



## LOTOS-EUROS Report

2011-001

LOTOS-EUROS v1.7 User Guide

**TNO** innovation  
for life



Rijksinstituut voor Volksgezondheid  
en Milieu  
Ministerie van Volksgezondheid,  
Welzijn en Sport



Koninklijk Nederlands  
Meteorologisch Instituut  
Ministerie van Verkeer en Waterstaat



Planbureau voor de Leefomgeving



**TNO**

**LOTOS-EUROS Report**

**2011-001**

**LOTOS-EUROS v1.7 User Guide**

Date	May 2011
Author(s)	Arjo Segers
Projectnumber	
Keywords	LOTOS-EUROS, User Guide
Target	LOTOS-EUROS users



## Contents

<b>1</b>	<b>Introduction</b>	<b>7</b>
1.1	What could be found in this guide ?	7
1.2	Raw documentation	7
1.3	Automatic compilation of this document	7
<b>2</b>	<b>Version control system</b>	<b>9</b>
2.1	Introduction	9
2.2	Subversion server	9
2.3	Checkout a copy of the source	9
2.4	Managing source files with subversion	9
2.5	Daily export of latest code	10
<b>3</b>	<b>Source tree</b>	<b>11</b>
3.1	Identification of a model version	11
3.2	Directory structure	11
3.3	The concept of source projects	11
3.4	Patches	12
3.5	Macro definitions	12
<b>4</b>	<b>Running LOTOS-EUROS - up to v1.7.1</b>	<b>13</b>
4.1	Quick start	13
4.2	The resource file	13
4.3	The 'run-le' script	13
4.4	Preprocessing the rc-file	14
4.5	Collection of a source code	14
4.6	Compilation	14
4.7	Setup of rundirectory	14
4.8	Submit script	14
<b>5</b>	<b>Running LOTOS-EUROS - v1.7.2 and higher</b>	<b>17</b>
5.1	Quick start	17
5.2	Version directory	17
5.3	The rc-file	17
5.4	The 'setup_le' script	19
5.5	Preprocessing of the rc-file	19
5.6	The run directory	19
5.7	Collection of a source code	20
5.8	Creation of the Makefiles	20
5.9	Compilation	20
5.10	Setup of rundirectory	20
5.11	Submit script	20
5.12	Setup and submit in one step	21
<b>6</b>	<b>Tracers</b>	<b>23</b>
6.1	Chemistry and/or tracer schemes	23

<b>7</b>	<b>Meteo input</b>	<b>25</b>
7.1	Supported meteo . . . . .	25
7.2	ECMWF meteo . . . . .	25
7.3	RACMO meteo . . . . .	26
<b>8</b>	<b>Boundary conditions</b>	<b>27</b>
8.1	Methods . . . . .	27
8.2	LOTOS-EUROS boundary conditions . . . . .	27
8.3	TM5 boundary conditions . . . . .	28
8.4	MACC Mozart boundary conditions . . . . .	30
<b>9</b>	<b>Emissions</b>	<b>31</b>
<b>10</b>	<b>Output</b>	<b>33</b>
<b>11</b>	<b>Post processing and visualization</b>	<b>35</b>
11.1	DIADEM . . . . .	35
<b>12</b>	<b>Restart files</b>	<b>37</b>
<b>13</b>	<b>Coding conventions</b>	<b>39</b>
<b>A</b>	<b>Tracers</b>	<b>43</b>

# 1 Introduction

## 1.1 What could 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;
- configure the model;
- compile and run the model.

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

## 1.2 Raw documentation

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

```
lotos-euros/v1.7/doc/user-guide/tex
```

A pdf version could be created with:

```
make pdf
```

Similar, browsable version could 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 this document is compiled automatically every night from the latest source files. Known locations:

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

See section 2.5 for details on how this is done.



## 2 Version control system

### 2.1 Introduction

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

Throughout this User Guide some examples will be given on how to use 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.

### 2.2 Subversion server

The heart 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 Arjo Segers at TNO.

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, select the 'le' project.

### 2.3 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.

### 2.4 Managing source files with subversion

To learn more about use of subversion, just search on the internet for an introduction 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:  

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

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

## 2.5 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
```

## 3 Source tree

### 3.1 Identification of a model version

A model version is identified by three number levels, for example "1.7.2". In here, the first 2 identify the base number ("1.7") while the third one is a patch number ("2"). New base versions are usually released at the begin of a new year. Throughout the year, patches might be released to fix a bug or to add new functionality to a base version.

Note that the patch number is often denoted with a 3-digit number ("1.7.002") since that is used in directory names too.

In this rest of this document the examples are based on patch 1.7.2 . This just reflects the version time or writing this part of the documents. If you are a new user it is advised to simply use the latest version, which has probably a patch number higher than 2 . List the directory 'proj/patch' to see what the latest patch is in your copy.

### 3.2 Directory structure

The LOTOS-EUROS model is shipped in the following directory structure:

```

README           # First aid information.
doc/             # Documentation
v1.7/            # Version directory
  run-le         # Run script (depricated from v1.7.2 onwards)
  base/          # Base source, including a default grid and chemistry
  proj/          # Modifications to the base source for special projects.
  rc/            # Sample rc-files with settings for a particular run.
```

The 'base/' directory and the subdirectories of 'proj/' contain subdirectories for Fortran sources and scripts:

```

base/src
base/bin
proj/patch/002.io/src
proj/patch/002.io/bin
:
```

This is to keep Fortran files and scripts seperated.

When setting up a run, a copy of the source files is created in a temporary build directory and compiled. The source directories are therefore not polluted by object and module files.

### 3.3 The concept of source projects

The base/ directory contains a frozen version of the model. The base source should run without any problem.

In addition to the base, a project is a modification of files in the base source, or new files not part of the base code. Projects could be defined to:

- implement a non-standard feature, which is very specific to a not-often used application;

- specila output required for some application;
- sensitivity tests.

A list of projects to be used should be specified in the settings, for example:

```
chem/cbiv99 eumetsat2
```

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

1. the files in 'base/src' ...
2. ... with the chemistry files in 'proj/chem/cbiv99/src/';
3. ... and some files replaced by special versions for the EUMETSAT2 project in 'proj/eumetsat2/src/'.

The test files in 'proj/eumetsat2/src/' will not disturb 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.

## 3.4 Patches

A special set of project directories are the patches. For version 1.7 the first patches are for example:

```
proj/patch/000.base/    # empty
proj/patch/001.fpe/    # fixes floating-point-execptions
proj/patch/002.io/     # extra i/o features
:
```

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 patches are incremental, thus a new patch contains all files of the previous patch plus some extra modifications. In this way the source of a certain version, e.g. v1.7.2, is defined by the 'base' directory and patch '002'.

An informative name is usually added to the name of the patch directories in order to remind users quickly on what was the purpose of a certain patch.

A special project is the '000' patch, which is completely empty, but allows configuration of the base model in terms of 'v1.7.0'.

## 3.5 Macro definitions

Sometimes a minor modification of the source code is needed which is too small to be implemented with a code project. Instead, so-called pre-processing macro's are used. In a code this could look like:

```
use dims, only : u, v
#ifdef meteo_ecmwf_extra
use dims, only : sst
#endif
```

In this example, if the macro 'meteo\_ecmwf\_extra' is defined, then the code between '#ifdef ... #endif' is compiled, otherwise it is ignored. Wich macro's are defined is specified in the rcfile.

Note that the code should be completely rebuild if the macro definitions changes, thus use './setup\_le -n' or './run-le -c'.

## 4 Running LOTOS-EUROS - up to v1.7.1

### 4.1 Quick start

A typical way to run LOTOS-EUROS is to take following steps:

1. Go to the required version directory:

```
cd lotos-euros/v1.7
```

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

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

Modify it using a text editor.

3. Create a run environment and executable using the run script:

```
./run-le lotos-euros-test.rc
```

4. Follow the instructions written by the run script.

Probably, the instructions tell you to go to the run directory and start the submit script:

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

The following sections will provide more details.

### 4.2 The resource file

The configuration of a model is done through a text file called 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.

A simple example of an rc-file could be:

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

This assigns a value ("/data/a.txt") to a key ("le.input"). The basic format rules are:

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

More advanced usage is possible too. For a description of all features:

```
base/bin/go_pprc --help
```

The best way to learn about all configuration options is to browse through the template rc-file. The comment added should make it clear what the use of a key is and what values it could take.

### 4.3 The 'run-le' script

The 'run-le' script is the start of each run. For exact synopsis, use:

```
./run-le --help
```

Useful option is '-c', this will remove all old object and module files prior to compilation.

## 4.4 Preprocessing the rc-file

The first step take by the 'run-le' script is the preprocessing of the rc-file. The preprocessed file is given an unique name including the process id number, for example:

```
jb.12345.rc
```

Preprocessing an rc-file includes evaluation of environment variables and other features to make configuration easier. For more info, use:

```
base/bin/go_pprc --help
```

## 4.5 Collection of a source code

The second 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 first collected source files are those in the 'base/' directory. The files from the base version are then optionally overwritten by patches or project specific versions from 'proj/' directories. See section 3.3 for the concept of source projects.

## 4.6 Compilation

The next step is the actual compilation of the executable.

An important configuration choice are 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). *After changing the code, first run with checks enabled!*. See the rc-file for the selection of the flags.

Note that the subdirectory where the code is collected includes the name of the compiler and the choices for the compiler flags, e.g.:

```
<rundir>/ifort_optim-fast_check-no_debug-no/src/
```

This is done to be able to quickly switch between for example the 'fast' and the 'check' options. When switching from 'fast' to 'check' because some error popped up, this ensures that either all files were compiled with 'fast' options, or all files were compiled with 'check' options, and no mixture will occur.

## 4.7 Setup of rundirectory

The executable, pre-processed rc-file, and a submit script are installed in the run directory.

*Never edit the settings in pre-processed rc-file !* Better do that in the source directory to ensure that a run could be re-produced afterwards.

## 4.8 Submit script

Finally, the LOTOS-EUROS executable should be started using the submit script. Usage:

```
./subm-le <rc-file>
```

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), try:

```
./subm-le <rc-file> -b
```

Similar, use '-q' to submit to a queue system. For all options, use:

```
./subm-le --help
```



## 5 Running LOTOS-EUROS - v1.7.2 and higher

A new set of setup and submit script was introduced in version 1.7.2. The so-called 'pycasso' scripting is written in Python. The new scripting has the advantage over the previous 'run-le' scripting that all configurations are done through the rc-file, also the settings for compiler flags and library paths which used to be hidden in a Makefile before.

### 5.1 Quick start

A typical way to run LOTOS-EUROS is to take the steps listed here. This is 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/v1.7
```

2. Ensure that the the 'setup\_le' script is present:

```
ln -s proj/patch/002.io/bin/pycasso_setup_le setup_le
```

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

```
cp proj/patch/002.io/rc/pycasso-lotos-euros.rc \  
    pycasso-lotos-euros-test.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 lotos-euros-test.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 5.11 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 it before following the next steps:

```
cd lotos-euros/v1.7
```

### 5.3 The rc-file

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

### 5.3.1 Copy from template

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

```
cp proj/patch/002.io/rc/pycasso-lotos-euros.rc \
    pycasso-lotos-euros-test.rc
```

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

### 5.3.2 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.

A simple example of an rc-file could be:

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

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

More advanced usage is possible too. For a description of all features, see:

```
proj/patch/002.io/bin/rcget --help
```

The best way to learn about all configuration options is to browse through the template rc-file. The comment added should make it clear what the use of a key is and what values it could take.

### 5.3.3 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. The machine-specific options are collected in a machine-specific rc-file, for example:

```
proj/patch/002.io/rc/pycasso-machine-tno-azoren.rc
```

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

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

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

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

### 5.3.4 The compiler rc-file

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

```
! compiler specific settings:
#include proj/patch/002.io/rc/pycasso-compiler-ifort-v12.1.rc
```

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

### 5.3.5 *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, included into the main rcfile:

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

Don't touch this file if you don't need to.

## 5.4 The 'setup\_le' script

The 'setup\_le' script is the start of each run. It is part of the 'bin' directories, which allows that a patch version has different 'setup\_le' script than the base.

Since the script has to be called often, it will be convenient to link a recent copy of the script to the version directory:

```
ln -s proj/patch/002.io/bin/pycasso_setup_le setup_le
```

Setup a run-directory and compile an executable by executing the script with the rc-file as argument:

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

The setup script accepts a number of extra options; to see them all, 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 will create a complete new build directory by first removing the existing one. If you receive strange errors from the compiler, which might have to do with messing up old and new objects, try this option first.

Another common option is '--jobs=1' or '-j 1' which will ensure that source files are compiled one-by-one. Without this limitation, the 'make' program will try to compile as much files as possible at the same time (if they can be compiled independently), which is much faster ofcourse, but will mess up the messages printed to the screen.

## 5.5 Preprocessing of the rc-file

The first step take by the 'run-le' script is the preprocessing of the rc-file. The preprocessed file is given an unique name including the process id number, for example:

```
jb.12345.rc
```

Preprocessing an rc-file includes evaluation of environment variables and other features to make configuration easier. For an overview of all features, use:

```
proj/patch/002.io/bin/rcget --doc
```

## 5.6 The run directory

A run directory is created 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 Collection of a source code

The second step is the collection of a source code in the build directory. The first collected source files are those in the 'base/' directory. The files from the base version are then optionally overwritten by patches or project specific versions from 'proj/' directories. See section 3.3 for the concept of source projects.

The subdirectory where the code is collected includes the name of the compiler and the choices for the compiler flags, e.g.:

```
<rundir>/build.optim-none.check-all/src/
```

This is done to ensure that an executable is compiled with the same flags applied for all source files; see section 5.9 for the compiler flags settings.

## 5.8 Creation of the Makefiles

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

## 5.9 Compilation

The next step is the actual compilation of the executable.

An important configuration choice are the compiler flags. Which flags are applied is defined by a list of keywords:

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

The actual flags assigned to these keywords are set in the compiler rc-file described in section 5.3.4.

By default, the executable is compiled with the 'fast' flags to have an executable running as fast 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):

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

After changing the code, first run with checks enabled!

## 5.10 Setup of rundirectory

The executable, pre-processed rc-file, and a submit script are installed in the run directory.

*Never edit the settings in pre-processed rc-file !* Better do that in the version directory to ensure that a run could be re-produced afterwards.

## 5.11 Submit script

Finally, the LOTOS-EUROS executable should be started using the submit script.

First go to the run directory. It's location can be found in the the message displayed at the end of the setup by the 'setup.le' script, for example:

```
cd /scratch/yourname/PROJECT/test/run
```

Then execute the script with the executable and the rcfile 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), try:

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

Similar, use '--queue' to submit to a queue system. For all options, use:

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

## 5.12 Setup and submit in one step

To setup and submit in one 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 passed to 'submit.le' to submit to the background or a queue system respectively.



## 6 Tracers

### 6.1 Chemistry and/or tracer schemes

LOTOS-EUROS could run with different chemical schemes. Which tracers are present in the model depends on the scheme. The following schemes are implemented now (see also table A.1 for a summary):

- Tropospheric chemistry schemes:
  - **CBM4** : Carbon-Bond Mechanism version 4; several updates of reaction rates have been made in the past.
  - **CB99** : Carbon-Bond Mechanism 1999 ; update of CBM4, but hardly used.
  - **CB05** : Carbon-Bond mechanism 2005 : latest version; much more tracers, not used operationally yet; not part of the standard code, but a code is present at TNO.
- Selected tracer schemes:
  - **methane** : Only methane, with OH read from a previous run with full chemistry.
  - **sulphur** : Only sulphur tracers SO<sub>2</sub> and SO<sub>4</sub>, with OH read from a previous run with full chemistry.
- Aerosol schemes:
  - **prim** : Primary aerosols: black carbon, primary particulate matter; "ppm25" denotes particles 0-2.5µm (fine mode), "ppm10" denotes particles 2.5-10µm (coarse mode).
  - **sea\_salt** : Sea-salt in fine and coarse mode.
  - **dust** : Dust in fine and coarse mode.
  - **sia** : Secondary In-organic Aerosols: sulphate, ammonia, and nitrate aerosols formed from gaseous tracers; this requires CBM4.
  - **soa** : Secondary Organic Aerosols: not used recently.
- Metals
  - **cat** : Cat-ions; not used recently.
  - **hm** : Heavy-Metals; not used recently.
- Other
  - **pops** : Persistent Organic Pollutants; not used recently.

The names used above are similar to the names used in the rc-file and source code to enable certain groups or tracers and the associated reactions.



## 7 Meteo input

### 7.1 Supported meteo

Meteorological fields from the following models could be read:

**ECMWF** Default meteo for operational and re-analysis runs.

**RACMO** Climate model from KNMI, used for climate-air-quality interaction studies.

**WRF** Open source meteorological model used in specific projects. Not supported in the standard code yet.

### 7.2 ECMWF meteo

The ECMWF meteo is the standard input for LOTOS-EUROS. The current default archive has the following characteristics:

- file format is grib
- domain: [-30,40] x [35,70]
- resolution 0.50 x 0.25, defined at corners of LOTOS-EUROS grid cells
- defined a 5 pressure levels (1000, 925, 850, 700, and 500 hPa); these are then interpolated to the mid of the 4 LOTOS-EUROS layers using geo-potential height fields;

An archive with ecmwf grib files looks like:

```

europe_w10e40n35n70_0.50x0.25/sfc /2007/LOTEUR_SL_20070101_fc00.gb
                               /sfc /2007/LOTEUR_SL2_20070101_fc00.gb
                               /pl-z4 /2007/LOTEUR_PL_20070101_fc00.gb
    
```

For forecast runs of the next 3 days, also files with extension `'_fc00h72.gb'` are available.

For wet-deposition tests, also some cloud fields from the lowest 22 layers of the ECMWF model are retrieved:

```

europe_w10e40n35n70_0.50x0.25/ml-z22/2007/LOTEUR_ML1_20070101_fc00.gb
                               /ml-z22/2007/LOTEUR_ML2_20070101_fc00.gb
                               /ml-z22/2007/LOTEUR_ML3_20070101_fc00.gb
    
```

The base directory of a meteo archive should be defined in the rc-file. It is also possible to define more directories which are then scanned one by one, which could be useful if meteo is available at 2 different resolutions.

To read this meteo, define the following macro's:

- `'with_meteo_ecmwf'` to enable the correct code;
- `'meteo_ecmwf_extra'` for the latest archive (with 'SL2' files).

#### 7.2.1 Rain

Rain below 0.1 mm/hour is ignored, otherwise too much rain from ECMWF input.

### 7.3 RACMO meteo

For climate studies, see rc-file for options.

To read this meteo, define the `'with_meteo_racmo'` macro to enable the correct code.

## 8 Boundary conditions

### 8.1 Methods

Table 8.1 shows a number of methods to fill boundary conditions in LOTOS-EUROS. Table A.1 shows on which tracer most common methods act.

A list should be specified to select which input sets should be applied and in which order:

```
! Provide a list with sources to apply; later sources overwrite previous:
le.bound.types      : isak const logan emep steady moztart
```

If a method applies to more than one tracer, the latest method in the list defines which value finally ends up in de bc arrays. A warning message is printed if some tracers have not been filled; these are set to zero.

### 8.2 LOTOS-EUROS boundary conditions

Only usefull in zoom mode; the boundaries outside the zoom region are read from 3D output files of a previous run.

The run that produces the boundary conditions should write all layers except for the diagnostic surface layer '0':

```
! levels to write output (0=diagnostic surface layer, 1..nz)
output.layers      : 1 2 3 4
```

Rc-file settings:

```
le.bound.types      : ... le_nc

le.bound.le_nc.path : /data/output/myproject/myrunid
le.bound.le_nc.key  : model=LE;expid=myrunid;name=conc-3d
```

will read from the files:

```
/data/output/myproject/myrunid/LE_myrunid_conc-3d-20010203.nc
:
```

method	init	description
isak		Isaksen model (2D?)
const		constant values for some aerosols, heavy-metals, and pops
logan		Logan ozone climatology
emep		following some table in some EMEP report
steady		steady-state assumption for O3/NO2/NO
le_nc	yes	LOTOS-EUROS output (NetCDF output files)
tm5_nc	yes	TM5 NetCDF output
tm5_mmix	yes	TM5 monthly mix files
moztart	yes	MACC project, MOZART model
ifs_boundary	yes	MACC project, IFS model

**Table 8.1:** Boundary condition methods in LOTOS-EUROS. Column 'init' indicates if initial fields are filled using the boundary condition fields, usually if this is a model run on a coarser grid.

## 8.3 TM5 boundary conditions

### 8.3.1 Monthly mix files

A number of TM5 simulations is available in the form of monthly averaged mixing ratio's.

- For 2000, a set of monthly mean mix files was obtained via Frank Dentener (JRC). Preferred description:

*"Boundary conditions were taken from TM5 simulations performed in the frame of IPCC/Photocomp [Dentener et al., 2006, Stevenson et al., 2006]."*

Only the global region (glb6x4 = 6 x 4 deg resolution) encloses the European domain of LOTOS-EUROS. For the eur3x2 region, the 2 rows of interface cells at the north boundary are on the northern boundary condition of LE, and these have dummy values in the files.

For other details, see the README file in the archive.

RC file settings:

```
! define TM5 archive:
le.bound.tm5_mmix.path      :  ${LOTOS}/bound/TM5/PHOTOCOMP/output
le.bound.tm5_mmix.key      :  model=TM5;expid=PHOTOCOMP;grid=glb6x4;year=2000
```

- For 2006, a set of monthly mean files was obtained via Elina Marmer (JRC). Experiments focus on impact of ship emissions on tracer concentrations in the Mediterranean.

Description is available in the form of a conference presentation [Marmer et al., 2009].

Only the global region (glb6x4 = 6 x 4 deg resolution) encloses the European domain of LOTOS-EUROS. The 3x2 zoom over the northern hemisphere reaches to 60N, which could be sufficient for runs on a smaller domain.

For other details, see the README file in the archive.

RC file settings:

```
! define TM5 archive:
le.bound.tm5_mmix.path      :  ${LOTOS}/bound/TM5/MAP/n_ships34_edgar
le.bound.tm5_mmix.key      :  model=TM5;expid=MAP-edgar;grid=glb6x4;year=2006
```

### 8.3.2 NetCDF output

TM5 output in NetCDF form following RETRO/GEMS conventions is supported.

The TM5 run for the UBA Feinstaub project provided output on a 1.00x1.00 zoom grid enclosing the domain of the RCG model. Example of the file names (daily files):

```
TM5_V1_griddef_rcg100x100.nc
TM5_V1_TP_20050101_rcg100x100.nc
TM5_V1_aero_20050101_rcg100x100.nc
TM5_V1_reac_20050101_rcg100x100.nc
TM5_V1_carb_20050101_rcg100x100.nc
TM5_V1_isot_20050101_rcg100x100.nc
```

The output has a 3 hour time resolution. A linear temporal interpolation is applied to fill boundary conditions at times for which no instantaneous field is available.

Table A.1 shows which variables are read into LOTOS-EUROS arrays. Note that not all variables might be available in every data set.

RC file settings:

```
! define TM5 archive:
le.bound.tm5_nc.path        :  ${PROJECTS}/FEINSTAUBBELASTUNG_Umweltbundesamt/TM5/tm5_out
le.bound.tm5_nc.key        :  model=TM5;expid=V2;grid=rcg100x100;dhour=3;year=2005
```

mode	name	diameters	sigma	sea-salt	dust
1	Aitken mode	0.01 - 0.1 um	1.59	x	
2	accumulation mode	0.1 - 1 um	1.59	x	x
3	coarse mode	1 - 20 um	2.00	x	x

**Table 8.2:** Aerosol size distributions in TM5.

### 8.3.3 Aerosols

In a TM5 run with chem/aero setup, the sea-salt and dust aerosols are described by size distributions in 2 or 3 modes (table 8.2).

The size distributions are assumed to be log-normal, see [Seinfeld and Pandis, 1998, chapter 7] for details. In summary, define the number distribution function:

$$\begin{aligned} n_N(\ln D) & \quad [\mu\text{m}^{-1} \text{cm}^{-3}] \\ n_N^e(\ln D) & \quad [\text{cm}^{-3}] \end{aligned} \tag{8.1}$$

by:

$$\begin{aligned} n_N(D) dD & = \text{the number of particles per cm}^3 \\ & \quad \text{with diameters within } [D, D + dD] \\ n_N^e(\ln D) d \ln D & = \text{the number of particles per cm}^3 \\ & \quad \text{with sizes within } [\ln D, \ln D + d \ln D] \end{aligned} \tag{8.2}$$

Related via:

$$n_N(D) = n_N^e(\ln D) / D \tag{8.3}$$

Total number of particles:

$$N = \int_{D=0}^{\infty} n_N(D) dD \quad [\text{cm}^{-3}] \tag{8.4}$$

The logarithm of the aerosol diameter is assumed to be normal distributed with the diameter:

$$n_N^e(\ln D) \sim \text{norm}(\ln D; \ln D_g, \ln \sigma_g, N) \tag{8.5}$$

where  $N$  is the total aerosol number,  $D_g$  is the median diameter ('geometric mean'),  $\sigma_g$  the geometric standard deviation, and:

$$\text{norm}(u; \mu, \sigma, N) = \frac{N}{\sqrt{2\pi}\sigma} \exp\left(-\frac{(u - \mu)^2}{2\sigma^2}\right) \tag{8.6}$$

The mean ('average') diameter of the aerosol population is:

$$D_m = D_g e^{3/2(\ln \sigma_g)^2} \quad [\text{m}] \tag{8.7}$$

Total aerosol mass  $M$  (kg) and total aerosol number  $N$  are related to each other via the aerosol density  $\rho$  (kg/m<sup>3</sup>) and the mean diameter:

$$M = 4/3\pi(D_m/2)^3 \rho N \quad [\text{kg}] \tag{8.8}$$

TM5 internal representation is 2 fields per mode: total mass  $M$  (kg) and total number  $N$ .

For conversion to LOTOS-EUROS we assign TM5 modes 1 and 2 to the LOTOS-EUROS 'fine' mode and TM5 mode 3 to the LOTOS-EUROS 'coarse' mode. Since the 'fine' and 'course' modes are not really well defined, the total aerosol masses are now simply copied. In future, the geometric radius and standard deviation might be used to distribute the aerosols over other size bins. Define the particle concentration for diameters in  $[0, D^*]$ :

$$F_N(D^*) = \int_{D=0}^{D^*} n_N(D) dD \quad [\text{cm}^{-3}] \tag{8.9}$$

$$= \frac{N}{2} + \frac{N}{2} \text{erf}\left(\frac{\ln D^* - \ln D_g}{\sqrt{2}(\ln \sigma_g)}\right) \tag{8.10}$$

using the wide-spread error function:

$$\operatorname{erf}(z) = \frac{2}{\sqrt{\pi}} \int_{\zeta=0}^z e^{-\zeta^2} d\zeta \quad (8.11)$$

The TM5 netcdf output contains volume mixing ratio's, with the assumed mole masses of the aerosol and the air as meta data. This is converted to (kg/m<sup>3</sup>) via:

$$\begin{array}{l} \text{dust\_c} = \text{DUST3\_M} \quad \text{xm\_dust/xm\_air air\_mass/air\_volume} \\ \\ \frac{\text{kg dust}}{\text{m}^3 \text{ air}} = \frac{\text{mol dust}}{\text{mol air}} \frac{(\text{kg dust})/(\text{mol dust})}{(\text{kg air})/(\text{mol air})} \frac{\text{kg air}}{\text{m}^3 \text{ air}} \end{array}$$

## 8.4 MACC Mozart boundary conditions

Available every day from the runs with IFS/MOZART within the MACC project

## 9 Emissions



## 10 Output



## 11 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.

### 11.1 DIADEM

#### 11.1.1 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 11.1 shows an example of how the main index could look like.

#### 11.1.2 Location of the scripts

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

```
lotos-euros/v1.7/tools/diadem
```

This directory contains scripts, settings, and some documentation.

#### 11.1.3 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.

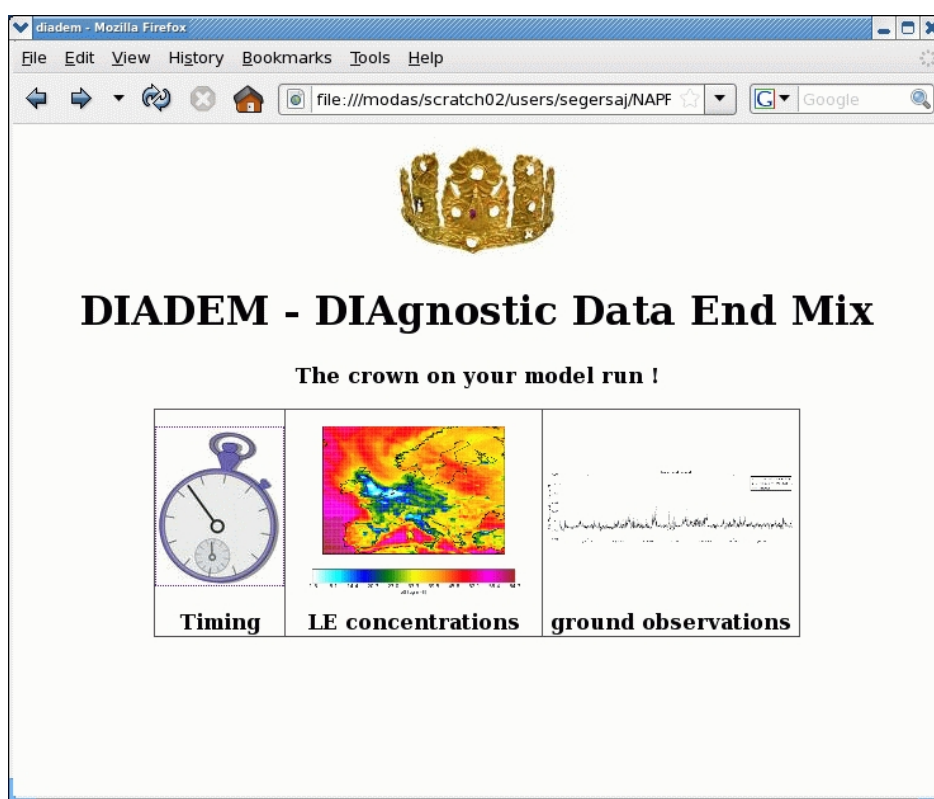
To keep the run settings and the DIADEM settings together, you may insert the content of the 'diadem.rc' file into the rcfile with the model settings and pass this one:

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

When using the pycasso scripting (v1.7.2 and higher), one could also include the special 'pycasso' DIADEM settings in the model settings with an include command:

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

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



*Figure 11.1: Example of DIADEM main index.*

## 12 Restart files



## 13 Coding conventions

How should new parts of code be written ? Some ideas for the implementation of ENTOM, *Europe's Next Top Model*<sup>1</sup>.

- Files contain either 1 module or 1 main program.
- Model specific files start with the prefix 'entom\_' :

```
entom_data.F90
entom_process.F90
:
```

- Modules have the same name as the source file:

```
module ENTOM_Process
...
end module ENTOM_Process
```

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

```
! trashcan formerly known as the 'dims' module
module ENTOM_Data
...
end module ENTOM_Data
```

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

```
ENTOM_Data                # top module
ENTOM_Data_grib ENTOM_Data_nc ... # specific
ENTOM_Data_Base           # shared entities
```

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

```
use ENTOM_Data, only : ENTOM_Data_Setup
use ENTOM_Data, only : temper, humid, tsurf
```

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

```
module ENTOM_Data

  private
  public :: ENTOM_Data_Setup
  ...

contains

  subroutine ENTOM_Data_Setup( t1, t2, status )
```

---

<sup>1</sup>(c) Copyright Astrid Manders, 2009.

```

    ...
end subroutine ENTOM_Data_Setup

...

end module ENTOM_Data

```

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

```

module ENTOM_Data

  private
  public :: the_answer
  public :: ENTOM_Data_Init, ENTOM_Data_Done
  ...

  integer    :: the_answer

contains

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

  subroutine ENTOM_Data_Done( status )
    ! nothing to be done.
  end subroutine ENTOM_Data_Setup

  ...

end module ENTOM_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 initialisation and donialisation 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( g%id, iostat=status )
  ...
end subroutine RcFile_Done

...

end type RcFile

```

- Subroutines retron 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'*. If you don't know what this means, don't use functions.



## A Tracers

The table on the next pages summaries the tracers supported in the model. The columns denote:

- which scheme selects a particular tracer;
- which data sets provide boundary conditions.

**Table A.1:** Overview of tracers in LOTOS-EUROS. Tracers selected by chemistry modes. Tracers to which boundary condition methods apply.

tracer	unit	chemical scheme			boundary conditions						
		CBM4	CB99	tracer	isak	const	logan	emep	steady	tm5_nc	tm5_mmix
NO2	ppb	cbm4	cb99		i			e	s	NO2	NO2
NO	ppb	cbm4	cb99		i			e	s	NO	NO
O3	ppb	cbm4	cb99		i		1		s	O3	O3
ETH	ppb	cbm4	cb99		i			e		ETH	ETH
OLE	ppb	cbm4	cb99		i					OLE	OLE
PAR	ppb	cbm4	cb99		i					PAR	PAR
ALD	ppb	cbm4	cb99		i			e		ALD2	ALD2
FORM	ppb	cbm4	cb99		i			e		CH2O	CH2O
XYL	ppb	cbm4	cb99		i						
TOL	ppb	cbm4	cb99		i						
CO	ppb	cbm4	cb99		i			e		CO	CO
CH4	ppb	cbm4	cb99	methane	i					CH4	CH4
SO2	ppb	cbm4	cb99	sulpher	i			e		SO2	SO2
SO4	ppb		cb99								
PAN	ppb	cbm4	cb99		i			e		PAN	PAN
MGLY	ppb	cbm4	cb99		i						MGLY
CRES	ppb	cbm4	cb99		i						
HNO2	ppb	cbm4			i						
HNO3	ppb	cbm4	cb99					e		HNO3	HNO3
NH3	ppb	cbm4	cb99			c				NH3	NH3
H2O2	ppb	cbm4	cb99							H2O2	H2O2
OPEN	ppb	cbm4	cb99								
TO2	ppb	cbm4	cb99								
ISO	ppb	cbm4	cb99								
ISPD	ppb	cbm4	?								
TERP	ppb	cbm4	?								
NO3	ppb	cbm4	cb99								NO3
OH	ppb	cbm4	cb99	meth,sulp						OH	OH
HO2	ppb	cbm4	cb99								HO2
N2O5	ppb	cbm4	cb99								N2O5
C2O3	ppb	cbm4	cb99								C2O3
XO2	ppb	cbm4	cb99								XO2

tracer	unit	CBM4	CB99	tracer	isak	const	logan	emep	steady	tm5_nc	tm5_mmix
XO2N	ppb	cbm4	cb99								XO2N
CRO	ppb	cbm4	cb99								
NTR	ppb		cb99								
O1D	ppb		cb99								
MEOH	ppb		cb99								
ETOH	ppb		cb99								
PNA	ppb		cb99								
HONO	ppb		cb99								
ROR	ppb		cb99								
O	ppb		cb99								
BC	ug/m3		prim							BC	BC
PPM25 (0-2.5um)	ug/m3		prim								
PPM10 (2.5-10um)	ug/m3		prim								
SO4a	ug/m3		sia			c		e		SO4	SO4
NH4a	ug/m3		sia			c		e		NH4	NH4
NO3a	ug/m3		sia							NO3_A	NO3_A
CG	ppb		soa								
CG1	ppb		soa								
CG2	ppb		soa								
CG3	ppb		soa								
CG4	ppb		soa								
CG5	ppb		soa								
CG6	ppb		soa								
SOA	ppb		soa								
SOA1	ppb		soa								
SOA2	ppb		soa								
SOA3	ppb		soa								
SOA4	ppb		soa								
SOA5	ppb		soa								
SOA6	ppb		soa								
PHEN	ppb		soa								
PHO	ppb		soa								
BZA	ppb		soa								
BZO2	ppb		soa								

tracer	unit	CBM4	CB99	tracer	isak	const	logan	emep	steady	tm5_nc	tm5_mmix
NPHN	ppb		soa								
Na_f	ug/m3		sea_salt							SS[12]_M	SS[12]_M
Na_c	ug/m3		sea_salt							SS3_M	SS3_M
dust_f	ug/m3		dust							DUST2_M	DUST2_M
dust_c	ug/m3		dust							DUST3_M	DUST3_M
K_f	ug/m3		cat								
K_c	ug/m3		cat								
Ca_f	ug/m3		cat								
Ca_c	ug/m3		cat								
Mg_f	ug/m3		cat								
Mg_c	ug/m3		cat								
Pb_f	ug/m3		hm			c					
Pb_c	ug/m3		hm			c					
Cd_f	ug/m3		hm			c					
Cd_c	ug/m3		hm			c					
pops			pops			c					

## Bibliography

- [Dentener et al., 2006] Dentener, F., Stevenson, D., Ellingsen, K., Noije, T. V., Schultz, M., Amann, M., Atherton, C., Bell, N., Bergmann, D., Bey, I., Bouwman, L., Butler, T., Cofala, J., Collins, W., Doherty, R., Drevet, J., Eickhout, B., H., E., Fiore, A., Gauss, M., Hauglustaine, D., Horowitz, L., Isaksen, I., Josse, B., Krol, M., Lamarque, J., Lawrence, M., Montanaro, V., Müller, J., Peuch, V., Pitari, G., Pyle, J., Rast, S., Rodriguez, J., Sanderson, M., Savage, N., Shindell, D., Strahan, S., Szopa, S., Sudo, K., Dingenen, R. V., Wild, O., , and Zeng, G. (2006). Global atmospheric environment for the next generation. *Env. Sci. Techn.*, 40:3586–3594.
- [Marmer et al., 2009] Marmer, E., Dentener, F., Aardenne, J. v., Cavalli, F., Vignati, E., Velchev, K., Hjorth, J., Boersma, F., Vinken, G., Mihalopoulos, N., and Raes, F. (2009). What can we learn about ship emission inventories from measurements of air pollutants over the mediterranean sea? *Atmos. Chem. Phys. (Discuss.)*, 9(2):7155–7211.
- [Seinfeld and Pandis, 1998] Seinfeld, J. and Pandis, S. (1998). *Atmospheric Chemistry and Physics*. John Wiley and Sons, Inc.
- [Stevenson et al., 2006] Stevenson, D. S., Dentener, F. J., Schultz, M. G., Ellingsen, K., Van Noije, T. P. C., Wild, O., Zeng, G., Amann, M., Atherton, C. S., Bell, N., Bergmann, D. J., Bey, I., Butler, T., Cofala, J., Collins, W. J., Derwent, R. G., Doherty, R. M., Drevet, J., Eskes, H. J., Fiore, A., Gauss, M. A., Hauglustaine, D. A., Horowitz, L. W., Isaksen, I. S. A., Krol, M. C., Lamarque, J. F., Lawrence, M. G., Montanero, V., Müller, J. F., Pitari, G., Prather, M. J., Pyle, J. A., Rast, S., Rodriguez, J. M., Sanderson, M. G., Savage, N. H., Shindell, D. T., Strahan, S. E., Sudo, K., and Szopa, S. (2006). Multi-model ensemble simulations of present-day and near-future tropospheric ozone. *J. Geophys. Res.*, 111(D8). doi:D0830110.1029/2005JD006338.