DATA)
```

```
Compilation: '/your-path-to-QA-DKRZ/install CF' unless
the package was downloaded via 'conda create -n qa-dkrz  -c conda-forge -c h-dh qa-
↪dkrz'.
  -C str     CF Convention string; taken from global attributes by default.
  -F path    Find recursively all nc-files from starting point 'path'.
  -p str     Path to one or more NetCDF files.
  -R         Apply also recommendations given by the CF conventions.
  --debug    Show execution commands.
  --help
  --param    Only for program development.
  --ts[=arg] Run the files provided in the Test-Suite for CF Conventions
             rules in QA-DKRZ/CF_TestSuite. If particular NetCDF files are
             provided additionally, then only these are used. If a filename
             cannot be resolved unambiguously, then use optional arg F[ail] or P[ass}.
```

CF Test Suite

A collection of NetCDF files designed to cover all rules of the CF Conventions derived from the examples of the cf-conventions-1.x.pdf documents. There are two branches.

PASS Proper meta-data and data sets, which should never output a failure. Number of files: 81

FAIL Each file contains a (mostly) single break of a rule and should never pass the test. Number of files: 187

The suite is checked entirely by applying option `--ts`. When filenames are additionally passed to the checker, then only these are checked. If such filenames are without a leading path, the checker tries to find out the location in the two branches. If it should be available in both, then provide P or F as argument to `--ts`.

Please, note that running some files from the suite by a different checker may raise additional annotations, because the files are in fact only snippets with a partly missing data segment.

CHAPTER 6

Indices and tables

- genindex
- modindex
- search

Index