NOTE 262 – Casacore Testing Framework

Ger van Diepen, ASTRON Dwingeloo

2013 Aug 21

Abstract

Casacore has an advanced testing framework on top of the CMake environment. It makes it possible to do regression testing, coverage testing, and use tools like valgrind in an automatic way.

Contents

1 Introduction

Casacore comes with an extensive set of test programs. The tests can be built in the cmake environment used by casacore by adding `-DBUILD_TESTING=ON` to the `cmake` command. Running the tests can be done using the `make test` or `ctest` command. For regression testing purposes the standard output of a test run can be compared with the expected output. Casacore uses a few scripts doing the actual execution of a test.

A test be embedded in a script for optional preprocessing or postprocessing of a test or to run a test in multiple ways.
Such a script can also be used to test an application (like `taql`).

2 Casacore build system

Casacore uses the cmake system to build the code and test programs. Ctest is used to execute the test programs and to check for success or failure.

2.1 Source code structure

The casacore source code is divided into modules (e.g. casa, tables). Each module is divided into packages (e.g. casa/IO) containing the source files. Each package should have a test directory containing the test programs. Schematically it is organized like:

```
casacore
  module1        #
    package1
      *.h
      *cc
      test
        tX.cc
```

The build system creates a library per module (e.g. `libcasa_tables`). Modules are layered, thus the libraries not mutually dependent. Packages in a module can use each other, also mutually.

2.2 Build directory structure

It is best to build casacore in a `build` directory at the same level as the `casacore` directory. In this way the source and build files are well separated.

The code can be built in various ways (optimized, debug, etc.), so in the build directory a directory should be created telling the build type. The build system makes uses of the name to automatically set the compile options for a build type. It recognizes:

- `dbg` or `debug` defining a debug build (sets `-O0 -g`).
- `cov` defining a coverage build (sets `--coverage`).
- Any other name defines an optimized build (sets `-O3`).

In this directory the source code structure is mirrored, thus it will have directories like `casa/IO/test`.

3 Creating a test

Casacore consists of several modules (e.g. `scimath`, `tables`). A module can consist of one or more packages (e.g. `casa/IO`, `casa/OS`). Each package contains one or more classes, usually one class per file. Test programs should be created in the `test` directory of a package.

Usually a test for class X is written in C++. The casacore convention is to name such a test tX, thus the source code for a test is in `<module>/<package>/test/tX.cc`. In order to build the test program, it needs to be added to the `tests` list in the `CMakeLists.txt` file in that test directory.

The test executable is created in the build directory `build/<type>/<module>/<package>/test`.

Besides the .cc file, a few more files can be created for a test. They also need to reside in the source test directory. When using a standard name as in the list below, they will be copied automatically to the build test directory when the test is run.

- `tX.run` is a script. It can contain commands to do some preprocessing, postprocessing, to run the test in different ways, etc. Note that such a script is usually a shell script, but could also be a python script or any other executable script. If the script is not executable, it is run through `sh`.
  Note that the invocation of the test program in a script should in principle be preceeded by `$casa_checktool`. This environment variable defines an optional check tool (like valgrind) as described in a next section.
  Also note that in a script a test program should be run as `./tX`, because . does not need to be part of the PATH.

- `tX.in` is a text file giving test input lines. If existing, the standard input of the test program (or script) is redirected to this file.

- `tX.out` is a text file containing the expected standard output of the test. It is very useful for regression testing. The next section discusses how it is used.

- `tX.in_*` are other inputs that can be used by the test. They can be simple files, but also tar-files (which can be unpacked in the .run file) or directories (e.g., casacore tables).

Note that to run test tX, the test system will only copy files/directories as named above to the build test directory. After the test they will be removed from the build test directory. If test input files are named differently, they can be used in one of the two following ways.

- For each test the test system defines the environment variable `testsrcdir` as the source directory of that test. It can be used in test programs or scripts to copy a test file from the test source directory or to use it directly. If using it directly, it should (of course) be used readonly.

- Another option to copy files from the source to the working directory is by adding appropriate lines to `test/CMakeLists.txt` like:

```
set (datafiles
amp_0.fits
ampErr_0.fits
)
```

```
foreach (file ${datafiles})
    configure_file (${CMAKE_CURRENT_SOURCE_DIR}/${file} ${CMAKE_CURRENT_BINARY_DIR}/${file
endforeach (file)
```

to let the cmake system copy the file to the working directory. An entire directory cannot be done in a single line; each individual file has to be named.

If the test program or script creates output files, it is best to call them `tX_tmp<something>`. In this way they will be removed automatically after the test is run.

4 Running tests

The standard way to run a test is using `ctest`. It makes it possible to run all or some tests.

```
ctest                    # all tests
ctest -R tArray          # only tests matching *tArray*
ctest -E tArray          # exclude tests matching *tArray*
```

If such a command is run in the casacore top build directory, all casacore tests are considered. However, it is also possible to run it in a subdirectory (e.g. casa or casa/IO/test) to execute only tests of a specific module or package.

The system runs a test as follows:

1. `ctest` starts the script `cmake_assay` to run the test.
2. `cmake_assay` copies the tX files mentioned in the previous section to the build test directory where the test is run. Thereafter it starts `casacore_assay` to run the actual test.
3. `casacore_assay` executes the test as follows:
   - If `tX.run` exists, that script will be executed. otherwise `tX` itself.
   - If `tX.in` exists, it will be redirected as input to tX.
   - If `tX.out` exists, it will be compared with the standard output of the test. The comparison is not done if the test exits with an error status. Before the comparison is done the output is massaged as follows to cater for system specific output.
     - The working directory is removed from the test output.
     - Text enclosed in `>>>` and `<<<` on a single line is removed from the test output and the expected output.
     - Text demarked by lines starting with `>>>` and `<<<` is removed from the test output and the expected output.
     - If a diff of the resulting test output and expected output finds no differences, the test passes. If any non-numerical value mismatches, the test fails.
     - Otherwise `casacore_floatcheck` is run to compare all numerical values with a relative error margin of 10e-5. The test fails if a value is found unequal.
   - At the end of the test the `tX_tmp*` files/directories and the copied tX files are removed by `casacore_assay`.

4.1 Return status

The return of a test program or script can be:
- 0 indicating success.
- 3 indicating that the test is skipped. This might be used if an expected input file is not available or if the test is run on a platform not supporting it. `cmake_assay` will change this to 0, because ctest can only deal with success and failure.
- Any other value indicates a failure.

5 Valgrind support

If the environment variable CASACORE_CHECK has the value 1, the `casacore_assay` script runs the test of an executable through `casacore_memcheck`. This script runs the test through valgrind's memcheck tool and tests for errors, definite and possible leaks, and leaked file descriptors. If any such problem is found, the valgrind output is put into a file `tX.checktool.valgrind` in the working directory.

If a .run file is used to execute the test, each invocation of the test program in the .run file should be preceeded by `$casa_checktool`. This environment variable defines the valgrind command. If valgrind is not used, the variable is empty.

5.1 Using other checking tools

The same mechanism can be used to use a check tool different from valgrind's memcheck.
CASACORE_CHECK can be defined as the name of a script or a command to use. In fact, defining CASACORE_CHECK as `casacore_memcheck` would have the same effect as defining it as 1.

5.2 Omitting tests in check tool

Sometimes a test run with a check tool can take too much time and should be omitted. There is no direct way to tell that a specific test should be omitted. However, it can easily be achieved by having a .run file for such a test without the `$casa_checktool` prefix.

6 Test coverage support

The casacore build system has support for test coverage analysis. By building in the directory `build/cov` the code is automatically built this way. Currently, only the g++ compiler is supported. Support for clang will be added later.
The script `casacore_cov` (in casacore/scons-tools) should be used to run the tests, analyze the test output, and generate coverage reports from it using `lcov` and `genhtml`. It works as follows:

- The coverage counters are initialized.
- All tests are run using `ctest`.
- Coverage results are assembled with `lcov` and converted to nice html pages using `genhtml`. The root page is ./cov/index.html.

- Both code and branch coverage info is generated for the module code, thus test program code is removed. Function coverage info is not generated, because the tools cannot cope well with functions automatically generated by the compiler.
- The file `testcov.log` contains a log of the script's output.
- The script can be run at various levels.
  - If run in build/cov it will run all tests and generate test coverage for all modules.
  - If run in e.g. build/cov/casa, it will generate test coverage for module casa.
  - If run in e.g. build/cov/casa/IO/test, it will generate test coverage for package IO of module casa.