# QA-DKRZ Documentation

*Release 0.5*

**hdh**

**Nov 30, 2017**

# Contents

Contents:

CHAPTER 1

# QA-DKRZ - Quality Assurance Tool for Climate Data

**Version** 0.6.7

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 (D. Nadeau, LLNL) is supported. The check results are summarised in json-formatted files.

## 1.1 Getting Started

The recommended way to install QA-DKRZ is by the conda package manager. External tables are downloaded at run-time, when needed for dedicated projects.

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

**Note:** *CMIP6*: The development of the internal CMOR checker of QA-DKRZ is frozen. Instead, `PrePARE` (https://cmor.llnl.gov) is run by `qa-dkrz`. The output of `PrePARE` is merged into the flow of QA-DKRZ annotations.

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

There are different ways to access the QA-DKRZ checker `qa-dkrz`:

- `source [conda-path/]activate qa-dkrz` and run with `qa-dkrz`
- `path/miniconda/envs/qa-dkrz/bin/qa-dkrz`
- `alias qa-dkrz=path/miniconda/envs/qa-dkrz/bin/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 by running `qa-dkrz` for a single NetCDF file. Provision of a project name, e.g. *-P CORDEX*, is required in this simple case. Initially, you'll be asked for a path to QA-TABLES (to put it simply, a directory for external tables; details in *Installation*).

```
$ qa-dkrz -P PROJECT-Name file.nc
```

The QA-DKRZ module for checking CF Conventions is also available stand-alone:

```
$ dkrz-cf-checker [ops] file.nc
```

Running the QA_DKRZ tool requires external tables which are not provided by conda packages. `qa-dkrz` delegates installation and updates to a script, when the first argument is `install` .

```
$ qa-dkrz install [opts] PROJECT-Name
```

## 1.2 Documentation

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

## 1.3 Getting Help

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

### 1.3.1 Mailing list

Join the mailing list ...

## 1.4 Bug tracker

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

## 1.5 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.

## 1.6 License

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

CHAPTER 2

# Frequently Asked Questions

## 2.1 Questions

### 2.1.1 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

## 4.1 0.6

**release-date** 2017

- bash scripts exchanged by python
- annotation summary re-developed (python, stand-alone)
- restructered flow (run/update strictly separated)
- abstraction of DRS checks (same for CORDEX, CMIP5/6)
- PROJECT_AS for checking projects similar to the standard ones
- shipping QA-DKRZ without internet

## 4.2 0.5

**release-date** 2015

- packaged as conda.
- travis continous integration added.
- published docs as sphinx on ReadTheDocs.
- prototype (bash) for summarising annotations.

## 4.3 0.4

**release-date** 2014

- CORDEX meta-data checker added

- CF Convention checker developed
- user control of annotations
- Annotation output by the YAML format

## 4.4 0.3

**release-date** 2012

- CMIP5 checker for meta-data completed

## 4.5 0.2

**release-date** 2010

- prototype of CMIP5 checker for meta-data added
- partial selection from a data pool

## 4.6 0.1

**release-date** 2007

- check of data/time values of NetCDF files
- prototype of bash script for managing the work-flow

CHAPTER 5

# User-guide

## 5.1 User Guide

**Release** 0.5

**Date** Nov 30, 2017

### 5.1.1 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.

The development of a QA-DKRZ CMOR3 check module was frozen with the 2016-ESGF_F2F_Conference (http://esgf.llnl.gov). Since then, the CMOR3 checker `PrePARE` is applied for checking CMIP6 files (http://cmor.llnl.gov/mydoc_cmip6_validator). The output by *PrePARE* is merged into the flow of QA-DKRZ annotations.

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

```
$ qa-dkrz [command-line opts, -f task-file, config-opts] [file.nc]
```

- command-line opts: usually for non-operational usage; mostly prefixed by `--`
- - -help displays description of command-line options.
- task-file: a container with config-options for specific settings
- config-opts: given in project specific files, e.g. CORDEX_qa.conf; provides a default. Every option of PROJECT_qa.conf can be provided on the command-line with the prefix `-e|E[ |_]option`, e.g. -e next. Note that option names are case-insensitive while values are not.
- file.nc: only for a quick test.

#### Available Versions

The QA-DKRZ package is a rolling release, however tagged.

### 5.1.2 Best Practices

#### Installation

- `conda create -n qa-dkrz -c conda-forge -c h-dh qa-dkrz`
- CMIP6 with PrePARE: `conda create -n cmor -c conda-forge -c pcmdi cmor`
- run `qa-dkrz install [opts] PROJECT-name` for downloading/updating required external tables (updating not when frozen).

#### Operation

- run `qa-dkrz -f task-file` with the following options contained in a task_file:
  - **PROJECT_DATA** path to the root of the data tree.
  - **QA_RESULTS** path to results (created by qa-dkrz) containing log information.
  - **DRS_PATH_BASE** directory name indicating the beginning of the DRS sub-path within the data path. May be omitted, when it equals the name of the project in capital letters. If your data path does not match a DRS pattern, set also LOG_PATH_INDEX and LOG_FILE_INDEX depending on your requirements. This will determine the name of the log-file in QA_RESULTS/check_logs.
  - **SELECT** partitioning of a data directory-tree. Usually starting with the next one behind PROJECT_DATA; regular expressions enabled.
  - **NUM_EXEC_THREADS** number of asynchronous, parallel proccesses; each for an atomic variable.
  - **EMAIL_SUMMARY=name@site.xy** a summary of the results are emailed.
  - **QA_CONF=PROJECT_qa.conf** default setting of options for a given project.
- Before starting to check data, please make sure that the configuration was set as expected by running `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 -f task-file] -e_next`. If accepted, then resume without `-e_next`.

- Files having checked before are skipped when they are scheduled again for a check. Use option -e_clear=note for rechecking files causing annotations.
- 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 that .
- Have a look at the QA results in directory `QA_RESULTS/check_logs`; annotations are summarised in sub-directory `Annotation/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 | grep qa-dkrz`, and execute the command 'kill -TERM pid'. This will close the current session neatly leaving no remnants and being ready for resumption.

#### General remarks:

- Projects taken into account are CORDEX, CMIP5/6, HAPPI.
- There is a proceeding with tables:
  - QA-DKRZ based tables, which are taken by default, reside in `QA-DKRZ/tables/projects/PROJECT`. Note that changes would be lost after an update, regardless of whether by `conda` or `git`.
  - the default tables as well as external tables are copied to the QA-TABLES path given in HOME/.qa-dkrz/config.txt or asked initially.
  - user-define adjustments have to be placed in QA-TABLES/tables/PROJECT, where PROJECT_DRS_CSV.csv has to be copied entirely and adjusted, while PROJECT_check-list.conf and PROJECT_qa.conf may only contain adjusted statements from the respective default table.
  - all tables required for a particular run are copied to QA_RESULTS/tables. Depending on the QA_RESULT managment, these may be overwriten when changed. The checking program will read from here.
- If the recommendations above are sufficient for checking your project, you may skipp reading from here.
- If you would like to check something similar to supported project, then you may try option PROJECT_AS and copy/adjust main-project tables to the user-defined tables.
- If the machine you're on does not have the ability to run the code, then you would like to download from GitHub and compile the executables.
- QA-DKRZ may be used in multi-user and multi-tasking mode. If everybody should use the same configuration, then `install --freeze` would be an option. Otherwise, each user may have his own `HOME/.qa-dkrz/config.txt` and external tables.

### 5.1.3 Installation

*QA-DKRZ* may be installed either by the `conda` package manager or from a GitHub repository. Recommended is `conda` installing a ready-to-use package (a 64-bit processor is required). If you want to work with sources or use a different machine architecture, then the installation from GitHUb should be the first choice.

Running the QA_DKRZ tool requires external tables which are not provided by conda packages (neither by QA_DKRZ from GitHub). `qa-dkrz` starts a script for installation and updates, when the first argument is `install` .

For CMIP6, *QA-DKRZ* binds the CMOR validator *PrePARE* (D. Nadeau, LLNL, https://cmor.llnl.gov). The CMOR package is downloaded by

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

Details in section Work-flow below.

No matter what the preference is *conda* or *git*, operational and update processes are separated.

---

**Note:**

- `qa-dkrz:` just for runs. While the completeness of required tables is checked, but the state whether tables / programs are up-to-date is not. If this check fails, the user is asked to run the command below.
- `qa-dkrz install [opts] PROJECT:` get/update external tables and, when installed from GitHub, also compile executables. If run initially, you are asked for a path to put external tables. The user is disengaged from knowing table names or their internet access.

---

### Conda Package Manager

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

```
$ search and download Miniconda-latest-Linux-x86_64.sh
$ bash Miniconda-latest-Linux-x86_64.sh
```

Have the `conda` in your `PATH` or execute it by `path/miniconda/bin/conda`. *Conda* provides the QA-DKRZ package with all dependencies included. It is recommended to use conda's default for named environments found in `path/miniconda/envs` .

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

### 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. If `install_configure` is not available, a template is created and the user is asked to edit the file.
- Environmental variables CC, CXX, CFLAGS, and CXXFLAGS are accepted.
- `qa-dkrz install` establishes access to libraries, which may be linked (recommended) or built (triggered by option `--build`. Note that this is rarely tested of the courwse of development).
- Proceedings are logged in file `install.log`.
- The executables are compiled for projects named on the command-line.

- If `qa-dkrz install` is started the first time for a given project, then corresponding external tables and programs are downloaded to directory *QA_TABLES*; the user is asked for the path.
- The user's home directory contains a config.txt file in directory .qa-dkrz by default, created and updated by `qa-dkrz install`.

The full set of options is described by:

```
$ qa-dkrz 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

```
$ qa-dkrz install --build [opts]
```

Sources of the following libraries are downloaded and installed:

- zlib: www.zlib.net,
- hdf5: www.hdfgroup.org,
- netcdf-4: www.unidata.ucar.edu (shared, no FORTRAN, non-parallel),
- udunits: http://www.unidata.ucar.edu/packages/udunits.
- uuid: mostly provided by the operating system.

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.

### 5.1.4 Update

Updates of tables are downloaded from external sources by

```
$ qa-dkrz install [opts] PROJECT-Name
```

with options:

**PROJECT-name (as single parameter)** installation of external tables for the named project. Additionally, compilation of executables for PROJECT, but only for GitHub based installation.

**- -force [PROJECT]** Unconditional update. Also required to unlock a frozen state.

**- -freeze** Lock current installation; *HOME/.qa-dkrz/config.txt* is copied to the root directory of QA-DKRZ. This is applied by every user of this particular installation and can not be modified.

**- -up [PROJECT]** Updating tables only when there was no update on that day before. The frequency may be prolonged by option `--uf=num` with `num` in days.

**- -ship** Prepare a working installation for transferring it to internet-free devices. (in preparation)

**- -shipped** internet-free installation. (in preparation)

### 5.1.5 Work-flow

Please note that the work-flow has changed. By now, operational usage is generally distinct from installation or updating. However, users may trigger that tables are updated regularly (by default daily). But, conda built packages are updated only manually. The work-flow is generally as:

**Installation/update**

- download the QA-DKRZ package (conda or GitHub)
- CMIP6: download the CMOR3 package (both conda and GitHub)
- run `qa-dkrz install [opts] PROJECT` : download external tables (both conda and HitHub), compilation of executables (only GitHub)
- file HOME/.qa-dkrz/config.txt is created

The paths to the `qa-dkrz install` script (bash) is:

***conda-built*** path: miniconda/env/qa-dkrz/opt/qa-dkrz/install

***GitHub*** path: QA-DKRZ/install

Paths and times of next updates are kept in a configuration file located in *HOME/.qa-dkrz/config.txt* by default. It is not necessary for a user to edit it, but permitted. It ismanaged by `qa-dkrz install` calls.

**Operation**

The names of the operational executables are different between conda-built and GitHub-based packages, because of backward-compatibility. However, all work (almost) the same.

```
*conda-built*
 qa-dkrz:    python script, path= miniconda/env/qa-dkrz/bin
 qa-dkrz.sh: bash script,   path= miniconda/env/qa-dkrz/bin

*GitHub*
 qa-dkrz.py: python script, path= QA-DKRZ/python/qa-dkrz/qa-dkrz.py
 qa-dkrz:    bash script,   path= QA-DKRZ/scripts/qa-dkrz
```

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

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

with task-file containing frequently modified 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 and explained in file `QA_DKRZ/tables/projects/PROJECT/PROJECT_qa.conf` with `QA_DKRZ` indicating the root of the QA-DKRZ package. Additionally, every valid option may be provided on the command-line by prefixing `-e|E[ ]`.

Note that option `RUN_CMOR3_LLNL` must be given for running *PrePARE*, however is set for CMIP6 by default .

The separation of operation and install/updating is superseded by option `--auto-up` for backward-compatibility and convenience, respectively.

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

### 5.1.6 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 configure 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 directory-tree. This can be further customised by 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' or '-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 path/filename on the command-line: cancellation of previous SELECTions.
- 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.
- LOCKing gets the higher precedence over SELECTion.
- SELECT: Path and variable items 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 particular 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 QA_DKRZ root path.

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

Multiple configuration files and options may be specified following a given precedence. This facilitates having a file with short-term options (in a file provided by 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. 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:

All options may also be provided on the command-line plus some more for testing, see `/package-path/QA-DKRZ/scripts/qa_DKRZ --help`.

#### Experiment Names

QA-DKRZ checks files individually. The results are contained in files 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.

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 point with index=0, e.g. `DRS_PATH_BASE=output` and `LOG_PATH_INDEX=1,2,3,4,6`. An example is given in *Results*.

Similar is for building the name of a consistency table, which is used to check consistency between a parent and its child experiment as well as between the temporal sequence within an atomic variable.

---

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

---

### 5.1.7 Results

Running `qa-dkrz` generates QA results with a few directories.

**check_logs** All results are gathered in this directory.

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

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

**Log-files (files: experiment-name.log, YAML format)** Files with applied QA-DKRZ options as preamble followed by entries for every checked file; possibly with annotations. The file name is composed corresponding to options DRS_PATH_BASE and LOG_PATH_INDEX, respectively LOG_FILE_INDEX; see see *Configuration*.

**Annotations (files: experiment-name.json)** Experiment-wide information about check conclusion, DRS structure and annotations if any. Each annotation refers DRS path components, and severeness levels (if any).

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

**Summary** (at present only for qa-dkrz.sh)

**experiment-names** Directories: experiment-name, contained files are human readable)

**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.

### 5.1.8 CF Conventions Checker

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

The checker is a bash script and resides in

**conda-built** path=‘‘miniconda/env/qa-dkrz/opt/qa-dkrz/scripts‘‘

**GitHub** path=‘‘QA-DKRZ/scripts‘‘

```
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-1.6.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

## Q