cr2010_05_26, September 11, 2013, (11.09.2013), sebastian.rast@zmaw.de

revisions: 3283 or higher

1 Systematic technical testing of ECHAM6

In many cases, scientists wish to modify the ECHAM6 code for their special applications. Before any "production" simulation can be started, the modified ECHAM6 version has to be tested thoroughly. The purpose of this collection of korn shell scripts is to provide a systematic and easy to use test bed of the ECHAM6 code on a technical level. These test scripts perform very short simulations in the T31L47 resolution over 12 time steps in different model configurations in order to trap errors in the code that cause technical malfunctions. However, this kind of tests can not detect any scientific failure or evaluate the scientific quality of the results. The tests rely on a comparison of the output of 12 time steps using the `cdo diff` tool. We apply the term that the results of two simulations are "bit identical" if the `cdo diff` command does not find differences between all netcdf or GRIB output files of these two simulations. This means that the output on the standard output device of these two simulations is allowed to be different, e.g. by new messages for a newly built in submodel facility. Futhermore, it is only checked whether the netcdf representation of the output of the two simulations is bit–identical but not whether all variables during the run of the ECHAM6 program have bit identical values in both simulations. In addition to tests on one model version that will be called the test version, such a test version of the model can be compared to a reference version in the so–called update test.
The package of scripts performing these tests can be used on various computers without queueing system and can be modified in such a way that individual namelists and input data can be provided to the test and reference model.
The following tests and combinations of them can be performed by the test tool (including checkout and compilation of the model which is always performed):

compile: This is not a real test. The respective test version is checked out from the svn version control system if necessary and compiled, but no run is performed.

single test: In this test, the test version is (checked out, compiled, and) run for 12 time steps. The test is successful if the program does not crash.

parallel test: For this test, a simulation of the test ECHAM6 version over 12 time steps is performed on 1 and 2 processors, respectively, and the result is compared by the `cdo diff` tool for every time step. The test simulation on a single processor is also performed using the parallel mode of the program. It is therefore not a test for the version of ECHAM6 without message passing interface (mpi). With this kind of test, possible parallelization errors can be detected like the usage of variables or fields which were not sent to all processors. The result of these two simulations should be bit identical. On massive parallel machines, using a lot of processors distributed over several nodes further problems may occur even if this test is passed. Such problems are often either subtle errors in the usage of mpi or compiler problems. Supplemental tests have to be performed on a later stage when the program is ported to such a platform.

nproma test: The section of the globe that is present on a processor after distribution of the data onto the processors, is vectorized by blocks of maximum length `nproma`. This means that — even if only one processor is used — surface fields of the earth do not simply have two dimensions of the size of longitudes $n_{\rm lon}$ and latitudes $n_{\rm lat}$ but are reshaped to `ngpblks` blocks of maximum length `nproma`. Since `nproma` may not be a divisor of $n_{\rm lon} \times n_{\rm lat}$, there may be a last block that contains fewer than `nproma` elements. This may lead to problems in the code, if such non–initialized elements of the last block are used accidentally. The nproma test traps such errors by using two different nproma lengths of 17 and 23 which are both not divisors of $n_{\rm lon}$ in the T31 resolution in the test simulations and comparing the results of 12 time steps. The results should be bit–identical.

rerun test: ECHAM6 has the possibility to split up a long term simulation into several runs of a shorter time period and to restart the model at a certain date. The results after restart are bit identical with those of a simulation without restart. There is a large variety of errors associated with a failure of the restart facility which can not all be trapped by this test like wrong scripting of the use of transient boundary conditions, but to pass this test is a minimum requirement. The base simulation starts at 1999-12-31, 22:00:00h, writes a restart file at 23:45:00h. It stops after a total of 12 time steps. The rerun files are used to restart the program and to complete the 12 times steps. The five time steps after restart are then compared with the simulation that was not interrupted. The results should be bit–identical.

update test: This test compares the results of two simulations with different model versions (test version versus reference version). Under certain circumstances, bit–identical results may be required in this test.

submodel off test: The above standard tests are all run in a model configuration that comprises submodels (configuration similar to the CMIP–5 simulations). In some cases, one may be interested in a configuration without any submodel. This test tries to run `ECHAM6` without any submodel. If two revisions are compared, the results of this model configuration are also compared for the test and reference revision.

1.1 System requirements

The `ECHAM6` test scripts can be adapted to UNIX computers without queuing system. The automatic configure procedure for the model compilation has to work and the environment has to provide the possibility to run programs using message passing interface (mpi). The initial and boundary condition data of `ECHAM6` have to be directly accessible in some directory. If there is no direct access to the version control system of echam (svn), individual model versions on the computer may be used in the tests, but the path name of the location of these model versions has to follow the below described conventions.

1.2 Description of the scripts

In figure 1, we present the flow chart of the scripts performing the test simulations of `ECHAM6` and the comparison of the results. The scripts need some additional variables that are written to files by the master script `test_echam6.sh` and read from these files by the dependent scripts. The variables can be set in the master script as described in Tab. 1. The corresponding files must not be modified by hand. The file `c.dat` contains the module name of the C compiler, the file `fortran.dat` contains the module name of the fortran compiler, the file `mpirun.dat` contains the absolute path and name of the command to start programs using message passing interface (mpi), the file `outfiletype.dat` contains a number associated with the type of the output files (1 for GRIB format and 2 for netcdf format).

test_echam6.sh: This script contains a definition part where all the path names and the model version for the test and reference model must be set. It is also the place at which the key word for the kind of test is defined. It calls the scripts for downloading the respective model versions from svn if they are not yet present on your computer and calls the compile and test run scripts.

test_directories.sh: Checks the existance of the directories and the svn URL of the test and reference model. You may enter your special settings on the command line if one of the items is not found. If it is found, it is used without further notice. Only the relevant items are checked.

compile_echam6.sh: This script downloads the respective model version from the revision administration system svn if it is not yet present on your computer and compiles the model. Compilation can be forced. Note that the compiler options depend on the settings in the input scripts of the configure procedure and may be different from revision to revision. Different compiler options may lead to numerically different results although the algorithms in the code are identical!

test_*mode*.sh: This family of scripts performs the various simulations and the comparison of the results. The *mode* is one of `single`, `parallel`, `nproma`, `rerun`, `update`, `submodeloff`, `parallelnproma`, `parallelnpromarerun`, `parallelnpromarerunsubmodeloff`, `all`.

test_echam6_run.sh: General run script for echam.

test_echam6_{test,reference}_links.sh: Script that provides the links to all input and boundary condition files needed for simulations with `ECHAM6`. In the standard version, the two scripts are identical but allow the user to apply different files for the reference and test model, respectively.

test_echam6_{test,reference}_namelists.sh: These scripts generate the namelists for the reference and test model separately. In the standard version, these two scripts are identical. They are useful if the introduction of a new submodel requires a namelist for the test model that is different from the namelist used for the reference model.

test_diff.sh: This script performs a comparison of all output files that are common to two test simulations. It also gives a list of outputfiles that are not common to the two test simulations. If there are no results written into an output file during the 12 time steps of the test simulations, the comparison of the files with the `cdo diff` command leads to an error message that the respective file structure is unsupported.

1.3 Usage

The scripts should be copied into a directory that is different form the original ECHAM6 directory so that you can savely change them without overwriting the original. The files `*.dat` must not be changed but contain values of "global" variables to all scripts. They are described in section 1.2. The variables that have to be modified in `test_echam6.sh` are listed in table 1. Note that the revision specific path of the ECHAM6 model will be automatically composed as `${REF_DIR}/${REF_BRANCH}_rev${REF_REVISION}` for the reference model and as `${TEST_DIR}/${TEST_BRANCH}_rev${TEST_REVISION}` for the test model, respectively. Inside these directories, the echam model sources are expected to be in a revision independent directory `${REF_BRANCH}` and `${TEST_BRANCH}`, respectively. The simulation results will be in directories `${REF_ODIR}/0000nrev${REF_REVISION}` and `${TEST_ODIR}/0000nrev${TEST_REVISION}` for the reference and test model, respectively. The number `n` is the number of the experiment. If in such a directory, an outputfile `*.err` exists, the test tool assumes that the simulation already exists and does not perform a new simulation. The results are not removed once a test is performed in order to avoid the repetition of the same test simulation over and over again (e.g. for the reference model). If experiments have to be repeated, the corresponding directory or at least the `*.err` file inside this directory has to be removed by hand.
The test is then started by typing `./test_echam6.sh` in the directory of the test scripts.
The test script `test_echam6.sh` can be started by adding three arguments `MODE TEST_REVISION REF_REVISION` giving the key word for the test mode, the revision number of the test model and the revision number of the reference model, respectively. It is possible to omit `REF_REVISION` or both `TEST_REVISION` and `REF_REVISION`.
The links to input and boundary condition data and the input namelists for the model revisions can be modified for the reference and the test model individually by editing the scripts `test_echam6_{reference,test}_links.sh` and `test_echam6_{reference,test}_namelists.sh`, respectively. This makes this collection of test scripts rather flexible: It may be used even for models containing extensions of ECHAM6 like ECHAM6-HAM or ECHAM6-HAMMOZ.

Table 1: Variables of `test_echam6.sh` that have to be modified by the user of the test scripts. The variables are listed in the order of their appearance in `test_echam6.sh`. Note that the revision specific path of the ECHAM6 model will be automatically composed as `${REF_DIR}/${REF_BRANCH}_${REF_REVISION}` for the reference model and as `${TEST_DIR}/${TEST_BRANCH}_${TEST_REVISION}` for the test model, respectively.

Variable	Explanation
`SCR_DIR`	Absolute path to diretory where test scripts are located.
`OUTFILETYPE`	File type of output files. Set to 1 for GRIB format output files and to 2 for netcdf output files. It is recommended to test ECHAM6 with both output formats.
`FORTRANCOMPILER`	If a module has to be loaded in order to use the correct fortran compiler version, give the fortran compiler module here.
`CCOMPILER`	If a module has to be loaded in order to use the correct C compiler version, give the C compiler module here.
`MPI_MODULE`	If a module has to be loaded in order to use the Message Passing Interface (MPI) runtime environment, specify the module here.
`MPIRUN`	command specification to run a program using MPI. When running, the script will replace `%n` and `%x` by the number of processes and the name of the executable, respectively.
`TEST_DIR`, `REF_DIR`	Absolute base path to directory containing model versions of test and reference model, respectively. Even if the model source code is loaded from svn, this directory has to exist.
`TEST_BRANCH`, `REF_BRANCH`	name of branch of test and reference model in the revision control system svn a revision of which has to be tested, respectively.
`TEST_REVISION`, `REF_REVISION`	revision number of test and reference model revision, respectively.

table continued on next page

Table 1: test_echam6.sh — continued

TEST_SVN, REF_SVN	URL address of test and reference model branch in svn system, respectively. Can be omitted if model source code is on local disk.
TEST_ODIR, REF_ODIR	Absolute path where test scripts can open directories for simulation results of test and reference model, respectively. This directory has to exist.
LCOMP	LCOMP=.true. forces compilation, with LCOMP=.false. compilation is done only if executable is not existing.
MODE	One of compile, single, parallel, nproma, rerun, update, submodeloff, parallelnproma, parallelnpromarerun, parallelnpromarerunsubmodeloff, all in order to perform the corresponding tests.

If some step or test was not successful, more information about the possible error is given in the protocol files that are written for each step. If the model was checked out from the svn system, there is a protocol file checkout.log of the checkout procedure in ${REF_DIR}/${REF_BRANCH}_${REF_REVISION} for the reference model and ${TEST_DIR}/${TEST_BRANCH}_${TEST_REVISION} for the test model, respectively. The configure procedure and compilation is protocolled inside the ${BRANCH} directory of the aforementioned paths in the files config.log and compile.log, respectively. Information about each simulation can be found inside the directories ${REF_ODIR}/0000nrev${REF_REVISION} and ${TEST_ODIR0000nrev${TEST_REVISION} with n being the number of the test case indicated during the test run procedure on the screen, respectively. In these directories, the standard and standard error output of the ECHAM6 program can be found in the 0000nrev${REF_REVISION}.{log,err} and the 0000nrev${TEST_REVISION}.{log,err} files, respectively. The detailed result of the cdo comparison for each output file is also in these output directories in respective files diff∗.dat. On the screen, only the most important steps and results are displayed. A certain test is successfully passed if the comparison for each file results in the message "0 of *r* records differ" where *r* is the number of records.

pn: 2010/05/20 (echam-Test)

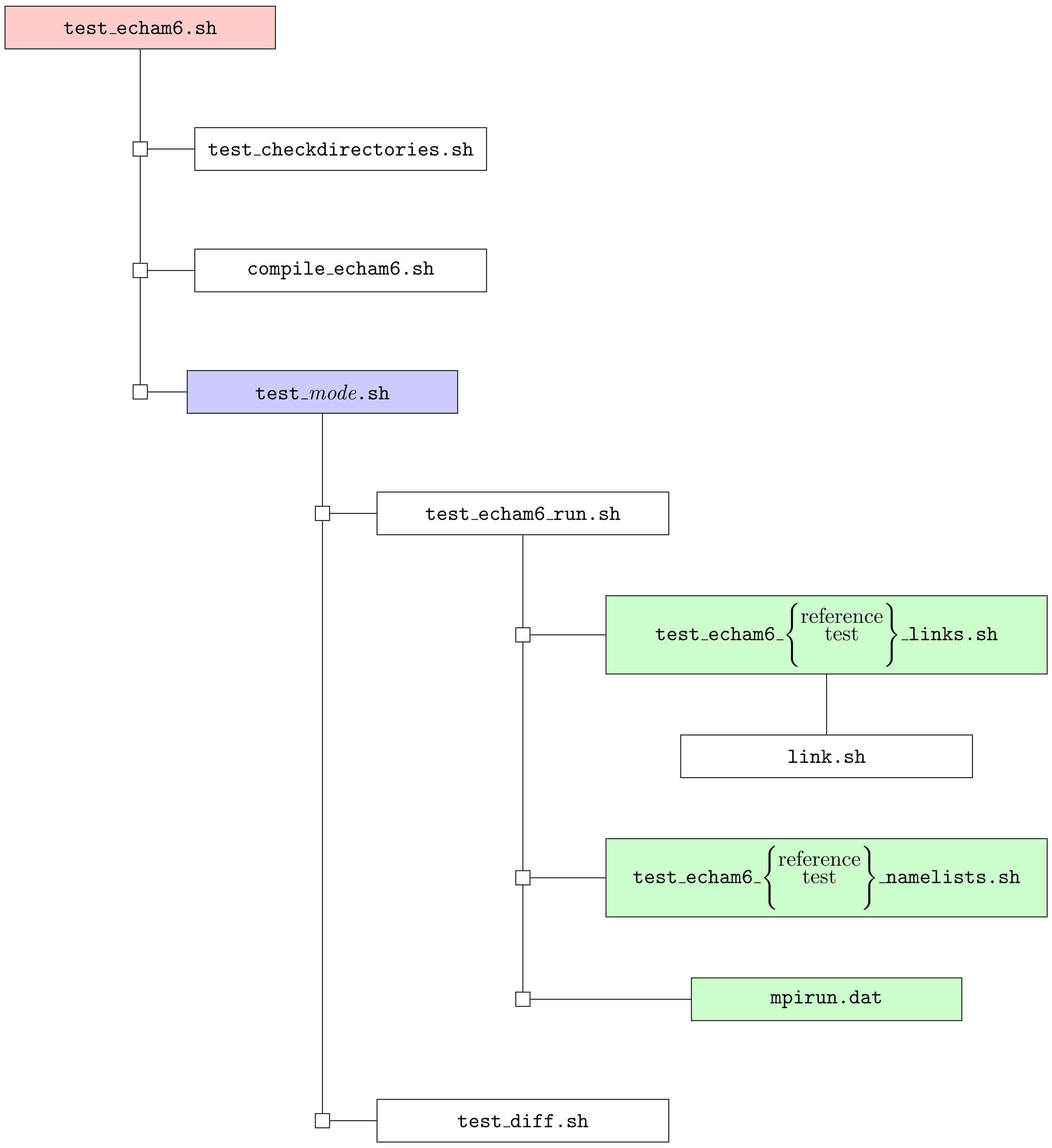

Figure 1: Flow chart of test scripts. The main script in the red box has to be modified by the user. The scripts in the green boxes can be modified in order to use different model settings than the standard ones for test or reference model, respectively. The script in the blue box depends on the test mode and is one of *mode*=`single`, `parallel`, `nproma`, `rerun`, `submodeloff`, `parallelnproma`, `parallelnpromarerun`, `parallelnpromarerunsubmodelloff`, `update`, `all`.