

TNO report

TNO-060-UT-2012-00109

LOTOS-EUROS v1.8 User Guide

**Earth, Environmental and Life
Sciences**

Princetonlaan 6
3584 CB Utrecht
P.O. Box 80015
3508 TA Utrecht
The Netherlands

www.tno.nl

T +31 88 866 42 56

infodesk@tno.nl

Date	January 19, 2012
Author(s)	<i>Dr. Ir. A.J. Segers</i>
No. of pages	50 (incl. appendices)
No. of appendices	0
Sponsor	NMDC - Nationaal Modellen- en Data Centrum
Project name	NMDC / Leerproject LOTOS-EUROS
Project number	034.23888

All rights reserved.

No part of this publication may be reproduced and/or published by print, photoprint, microfilm or any other means without the previous written consent of TNO.

In case this report was drafted on instructions, the rights and obligations of contracting parties are subject to either the General Terms and Conditions for commissions to TNO, or the relevant agreement concluded between the contracting parties. Submitting the report for inspection to parties who have a direct interest is permitted.

© 2012 TNO

Contents

1	Introduction	5
1.1	What can be found in this guide?	5
1.2	Raw documentation	5
1.3	Automatic compilation of this document	5
2	History	7
2.1	Release 1.8.000	7
3	Version control system	9
3.1	Introduction	9
3.2	Subversion server	9
3.3	Checkout a copy of the source	9
3.4	Managing source files with subversion	9
3.5	Daily export of latest code	10
4	Source files	11
4.1	Identification of a model version	11
4.2	Directory structure	11
4.3	Patch directories	11
4.4	The concept of source projects	12
5	Running LOTOS-EUROS	13
5.1	Quick start	13
5.2	Version directory	13
5.3	Run scripts	13
5.4	The rc-file	14
5.5	Setup a run	15
5.6	The run directory	16
5.7	Preprocessing of the rc-file	16
5.8	Collection of a source code	16
5.9	Generation of source files	16
5.10	Creation of the Makefiles	17
5.11	Compilation	17
5.12	Submit script	17
5.13	Setup and submit in a single step	18
6	Input data	19
6.1	Minimum package	19
7	Generation of chemistry code	21
7.1	Overview	21
7.2	Tracer and chemistry properties	21
7.3	Tracer table	21
7.4	Reaction table	22
7.5	Specification of table files	23
7.6	Tracer indices and arrays	23
8	Pre-processor macro's	25
8.1	Example of usage	25
8.2	Macro definition include files	25
8.3	User settings	25
8.4	Expert settings	26
9	Horizontal grid	27
9.1	Grid definition	27
9.2	Zooming	27

10	Time steps	29
11	Model output	31
11.1	Gridded output	31
11.2	Satellite validation output	33
11.3	Observation simulation	33
11.4	Emission summary	33
12	Run time	35
12.1	Timing	35
12.2	Parallelization	36
13	Post processing and visualization	39
13.1	DIADEM	39
13.2	GrADS	41
14	Development tools	43
14.1	TTB - The Test Bench	43
15	Coding conventions	45
16	Signature	49

1 Introduction

1.1 What can be found in this guide?

This is the User Guide of the LOTOS-EUROS model. The purpose of this guide is to provide a user information on how to:

- obtain a copy of the source code;
- compile and setup a simulation;
- run the model.

For information about the physical processes and parameterizations used in the model we refer to the '*LOTOS-EUROS v1.8 Reference Guide*'.

The model has been validated as described in the '*LOTOS-EUROS v1.8 Validation Report*', available through the website too. See chapter 2 for the changes in the model after the validation simulation were performed.

An electronic version of this report, as well as the Validation Report, the Reference Guide, and other documentation are available through the model website:

`www.lotos-euros.nl`

1.2 Raw documentation

The original text of this User Guide is written in LaTeX format. The files are included in the LOTOS-EUROS distribution:

```
lotos-euros/v1.8/doc/user-guide/tex
```

A pdf version can be created with:

```
make pdf
```

Similarly, a browsable version can be created with:

```
make html
```

Use 'make help' for other formats and the location of produced files.

1.3 Automatic compilation of this document

At some institutes the User Guide is compiled automatically every night from the latest source files. Known locations:

- TNO: `${LE}/doc/user-guide`

See section 3.5 for details on how this service is implemented.

2 History

2.1 Release 1.8.000

Release 1.8.000 of the model is the result of the following codes.

- The start was the latest patch of the previous version: v1.7.009 . The '*LOTOS-EUROS v1.8 Validation Report*' is based on simulations with this patch.
- During the validation process some deficiencies were discovered for which projects were added to the source tree of v1.7 . The changes cause small differences with respect to the validation experiments, but since it concern clear improvements they were included in the new release:
 - Improved re-gridding of MACC fire emissions also to zoom resolutions.
 - Changed threshold to distinguish land-cells from sea-cells when setting the sea-salt particle size.
 - Improved automatic setting of operator splitting time step (section 10)
 - Improved re-gridding of average soil-water field used for dust emission from to traffic re-suspension.
 - Fixed bug in vertical re-gridding of ECMWF cloud fields for use in optional in-cloud chemistry.
- To speedup the time step initialization, the re-gridding of the TNO anthropogenic emissions is in the new release performed directly after reading; this gives tiny round-off differences compared to the previous implementation.
- The default settings prescribe the use of the latest versions of the input files. The changed input is:
 - small fix of the tree database used for biogenic emissions;
 - new gamma-water file used for ammonia deposition;
 - update of the PM composition tables of the MACC emissions;
 - small fix of the time profiles used for the MACC emissions;
 - updated average soil-water field for larger domain.

For a complete and more detailed overview of the changes in this release, see the log file:

`base/000/CHANGES`

3 Version control system

3.1 Introduction

The source files of the LOTOS-EUROS model are managed with a version control system. With such a system users can obtain an up-to-date copy of the files, archive their changes, obtain changes made by other users. Currently we use the 'subversion' package for this.

Throughout this User Guide some examples will be given on how to use the `subversion` package to manage files. By default we assume that the 'svn' program is available, which is a standard tool in a Linux environment. In addition, a GUI tool for source file management with `subversion` might be available, for example `rapdidsvn`, or `kdesvn` in the K-desktop.

3.2 Subversion server

The base of the 'Subversion' system is a 'repository', which is a central database that keeps record of all changes.

For LOTOS-EUROS the repository is located at a subversion-server hosted at KNMI. Users need to have an account for this server; if you do not have it yet, contact the author. The server has a web interface that can be used to browse through the code. Browse to:

```
https://svn.knmi.nl/usvn/
```

After providing login name and password, first select the 'le' project and then choose 'Browser'.

3.3 Checkout a copy of the source

To obtain a fresh and up-to-date source code, checkout a complete version from the server:

```
svn checkout https://svn.knmi.nl/svn/le/trunk/lotos-euros
```

This includes historical versions, tools, and some documentation. If you only need the latest version, add the name of the version directory ('v1.8') to the path.

3.4 Managing source files with subversion

To learn more about using subversion, just search on the Internet for an introductory course.

Some quick steps:

- To see if you changed some files in or below the current directory:

```
svn status
```
- To see your changes, but also to check if an other user committed a change to the server:

```
svn -u status
```
- To import all changes:

```
svn update
```
- To add a new file to the repository:

```
svn add newcode.F90
```
- To commit a change to the server:

```
svn commit -m 'This is an important change.' newcode.F90
```

3.5 Daily export of latest code

At some institutes the version of the code might be available from a daily export from the Subversion server. Known locations are:

- TNO: `${LE}/source-export`

A template for a script to perform this export is available as:

```
lotos-euros/tools/bin/le_export_from_svn
```

4 Source files

4.1 Identification of a model version

A model version is identified by a three-leveled key, for example "1.8.000". Here, the first 2 numbers identify the base version number ("1.8") while the third one is a patch number ("000"). New base versions are usually released before the beginning of a new year. Throughout the year, patches might be released to fix a bug or to add new functionality to the current base version.

In the rest of this document the examples are based on version 1.8.000 . Note that this just reflects the status at the time of writing of that particular part of the document. If you are a new user it is advised to simply use the latest version, which has probably a higher patch number. As explained in the next section, list the directory 'base' to see what is the latest patch in your copy.

4.2 Directory structure

From v1.8 onwards the LOTOS-EUROS model is shipped in the following directory structure:

```

README           # First aid information.
v1.8/            # Version directory
  base/          # Base sources
    000/         # Patch directories
    001/         #
    :           #
  proj/         # Modifications to the base source for special projects
  doc/          # Documentation specific to this version
  tools/        # Postprocessing and other tools
doc/            # Other (general) documentation
tools/         # Other (general) tools

```

The 'base/000' directory and the subdirectories of 'proj/' contain a number of subdirectories with prescribed names:

```

base/000/src/    # Fortran sources
base/000/bin/    # scripts
base/000/rc/     # configuration files
base/000/data/   # data tables
:
proj/testchem/v1.8.000/src/
proj/testchem/v1.8.000/bin/
:

```

This is to keep Fortran files, scripts, and other data clearly separated from each other. During the setup of a run, a copy of the source files is created in a temporary build directory, and then compiled over there. The source directories are therefore not polluted by object and module files.

4.3 Patch directories

A patch is an official update of the base, for example with a bug fix or with a new feature that is supposed to become part of a new release. The code of a patch version is stored in the 'base' directory:

```

base/000/       # initial base version
base/001/       # first update

```

```
base/002/      # second update
:
```

From v1.8 onwards, the complete code of a base+patch is stored in the patch directory. (This is different previous versions, where the patches were incremental special projects, with only the files that were changed with respect to the base; see the User Guide of v1.7). In the rest of this document we will simply refer to a 'base+patch' code as a 'base' version. If not specified explicitly, the initial patch number is '000'.

4.4 The concept of source projects

A base directory like 'base/000' contains a frozen version of the model. A base version should run without any problem.

In addition to a base, a project could be defined as a modification of files in the base source. A project could also include new files that are not part of the base yet. Typically, projects are be defined to:

- implement a non-standard feature that is very specific to an infrequently used application.
- create special output required for some applications.
- test new module features.

For example, a source code could be combined from the following directories:

```
base/000/src/           # base+patch
proj/testchem/v1.8.000/src/ # test code
proj/myoutput/v1.8.000/src/ # user specific
```

In here, the 'base/000/src' directory contains a complete model source, from which some files are replaced by those from the 'proj/testchem/v1.8.000' directory, and in addition by some from the 'proj/myoutput/v1.8.000' directory. The order is important: the latest version of a file that is copied will be the one that is actually compiled. A list of projects to be used should be specified in the run configuration file (explained in section 5.4). For this example it will look like:

```
my.source.dirs      : base/000 testchem/v1.8.000 proj/myoutput/v1.8.000
```

Although not necessary, it is good practice to include the version+patch number as a sub-directory of a project. In this way it is immediately clear to what base version this project is an extension too, and it is possible to support a project for multiple base versions. 5.8.

5 Running LOTOS-EUROS

This chapter describes how to configure and start a run with the LOTOS-EUROS model. In section 5.1 a summary for the impatient is given; from section 5.2 onwards the steps are explained in more detail.

5.1 Quick start

LOTOS-EUROS can be run by taking the following steps. This is just to give a first impression or a quick reminder; for details, take a look at the sections that follow.

1. Go to the required version directory:

```
cd lotos-euros/v1.8
```

2. Ensure that a link to a 'setup_le' script is present:

```
ln -s base/000/bin/setup_le setup_le
```

which will be visible as:

```
setup_le -> base/000/bin/setup_le
```

3. Copy a template rc-file and give it a suitable name:

```
cp base/000/rc/lotos-euros.rc test-lotos-euros.rc
```

Modify it using a text editor, or at least browse through it to see if the settings for your run are ok.

4. Setup a run-directory and compile an executable using the setup script:

```
./setup_le test-otos-euros.rc
```

5. Follow the instructions written by the setup script. Probably, the instructions tell you to go to the run directory and start the submit script:

```
cd <rundir>
./submit_le lotos-euros-test.x lotos-euros-test.rc
```

Alternatively: supply the '-s' option to the 'setup_le' script to submit automatically after the setup is finished.

The 'submit_le' script accepts options to let the job run in the background or in a queue system; see section 5.12 for details.

5.2 Version directory

The best place to setup and start the model is from what we will call the '*version directory*' directory. Change to that directory before following the next steps:

```
cd lotos-euros/v1.8
```

5.3 Run scripts

The scripts used to setup, start, and submit a LOTOS-EUROS run are placed in (for v1.8.000):

```
base/00/bin/
```

Two groups of scripts can be distinguished:

- the 'pycasso' scripts (PYthon Compile And Setup Scripts Organizer) used to compile and setup a run;
- the 'submit_le' scripts used to submit a run to for example a queue system.

The 'setup_le' script is main script for each run. It is convenient to link the setup script to the version directory because the script is called frequently:

```
ln -s base/000/bin/setup_le setup_le
```

If the scripting changed in higher patch version, it might be necessary to let this link point to the latest version. The scripts that are used are always those that are next the setup script that will be called.

5.4 The rc-file

The configuration of a model run is done through a text file called the 'rc'-file.

5.4.1 Copy from template

Copy a template rc-file and give it a suitable name:

```
cp baese/000/rc/lotos-euros.rc test-lotos-euros.rc
```

Modify it using a text editor, or at least browse through it to see if the settings for your run are ok.

5.4.2 Format of the rc-file

The abbreviation 'rc' comes from 'resource', or specifically from the Unix 'X' resource file on which the format of the file is based. However, also 'run configuration' is a suitable expansion.

A simple example of an rc-file could be:

```
! name of input file:
le.input : /data/a.txt
```

This assigns the value "/data/a.txt" to the key "le.input". Tools are available for the run scripts and the model to read values from an rc-file given the keys. The basic format rules for the rc-file are:

- Keys and values are separated by ':' .
- Empty lines are ignored.
- Comment lines start with '!' and are ignored as well.

More advanced usage is possible, for example expansion of keys or environment variables, and conditional setting using if-statements. For a description of the features use:

```
base/000/bin/rcget --doc
```

The best way to learn about all configuration options is to browse through the template rc-file. The added comments explain the use of the various keys and the values they can take.

5.4.3 *The machine-specific rc-file*

An import setting in the main rc-file is the selection of the 'machine' specific rc-file. This file contains all settings specific for the computer on which the model is running, e.g. compiler settings, library locations, data locations, etc. This file is for example:

```
base/000/rc/machine-tno-azoren.rc
```

For each institute/machine combination, a machine-specific rc-file should be present. If it is not, create a new one. To load the settings in the rc-file, select the appropriate machine rc-file:

```
my.machine.rc : machine-tno-azoren.rc
```

This variable is used elsewhere in the rc file to include the settings from the correct directory:

```
! include settings:
#include ${my.pycasso.dir}/rc/${my.machine.rc}
```

5.4.4 *The compiler rc-file*

The machine-specific rc-file includes a file with compiler-specific settings:

```
! compiler specific settings:
#include ${my.pycasso.dir}/rc/compiler-ifort-v12.1.rc
```

For each compiler suite used to compile the model, a compiler-specific file should be present.

5.4.5 *The expert rc-file*

Many settings that have to do with the setup and installation of the model have been hidden for the users by collecting them in the 'expert' rc-file, which is included into the main rc file:

```
! include expert settings to build source code
#include ${my.pycasso.dir}/rc/lotos-euros-expert.rc
```

Don't touch this file unless you know what you are doing.

5.5 **Setup a run**

Call the script with the rc file as argument to setup a run-directory and compile an executable:

```
./setup_le lotos-euros-test.rc
```

To see the extra options that are accepted by the setup script, use:

```
./setup_le --help
```

Note that all 'long' options like '--help' usually have a 'short' version too, in this case '-h'. A useful option is '--new' or '-n', which first removes the existing build directory followed by creating a completely new one. In case you receive strange errors from the compiler, that might have to do with messing up old and new objects, so try whether this option solves the problems.

Another commonly used option is '--jobs=1' or '-j 1' which ensures that source files are compiled one by one. Without this limitation, the 'make' program will try to compile as much files as possible simultaneously (if they can be compiled independently). Although the latter strategy is obviously much faster, it may mess up the messages printed to the screen.

5.6 The run directory

The first step taken by the 'setup_le' script is to create a run directory on a location specified in the rc-file. Typically the run-directory will be located on a scratch space, since a lot of temporary files are created while running that do not have to be backed-up. A typical content of the the run directory is:

```
<rundir>/build/      # source code, object files
<rundir>/run/       # executable, rc-files, submit script, log files
<rundir>/output/    # output files
<rundir>/restart/   # restart files
```

5.7 Preprocessing of the rc-file

The next step taken by the 'setup_le' script is the preprocessing of the rc-file. The preprocessed file is put in the directory <rundir>/run/. Preprocessing the rc-file includes evaluation of environment variables and other features to make configuration easier. For an overview of all features, use:

```
base/000/bin/rcget --doc
```

5.8 Collection of a source code

The next step is the collection of a source code in a build directory. The build directory is usually located on a temporary scratch disk, the exact location is specified in the rc-file. The sub-directory where the code is collected includes the name of the compiler and the choices for the compiler flags (see section 5.11 for the compiler flag settings). This is done to ensure that an executable is compiled with the same flags applied for all source files. An example of a sub-directory name:

```
<rundir>/build_optim-none_check-all/src/
```

The concept of source projects is explained in section 4.4. In summary, the first collected source files are those in the 'base/ppp' directory. Subsequently, the files from the base version are optionally overwritten by patches or project specific versions from 'proj/' directories.

For example, the rc-file might contain the following setting for the list of source directories to be included:

```
my.source.dirs      : base/000 testchem/v1.8.000 proj/myoutput/v1.8.000
```

In this example, the LOTOS-EUROS source is formed from:

1. the files in 'base/000/src' ...
2. ...or those from 'proj/testchem/v1.8.000/src/' ...
3. ...or those from 'proj/myoutput/v1.8.000/src/'.

The test files in 'proj/testchem/v1.8.000/src/' will not affect the 'official' versions of the code. Thus, if you want to change something in the code and test it first, create a new project directory.

5.9 Generation of source files

Some source code files are generated by the scripts using settings in the rcfile and data tables. These are:

- source files with tracer definitions and chemistry codes (see section 7);
- preprocessing macro include files (section 8).

Default versions of the generated source files are included in the base source. Editing won't have any effect; instead change the rcfile settings or the tables that were used to generate them.

5.10 Creation of the Makefiles

The 'pycasso' scripting automatically creates the Makefiles using the 'makedepf90' program. If this program is not available, instructions are displayed.

5.11 Compilation

The final step performed by the 'setup_le' script is the compilation of the executable. An important configuration choice for compiling is setting the compiler flags. By default, the executable is compiled with the 'fast' flags to have a run-time as low as possible. For testing and debugging it is however useful to enable the 'check' flags, which could for example trap out-of-array-bound problems and floating-point-exceptions (division by zero etc). *So after changing the code, first run with checks enabled!*

The flags to be applied can be set in the pycasso rc file by a list of keywords. The default setting for compilation with the fast flags is:

```
my.build.configure.flags      :  optim-fast
```

For testing and debugging purposes, the 'check' flags can be turned on with:

```
my.build.configure.flags      :  optim-none check-all
```

The actual flags assigned to these keywords are set in the compiler rc-file described in section 5.4.4.

5.12 Submit script

The LOTOS-EUROS executable should be started using the submit script. First switch to the run directory. Its location can be found in the message displayed at the end of the setup by the 'setup_le' script, for example:

```
cd /scratch/yourname/PROJECT/test1/run
```

Next, execute the script with the executable and the rc file as arguments (see again the message displayed by 'setup_le'):

```
./submit_le  lotos-euros-PROJECT-test.x  lotos-euros-PROJECT-test.rc
```

The submit script has some extra options. For example, to let the model run in the background (so that you can log out without killing the run), use:

```
./submit_le  lotos-euros-PROJECT-test.x  lotos-euros-PROJECT-test.rc \
--background
```

Similarly, use '--queue' to submit to a queue system. To see all options, use:

```
./submit_le --help
```

5.13 Setup and submit in a single step

To setup and submit in a single step (so without the need to go to the run directory and call 'submit_le'), just add the '-s' or '--submit' option to the setup script:

```
./setup_le lotos-euros-test.rc --submit
```

The setup script also accepts the '--background' and '--queue' options that are then passed on to 'submit_le' (i.e., to submit to the background or to a queue system, respectively).

6 Input data

6.1 Minimum package

A package with input data is available to run the model with the default settings.

6.1.1 Content

The content is as small as possible, but large enough for a model run with the following properties:

- simulation period July 2006;
- European domain as used for operational forecast;
- MACC-2003-2007 emissions;
- boundary conditions 'macc-r-aer' for dust and 'macc-g-rg' for reactive gasses.
- extract of the local 'AQORD' data base with observations that can be compared with the output using the DIADEM post processor (section 13.1).

The directory tree of the data unfolds too is:

```
models/
  /LOTOS-EUROS/
    /data/
      /cf-standard/
      /meteo/
        /ecmwf/od/europe_w30e60n25n70_...
      /landuse/
        /forest/
        /soiltext/
        /traffic/
        /soilwater_avg/
      /ammonium/
      /emissions/
        /MACC-2003-2007/
      /bound/
      /standard/
    /GEMS/
      /rd/
      /mozart_out/
    /MACC/
      /fire/
  observations/
    /AQORD/
```

6.1.2 Settings

The location of the input data is machine specific and therefore set in the 'machine.rc' file. With the data unpacked to a directory:

```
/data/models
```

use the following two settings to have the correct locations in the main rcfile:

```
MODELS          : /data/models
my.data.dir     : ${MODELS}/LOTOS-EUROS/data
```

6.1.3 *How to obtain ?*

Contact TNO to obtain the package (about 4 Gb).

7 Generation of chemistry code

7.1 Overview

With the number of tracers growing throughout the years it became more and more difficult to manage the source files solving the chemistry. In addition, there was no easy way to quickly enable or disable some tracers or chemical reactions without having to change the source code.

It was therefore decided to let some of the tracer and chemistry related file be generated by the scripts, based on settings in the rcfile. The scripts used to generate the source files start with the name 'genes' (GENERate Sources), and this name is also used in the settings keywords.

The generated files are:

- 'le_indices.inc', an include file with index parameters, names, units, and other settings for all tracers;
- 'le_chem_work.F90', a module with routines that fill the reaction rates and perform a single step of the iteration step in used by the chemistry solver.

Default versions of the generated source files are included in the base source. Editing won't have any effect; instead change the rcfile settings or the tables that were used to generate them, as explained below.

The new source files are generated in the build directory. For debugging it might be useful to take a look at the generated files after creation; this is facilitated by the rcfile setting:

```
genes.show.command : nedit ${genes.files} &
```

In this case, the new files are loaded into the 'nedit' editor; the ampersand ensures that the editor keeps running as background process while the rest of the model setup continues.

7.2 Tracer and chemistry properties

An important part of the configuration is done through so-called 'properties'. Table 7.1 shows the currently supported properties; their use will be explained later on. A list with supported properties is maintained in the expert rcfile, which is used to check the settings made by the user.

To select the tracers and reactions that should be included in a simulation, a list with properties should be specified in the rcfile. Default selection is:

```
genes.prop.selected : cbm4 ppm ec pom sia seasalt dust accum biascorr
```

This is used to select the appropriate lines from the tables described below.

7.3 Tracer table

The core of the tracer selection is a table with all supported tracers. The default table is available as:

```
base/000/data/tracers.csv
```

The content looks like:

```
name , description , units, formula, properties
NO , Nitric oxide , ppb , N + O , cbm4
O3 , Ozone , ppb , O 3 , cbm4
ALD , Aldehyde , ppb , C 2 + R, cbm4 nmvoc
PPM_f, prim.part.mat. fine mode, ug/m3, R , ppm aerosol fine_mode
:
```

cbm4	tracers/reactions of CBM4 scheme
cb99	depricated; used to support some ancient parts of the code;
sulphur	sulphur-only scheme (SO2 and SO4a, OH read in)
methane	methane-only scheme (CH4, OH read in)
nmvoc	non-methane volatile organic carbon
radical	radical
ppm	primary particulate matter
ec	elementary carbon
pom	primary organic matter
sia	secondary inorganic aerosols
seasalt	sodium aerosols representing seasalt
dust	dust aerosols
cg soa soa_prec	for secondary organic aerosol modelling
basecation	base-cat-ion aerosols
hm	heavy metals
accum	accumulated species
biacorr	bias corrected species
aerosol	aerosol tracer
fine_mode	fine mode aerosol
coarse_mode	coarse mode aerosol
all_modes	total aerosol collecting all size modes
in-cloud	used to select implicit in-cloud chemistry reactions

Table 7.1 Tracer and chemistry properties.

For each supported tracer, the following values should be set:

- *name* Short name, used in index variables `i_03` etc.
- *description* Longer description, only used in comments in generated source files.
- *units* Units of the tracer in the model; usually 'ppb' for gasses, and 'ug/m3' for aerosols.
- *formula* Chemical formula (if possible). For some applications (labeling) it is useful to have at least the number of C, N, and S atoms in a molecule; for the remainder, use the symbol 'R' .
- *properties* List with all tracer properties (table 7.1) that are apply to this tracer.

To be enabled in the simulation, a tracer should have at least one of the selected properties.

7.4 Reaction table

Chemical reactions are also specified in a text file. The default table is available as:

```
base/000/data/reactions.csv
```

The content looks like:

```
label, reactants      , products      , rate expression      , properties
R1   , NO2                , NO + O3       , 1.0 x <NO2_SAPRC99> , cbm4
R3   , O3+NO             , NO2           , 2.64 @ 1450         , cbm4
:
RH1f , SO4a_f + N2O5, SO4a_f + 2*HNO3, rk_het(ireac_N2O5_NH4HSO4a_f), cbm4 sia
:
```

For each reaction, the following values are specified:

- *label* A short label for the reaction, used in parameter names.

- *reactions* Tracers reacting with each other; tracer names following the tracer table.
- *products* Tracer products.
- *rate expression* Cryptic description of the reaction rate. See the comment in the top of the reaction table for how the expression is expanded. Some reaction rates are set by a special routine in the code, for example for the heterogeneous reaction 'RH1f' in the example lines above.
- *properties* List with all tracer properties (table 7.1) that that should be present to have this reaction enabled.

Reactions are only enabled if ALL of it's properties are selected in the rcfile. Thus, reaction 'RH1f' of the example above is only enabled if both 'cbm4' and 'sia' tracers are selected.

7.5 Specification of table files

The name of the tables file should be specified in the rcfile. The default setting in the rcfile is:

```
genes.tracers.file      : ../data/tracers.csv
genes.reactions.file   : ../data/reactions.csv
```

In this case, the tables are found in the 'data' sub-directory next to the 'src' sub-directory in the build directory, thus:

```
<rundir>/build/data/
```

These table files are copied from the base directory, and eventually replaced by project specific versions. Therefore, to test a new table, simply put it in the 'data' sub-directory of a project directory. Alternatively, specify an absolute path in the rcfile settings.

7.6 Tracer indices and arrays

The information on the selected tracers is used to create the file 'le_indices.inc'. This file is included into 'le_indices.F90', which provides the module 'Indices'. Through this module, the user can access the generated entities; see the comment in top of 'le_incdices.F90' for their definition.

8 Pre-processor macro's

Preprocessing macro's are a convenient tool in programming large applications. This chapter describes the use of these macro's in the LOTOS-EUROS source.

8.1 Example of usage

Sometimes a minor modification of the source code is needed to hide or enable certain features are too small to be implemented with a code project. Instead, so-called pre-processing macro's are used. For example:

```
#ifdef with_netcdf
status = NF90_Create( 'test.nc', NF90_CLOBBER, ncid )
#else
stop 'not compiled with NetCDF library enabled'
#endif
```

In this example, if the macro 'with_netcdf' is defined, then only the code between '#ifdef' and '#else' is used, otherwise the code between '#else' and '#endif' is used. If the code is compiled with the NetCDF library enabled, then the macro 'with_netcdf' should be defined and the file 'test.nc' will be created as specified, if this piece of code is called. However, a user wants to run the model, but does not need to write netcdf files, then this macro definition could be omitted. This is in particular useful if this library is not available; the user should not be bothered with a compiler complaining that a library is not available while it is not used anyway.

8.2 Macro definition include files

Macro's are defined by statements like:

```
#define with_netcdf
```

All macro definitions are collected in small include files. For example, for macro definitions in LOTOS-EUROS source files the include file 'le.inc' could look like:

```
!
! Include file with macro definitions.
!
#define with_netcdf
```

This file is included in the header of a file with:

```
#include "le.inc"
```

8.3 User settings

Which macro's are defined is something that is often user and application depended. Therefore, a list of macro's to be defined is part of the rfile:

```
my.le.define : with_grib_api with_netcdf
```

From the values in this list the macro include file(s) are written automatically by the source configuration scripts.

8.4 Expert settings

Working with macro's is considered an expert job, since small typo error could cause a part of the code to be omitted without noticing. Therefore, the expert rcfile includes lists of macro's that are supported. The source code is checked on use of macro's that are not supported; if these are found, an error is raised and configuration stops. The lists in the expert rcfile are used to add macro definitions to the proper macro include file.

9 Horizontal grid

9.1 Grid definition

The horizontal grid is defined in the rcfile by specifying the lower-left corner, the resolution, and the grid size. The default grid is currently the same as used in the operational forecast:

```
grid.west      : -15.0
grid.south     :  35.0
grid.dlon      :   0.50
grid.dlat      :   0.25
grid.nx        :  100
grid.ny        :  140
grid.nz        :    4
```

There are no restrictions on the position of the lower-left corner or the resolution. Input is converted automatically to the required grid; if the model domain extends the coverage of the input, an error is raised.

9.2 Zooming

Running in zoom modes simply means running the model twice: first for a large domain and coarse resolution, then for a smaller domain in fine resolution. The later should be configured to read concentrations from the first run as boundary conditions. The following steps are usually taken to setup a zoom run.

1. Perform a run on large domain:
 - large domain covering the future zoom grid;
 - coarse resolution;
 - output of 3D concentration fields:
 - all relevant tracers;
 - bounding box around zoom region to save disk space (section 11.1.1);
 - remember the run-id and the output directory.
2. Perform the zoom run:
 - small domain within the previous domain;
 - fine resolution;
 - boundary condition type 'le', configure this to read the (bounded) 3D concentration fields written in the previous run;
 - eventually put out out concentrations in the halo cells to check the inheritance of the boundary conditions, see section 11.1.1.

Note that the time steps for the zoom run are automatically reduced to match the CFL-criterion (section 10).

10 Time steps

The simulation time steps are set in the following way; see 10-1 for an illustration.

1. The user should specify the 'output' time-step in the rcfile:

```
! output time step in minutes:
timestep.output      :   60
```

A typical value is 1 hour. The model will arrive at every multiple of this output-time-step and put out simulated values.

2. Within an output step, the maximum allowed time step for the individual processes is determined. Currently the advection is the limiting process. The time step limit for advection is based on the CFL-criterion: within a time step, a parcel of air should not cross a complete grid cell, to avoid that some processes are not applied to it. For smaller grid size this leads to a smaller maximum time step; typically, if the resolution in at least one direction doubles, then the number of required time steps is the double too.
3. Within the operator splitting sequence, processes are performed after each other; first all processes in some order for a half the time step, and then in the reverse order for the other half. If a process is to be performed twice directly after each other, the two half-steps are combined into a full-step. In the current operator-sequence the chemistry is the first process and the advection the last; if three operator splitting steps are required within an output-step, then the advection is performed three times (full-steps), the chemistry four times (a half step, two full steps, and a half step), and all the other processes six times (six half steps).

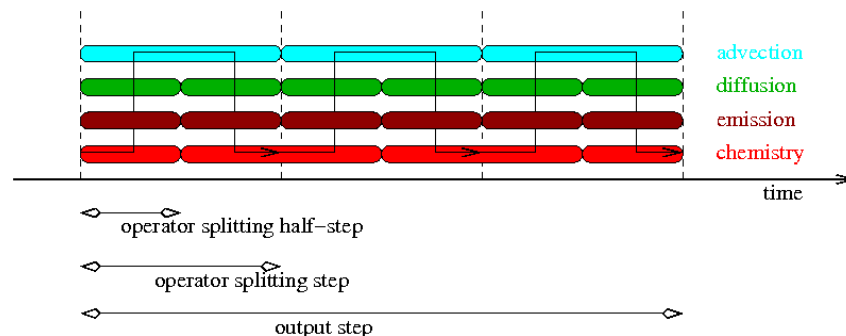


Figure 10-1 Illustration of time step settings.

11 Model output

11.1 Gridded output

The output of gridded fields is controlled by the `rcfile`. The supported output types are described below. It is possible to put out files of the same type but with different properties, for example files with concentration fields at the surface for many tracers, and files with 3D fields for some selected tracers only.

The gridded output is written to NetCDF files. By default the files are structured following the CF-conventions. GrADS description files are added for visualization (see section 13.2).

11.1.1 Concentration fields

For output of concentration fields the following properties could be set:

- which tracers to be put out:
 - model tracers;
 - accumulated tracers: total PM10, total carbon, etc; see `indices.F90` for the supported accumulation;
 - bias corrected tracers;
- vertical ax:
 - model levels, including the 2.5m surface 'layer' and the top boundary layer;
 - heights relative to orography;
 - elevations relative to sea-level;
- whether to put out the cell height too;
- temporal resolution (typically fields are put out every hour);
- collection per file: either daily or instantaneous;
- horizontal coverage: by default the whole grid, but optionally:
 - bounding box to limit the horizontal area, for example to save storage space while producing boundary conditions for a zoom run;
 - halo cells (boundary conditions values);

11.1.2 Tracer total columns

Total columns could be put out for comparison with satellite observations. Note that this only includes the model layers now; the top boundary is not included yet, since usually there is idea about its height. The following properties could be set:

- which tracers to be put out (normal or accumulated);
- temporal resolution (typically fields are put out every hour);
- collection per file: either daily or instantaneous.

11.1.3 AOD columns

The Atmospheric-Optical-Depth is computed from the tracer concentrations and put out as columns.

11.1.4 *Model data fields*

For output of various model data fields, the following properties could be set:

- which fields to be put out: meteo variables, stability fields, . . .
- vertical ax:
 - model levels, including the 2.5m surface 'layer' and the top boundary layer which are for most fields a copy of the nearby model layer;
 - heights relative to orography;
 - elevations relative to sea-level;
- temporal resolution (typically fields are put out every hour);
- collection per file: either daily or instantaneous.

11.1.5 *Emission fields*

The total emission for a certain tracer (or accumulated tracer) could be put at at regular times. The array that is put is 'emis_a' which contains for each tracer the total emission during the current (next) time step, independent of the source (anthropogenic, biogenic, sea-spray, etc). The following properties could be set:

- for which tracers the emissions should be put out, including accumulated tracers;
- vertical ax: either model layers or the total per grid cell;
- temporal resolution (typically fields are put out every hour);
- collection per file: either daily or instantaneous.

11.1.6 *Dry- and wet-deposition*

The dry- and/or wet-deposition budgets per output time step could be put out with the the following properties:

- for which tracers the deposition should be put out, including accumulated tracers;
- vertical ax: either model layers or the total per grid cell;
- temporal resolution (typically fields are put out every hour);
- collection per file: either daily or instantaneous.

11.1.7 *Daily budgets*

The daily budgets are updated every time step and are put out at midnight. The following budgets could be put out:

- dry deposition flux of SO_x, NO_x, or SO_x;
- wet deposition flux of SO_x, NO_x, or NH_x;
- ozone dry deposition flux per landuse;
- ozone daily maximum;
- average NH₃ concentration in soil.

11.2 Satellite validation output

For special purposes also the following fields could be produced:

- Simulation of OMI NO2 columns.
- Simulation of MODIS AOD columns.

The horizontal model grid is used to sample the satellite pixels. This output therefore requires that information on the pixels is available.

11.3 Observation simulation

The standard method to produce simulations of concentrations at observation sites is to run the DIADEM post processor (section 13.1).

An alternative is to use the MAORI (Model And Output Routine Interface) routines. The MAORI interface is configured via the rcfile by specification of a table file with station location and settings to define the simulated tracers etc. This is mainly used in the Kalman Filter context to simulate state values at observation sites; in the default model the code is present, but the use is not fully supported yet.

11.4 Emission summary

A summary of the emissions is written automatically to the output directory if possible. Currently this is only supported for emissions read from TNO files. Since these are year-dependent, the summary is written for every year that the simulation covers. A summary consists of four comma-separated-value file (plain text): a list of the emission categories, a list with country codes and names, a table with total emissions for a component per country, and similar per country and emission category.

12 Run time

12.1 Timing

The time spent on different tasks in the model is continuously measured.

A timing profile is written to a text file with extension '.prf' in the output directory. In the header of this file, a pretty print of the absolute and relative time spent on a task and its sub-tasks is shown:

```

-----
timer                                     system_clock      (%)
-----
...
model time loop                          11504.66
  time step output                         712.27 ( 6.2 %)
  time step setup                          1223.58 ( 10.6 %)
  adjust                                    49.89 ( 0.4 %)
  advection                                 573.52 ( 5.0 %)
  chemistry                                 6819.19 ( 59.3 %)
  vertical diffusion                        1144.94 ( 10.0 %)
  dry deposition                            217.74 ( 1.9 %)
  sedimentation                             24.93 ( 0.2 %)
  wet deposition                             23.61 ( 0.2 %)
  emission                                   35.59 ( 0.3 %)
  other                                     679.41 ( 5.9 %)
....

```

The DIADEM post processor (section 13.1) could be used to create a browsable table of this profile bar graphs to visualize. Figure 12-1 shows an example of this for the processes in the time loop, comparing the latest patch of v1.7.9 with the new release. The new version seems spends significantly less time on the time step initialization, which was achieved by moving the re-gridding of the anthropogenic emissions from the time step to the initialization phase. The chemistry is the most expensive part taking about 60% of the run time; it has become slightly more expensive since in this simulation it was performed more often as a result of the automatic time step setting.

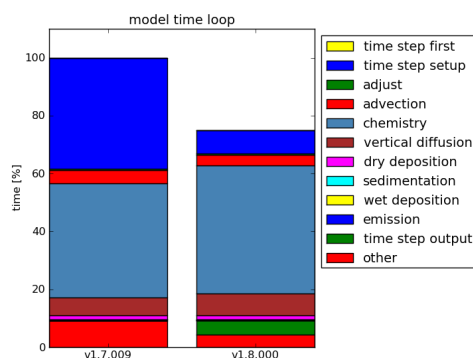


Figure 12-1 Run time spent on processes for v1.7.009 and v1.8.000 . Note that for v1.7.009 the 'time step output' task was included in the 'other' task. Note that for v1.7.009 the 'time step output' task was included in the 'other' task. Note that for v1.7.009 the 'time step output' task was included in the 'other' task. Note that for v1.7.009 the 'time step output' task was included in the 'other' task.

12.2 Parallelization

12.2.1 OpenMP

The LOTOS-EUROS model uses OpenMP to run in parallel. To run on multiple threads, enable the `'par.openmp'` flag in the rcfile and specify the number of available threads. Note that in version v1.8 the default secondary-inorganic-aerosol equilibrium scheme is 'ISORROPIA2'. For this scheme, no OpenMP support is available in v1.8.000; for the timing plots shown below, the 'EQUISAM' scheme was used instead. Support of OpenMP for 'ISORROPIA2' will be available in v1.8.001 however.

12.2.2 Visualization

The DIADEM post-processor (section 13.1) could be used to visualize run times and speed up for different numbers of OpenMP threads. Figure 12-2 shows the result for the current version. An almost perfect speedup is achieved for the advection, vertical diffusion, and chemistry. The overall speedup is hampered however by the I/O (reading meteo, writing results) and the time step setup (emission model). With 2 threads a reasonable speedup of 1.5 is reached, while with 4 threads only a speedup of 2 is reached. For Kalman Filter applications which perform the time loop many times but the setup only once, a much better overall speedup will be reached.

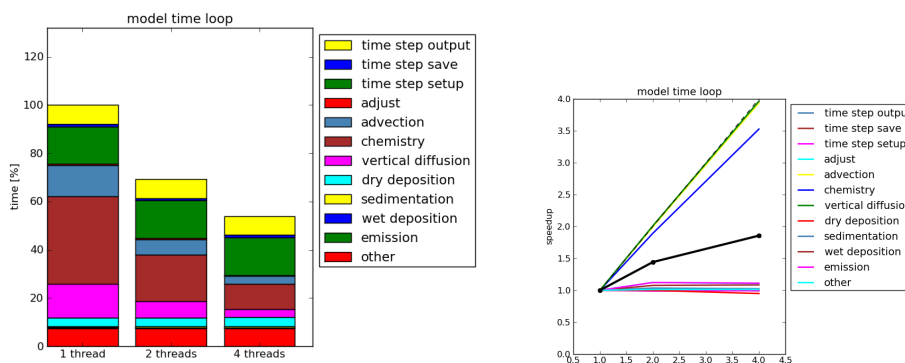


Figure 12-2 Time spent on different processes and their speedup for runs using OpenMP parallelization.

12.2.3 Adding new OpenMP directives

To add OpenMP directives to new code, the following approach is suggested.

1. Create a new rcfile that configures a short model run:
 - simulation over, for example, 6 hours only, since otherwise testing will take too long;
 - compiler flags `'optim-strict'`, since otherwise simulation results might differ slightly between two runs with and without OpenMP;
 - save restart files at least at the end of a run.
2. Setup a test environment using the `'ttb'` scripts (section 14.1). Configure a `'change'` test for run id's named `'ser'`, `'omp1'`, `'omp2'`, `'omp4'` etc. Change the rcfile such that the `'ser'` run defines a serial run, `'omp1'` defines a run with OpenMP enabled and 1 thread only, `'omp2'` defines a run on 2 threads etc. Use for example the following switch in the rcfile:

```
#if "ser" in "${run.id}"
par.openmp : no
par.nthread : 1
#elif "omp1" in "${run.id}"
par.openmp : yes
par.nthread : 1
#elif "omp2" in "${run.id}"
par.openmp : yes
par.nthread : 2
#else
#error could not set openmp for runid "${run.id}"
#endif
```

Configure 'ttb' to compare the restart files produced for each run.

3. Call the 'ttb' script with the defined test settings. The standard model should run properly for all defined tests, and the restart files produce should be exactly the same.
4. Use the 'diadem' post-processor (section 13.1) to show the run time profiles, produce bar graphs of the the total run time, and plots of the speedup. Search for a processes on which a relative large amount of time is spent, and is a candidate to add OpenMP. Eventually add extra timer statements to figure out in more detail which part of the model is expensive.
5. Add new OpenMP directives or change existing ones. Run 'ttb' and check if the results are still the same; if not, comment some of the new OpenMP directives and try again.
6. Repeat some the steps until the model cannot speedup anymore, or until there is are no routines left in which OpenMP directives could be inserted.

13 Post processing and visualization

Post-processing and visualization of the model output is often very project and user specific. In this chapter we give an overview of the available standard tools, which will help a user with the first steps.

13.1 DIADEM

13.1.1 Introduction

The 'DIADEM' software ('*DI*Agnostic *DA*ta *EN*d *MI*x') is a set of tools that could be used to produce some plots for the first insight in the model results. What it can do is:

- extract statistics and other numbers from the output:
 - concentration fields averaged over the simulation time
 - time series at observation stations
 - statistics of time series
 - ...
- create plots from the extracted data
- create an html index page to the plots to be able to browse through the data

Figure 13-1 shows an example of how the main index could look like. A gallery of produced figures is shown in figure 13-2.

13.1.2 Location of the scripts

The DIADEM software is shipped with the model distribution in the 'tools' directory, i.e. :

```
lotos-euros/v1.8/tools/diadem
```

This directory contains scripts, settings, and some documentation.

13.1.3 How to run

To start DIADEM, call the main script and pass an rc-file with settings to it:

```
tools/diadem/bin/diadem tools/diadem/rc/diadem.rc
```

You need to change the settings to point the scripts to the location of the model output, the time range to use, which plots to make, etc.

To keep the run settings and the DIADEM settings together, you may insert the content of the 'diadem.rc' file into the rcfile with the model settings and pass this one:

```
tools/diadem/bin/diadem lotos-euros-test.rc
```

One could also include the special 'pycasso' DIADEM settings in the model settings with an include command:

```
! include settings:  
#include tools/diadem/rc/diadem.rc
```

This will automatically set the location of the output and the time range.

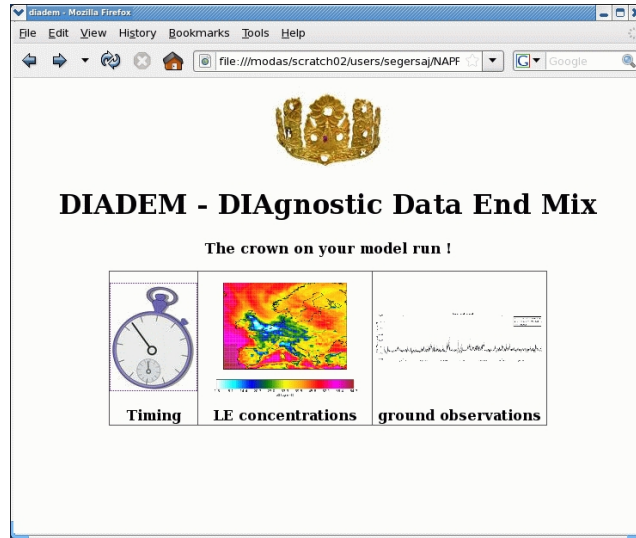


Figure 13-1 Example of DIADEM main index.

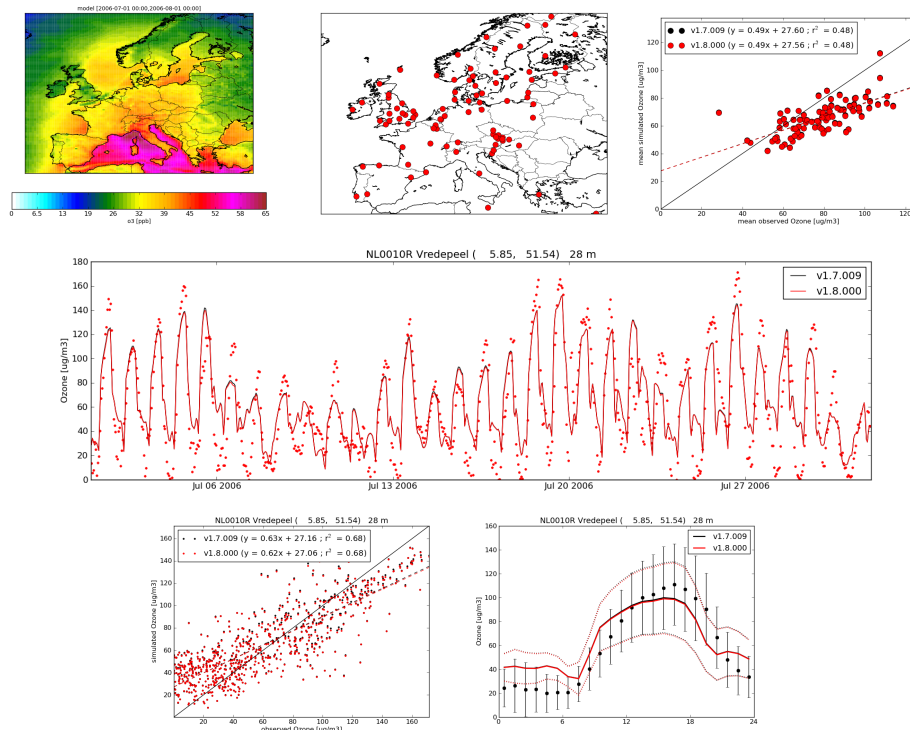


Figure 13-2 Gallery of DIADEM produced figures comparing v1.7.009 with v1.8.000 .

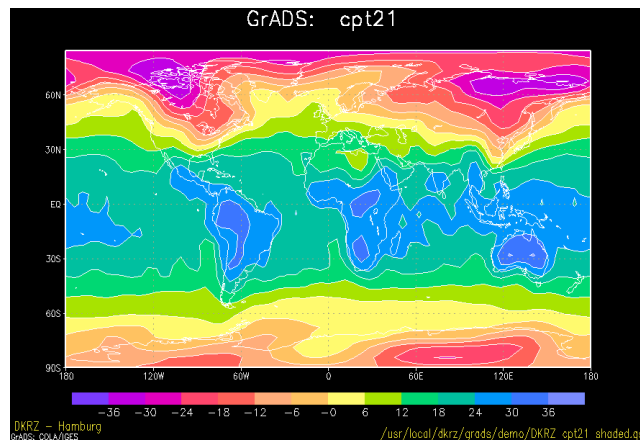


Figure 13-3 Screenshot of a GrADS graphics window.

13.2 GrADS

The *Grid Analysis and Display System* (GrADS) is an interactive desktop tool that is used for easy access, manipulation, and visualization of earth science data.

13.2.1 Control files

LOTOS-EUROS supports the use of GrADS by adding control files (`.ctl`) to the output directories that are used by GrADS to read the output and define the correct grid, time range, etc.

13.2.2 Running

If GrADS is installed, use the following command to start working with the LOTOS-EUROS netcdf output:

```
gradsnc
```


14 Development tools

This chapter shortly describes tools available for testing and debugging.

14.1 TTB - The Test Bench

TTB is a small set of scripts that can be used to quickly setup and run some simulations and check if the output is the same.

14.1.1 *Implemented tests*

Two types of test are implemented now:

change With this test a developer could check whether two versions or configurations of the model provided exactly the same result or not. Useful for testing if:

- a change to the code that is not supposed not to alter the results really gives the same numbers;
- results are the same if OpenMP is enabled;
- ...

restart Test if the restart function is implemented well: first perform a long run, and then two runs over the same period with a restart in between. The end result should be exactly the same.

14.1.2 *Usage*

The scripts are available in the tools directory of the version:

```
lotos-euros/v1.8/tools/ttb
```

A template for the settings is provided in:

```
lotos-euros/v1.8/tools/ttb/rc/ttb.rc
```

Edit the settings to select and configure the required test, the content should be self-explainable.

Then run The Test Bench with:

```
lotos-euros/v1.8 $ tools/ttb/bin/ttb tools/ttb/rc/ttb.rc
```

Short messages about the differences between the output files are displayed.

14.1.3 *Optimization flags*

Even changes that should not alter the simulations might sometimes lead to tiny different results because of changes in the optimizations. To ensure that the optimizations do not affect the results, run the model either without optimizations or with the strict optimizations only. For example, when using the pycasso scripting, take the following setting in the rcfile:

```
my.build.configure.flags      : optim-fast optim-strict
```


15 Coding conventions

How should new parts of code be written ? Some ideas.

- Files contain either 1 module or 1 main program.
- Model specific files have a prefix equal to the model name:

```
le_data.F90
le_process.F90
:
```

- Modules have the same name as the source file:

```
module LE_Process
...
end module LE_Process
```

- Module names (thus file names) should reflect the content:

```
! trash bin formerly known as the 'dims' module
module LE_Data

...
end module LE_Data
```

- The tasks to be performed by a module could be distributed over a number of sub modules:

```
LE_Data # top module
LE_Data_grib LE_Data_nc ... # specific
LE_Data_Base # shared entities
```

In this case, other model routines should access the entities only via the top module:

```
use LE_Data, only : LE_Data_Setup
use LE_Data, only : temper, humid, tsurf
```

- Public routines in a module should start with the module name:

```
module LE_Data

private
public :: LE_Data_Setup
...

contains

subroutine LE_Data_Setup( t1, t2, status )
...
end subroutine LE_Data_Setup

...

end module LE_Data
```

- If a module defines public data, a module initialization and finalization routine should be provided. These routines might be dummy to be prepared for future extensions.

```

module LE_Data

  private
  public  :: the_answer
  public  :: LE_Data_Init, LE_Data_Done
  ...

  integer  :: the_answer

contains

  subroutine LE_Data_Init( status )
    ...
    ! something to be done:
    the_answer = 42
    ...
  end subroutine LE_Data_Init

  subroutine LE_Data_Done( status )
    ! nothing to be done.
  end subroutine LE_Data_Done

  ...

end module LE_Data

```

- If a module defines a public type rather than public data, its name should start with the prefix 'T_' followed by the module name. An initialization and finalization routine should be created, eventually doing nothing:

```

module RcFile
  ...
  private
  public  :: T_RcFile, RcFile_Init, RcFile_Done
  ...
  type T_RcFile
    integer  :: id
    ...
  end type T_RcFile

contains

  subroutine RcFile_Init( rcf, fname, status )
    type(T_RcFile), intent(out)      :: rcf
    character(len=*), intent(in)     :: fname
    integer, intent(out)              :: status
    ...
    ! something to be done:
    rcf%id = 123
    open( unit=rcf%id, file=trim(fname), form='formatted', iostat=status )
    ...
  end subroutine RcFile_Init

  subroutine RcFile_Done( rcf, status )
    type(T_RcFile), intent(inout)    :: rcf
    integer, intent(out)              :: status
    ...
    ! something to be done:
    close( rcf%id, iostat=status )
    ...
  end subroutine RcFile_Done

```

```
...  
end type RcFile
```

- Subroutines return an integer status value:

```
! return status:  
! <0 : warning  
!  0  : ok  
! >0 : error  
  
subroutine Process_Init( status )  
  integer, intent(out)  :: status  
  ...  
  ! ok  
  status = 0  
end subroutine Process_Init
```

- Functions are *'pure'* and preferably *'elemental'*.

16 Signature

Name and address of the principal

NMDC - Nationaal Modellen- en Data Centrum
program manager: Jan Matthijssen
p/a Princetonlaan 6, 3584 CB Utrecht, The Netherlands

Names and functions of the co-operators:

A.J. Segers - researcher

Names and establishments to which part of the research was put out to contract:

-

Date upon which, or period in which the research took place:

September 2011 - December 2011

Name and signature reviewer:



Ir. R. Kranenburg

Place and date:

Utrecht, January 24, 2012

Signature:



Dr. Ir. A.J. Segers
Author

Release:



Ir. R.A.W. Albers, M.P.A
Research Manager

