



## Outline

- Introduction
- «history» files
- «restart» files
- controlling debug printing



## Introduction

LMDZ outputs include :

- «**history**» files : they gather instantaneous or averaged diagnostic variables
- «**restart**» files : used to restart or extend a simulation
- the «**output**» file: collects all control and error messages



## Introduction

The «history» and «restart» files in LMDZ are all in **NetCDF** format and written using either the **NetCDF** or **IOIPSL/XIOS** libraries.

The **IOIPSL** and **XIOS** library were developed at IPSL for model I/Os using the **NetCDF** library.

For «history» files, they allow variables to be « manipulated » (e.g. average/max/min) before being written out with **XIOS** providing many more possibilities than **IOIPSL**

The output of variables consists in 2 steps :

- definition of the variables to output (during initialisation of the run)
- computation and writing of the variables (during the simulation)

Further information : there will be examples using the **IOIPSL** and **XIOS** library later on. The **XIOS** library is fully fonctionnal and has replaced the **IOISPL** library within the modipsl/libIGCM framework.



## Introduction

The partition between the dynamical module and the physical module in the LMDZ code and the fact that one can be run without the other implies that both modules need to have their own «**restart**» and «**history**» outputs :

- «**restart.nc**» for the dynamics module and «**restartphy.nc**» for physics module
- «**history**» file for the dynamics module only consists of the variables of state (U, V, T, Q, Ps) at two frequencies : instantaneous and averaged.

When run in parallel mode with **IOIPSL**, each process writes its own history files in its domain. The global file is reconstructed from these various files by using the **rebuild** utility Distributed with the **IOIPSL** library. When using the **XIOS** package, **XIOS** can provide this rebuilding mechanism automatically.



## «history» outputs of the physics module :

- Scheme which allows to individually control the output of the variables in 9 files :
  - 5 basic files : ***hismth.nc, histday.nc, histhf.nc, histins.nc, histLES.nc***
  - 1 specific file (data sites) : ***histstn.nc***
  - 3 files with predefined pressure levels : ***hismthNMC.nc, histdayNMC.nc, histhfNMC.nc***  
***outputs on standard levels pressures :***  
1000., 925., 850., 700., 600., 500., 400., 300., 250., 200., 150., 100., 70., 50., 30., 20., 10.hPa

*(names of files can be changed easily)*

- There are 3 specific «history» outputs files :  
***hismthCOSP.nc, histdayCOSP.nc, histhfCOSP.nc***  
**Outputs from the COSP simulator**

**By using XIOS, this basic scheme can be redefined totally, e.g. the CMIP6 IPSL workflow in which LMDZ outputs files containing single timeseries of requested variables**



## «history» outputs of the physics module (principle):

- To each «history» file "histX.nc", an output level, **lev\_histX**, is associated
- To each variable, an output level **flag\_D** is associated

Then if

**flag\_D**  $\leq$  **lev\_histX** ==> variable "D" is defined and written in the file "histX.nc"

- These control keys are defined in the files **config.def/output.def**
- Each output file can be controlled to :
  - activate it
  - define its name,
  - define its output frequency,
  - define the mathematical operation on the output variables (average, max/min ...)
  - output the variables on the whole or a limited domain

# LMDZ outputs



## «history» outputs of the physics module (in practice):

In the **config.def** or **output.def**, file :

✓ define the control keys of the various files :

```
# Activate or not the output of the file :
phys_out_filekeys=      y      y      n      y      n      n      n      n      n
# Name of files :
phys_out_filenames=  histmth  histday  histhf  histins  .....  histhfNMC.nc
# Output level of files
phys_out_filelevels=   5        2        2        4        .....  5
# Archive operation
phys_out_filetypes=   ave(X)   ave(X)   ave(X)   inst(X)  .....  inst(X)
# Frequencie (mounths, mth, mois, day, days, jour, jours, heure, mn, TS, ... )
phys_out_filetimesteps= 30day   1day    6hr     1TS     .....  6hr
```

✓ define the control keys of each variable :

```
# Vertical wind
flag_vitw   =      2      3      7      6      .....  10
name_vitw   = vitw
```

# LMDZ outputs



«history» outputs of the physics module (in practice):

In the **config.def** or **output.def file** :

```
# Names of files :
phys_out_filenames = histmth.nc  histday.nc  histhf.nc  histins.nc  .....
# Activate or note the output on a limited domain
reg_out_fkeys =          n          n          y          n          .....

# Longitude min and max of domain
phys_out_lonmin =      -180      -180          0      -180      .....
phys_out_lonmax =      +180      +180          90      +180      .....

# Latitude min and max of domain
phys_out_latmin =       -90       -90       -30      -90      .....
phys_out_latmax =       +90       +90       +40      +90      .....

# Vertical level min and max
phys_out_levmin=         1         1         1         1      .....
phys_out_levmax=        39        39         3        39      .....
```

›Number of variables controled by this mechanism (approx.)  
**300 2D-fields and 150 3D-fields**





## «history» outputs of the physics module (in practice) How to add a variable

2 routines need to be modified:

- **libf/physlmd/phys\_output\_ctrlout.F90** :
  - declaration of name, description, unit and output level of the new variable
- **libf/physlmd/phys\_output\_write.F90** :
  - write the new variable in each file

Example for «t2m\_min» variable :

**.../libf/physlmd/phys\_output\_ctrlout.F90**

```
type(ctrl_out),save :: o_t2m_min = ctrl_out((/ 1,1,1,5,10,10,5,5,5 /), &  
      't2m_min','Temp 2m min', 'K', (/ 't_min(X)', 't_min(X)', 't_min(X)', &  
      't_min(X)', 't_min(X)', 't_min(X)', 't_min(X)', 't_min(X)' /))
```

**.../libf/physlmd/phys\_output\_write.F90**

```
CALL histwrite_phy(o_t2m_min, zt2m)
```



## «history» outputs of the physics module (specific files)

- The same mechanism applies to the COSP simulator
- Routines :
  - `.../libf/cosp/cosp_output_mod.F90`
  - `.../libf/cosp/cosp_output_write_mod.F90`



## The XIOS library

IOIPSL is progressively replaced by XIOS

XIOS : XML – IO – SERVER

Library developed at LSCE/IPSL for management of IO of climate codes

Based on client – server principle : IO server manages the outputs so that the climate code does not waste time on its outputs, it just sends them to the IO server

All aspects of the outputs (name, units, post-processing frequencies, operations, ...) are controlled by external xml files but can be overridden from the source code.



## The XIOS library

Advantages of **XIOS** library :

- Client – server model : code not slowed down by outputs
- Flexible
- Minimal modification to the simulation code
- No recompilation of code necessary when changing IO definitions (all done in xml files)
- Particularly geared towards parallel code (no more rebuilding output files, asynchronous writes ...)
- XML and its concept of inheritance
- Lots and lots of functionalities : reading files, filters and transformations, on-the-fly interpolation to a particular grid, on-the-fly timeseries output

« Disadvantages » :

- Another library to install
- Needs a mpi library



## XIOS in LMDZ

In working order. Is the default output mode for the new IPSLCM6 configuration.

**Use :**  
Need to download XIOS from <http://forge.ipsl.jussieu.fr/ioserver/> and compile it.  
(see tutorial)

### LMDZ compile :

3 options to the makelmdz/makelmdz\_fcm command :

- **-io ioipsl** only IOIPSL
- **-io mix** both IOIPSL and XIOS
- **-io xios** only XIOS

**ok\_all\_xml = y** in run.def file to get xml to control everything

Sample xml files in DefLists directory

To add variables to the « XIOS » output files, you will need to add them as before to the 2 routines `phys_output_ctrlout.F90`, `phys_output_write.F90` and add them to the xml files

# LMDZ outputs



## XIOS in LMDZ : the xml files

One file, at least must be present (model will crash if this file is absent) :

**iodef.xml :**

```
<?xml version="1.0"?>
<!--===== -->
<!-- iodef.xml : Configuration file for production of output files using XIOS -->
<!--           This file contains the context XIOS. -->
<!--===== -->

<simulation>
  <context id="xios">
    <variable_definition>
      <variable_group id="server">
        <variable id="using_server2" type="bool">>false</variable>
        ....
      </variable_group>

      <variable_group id="buffer">
        .....
      </variable_group>

      <variable_group id="parameters" >
        <variable id="using_server" type="bool">>false</variable>
        <variable id="info_level" type="int">1</variable>
        ....
      </variable_group>
      ....
    </variable_definition>
  </context>
```

# LMDZ outputs



## XIOS in LMDZ : the xml files

### context\_lmdz.xml :

```
<! Context LMDZ >
<!-- <calendar type="D360" start_date="1980-01-01 00:00:00" /> -->

<! Define available variables >
<field_definition src="./field_def_lmdz.xml"/>

<! Define output files
    Each file contains the list of variables and their output levels >

<file_definition src="./file_def_histday_lmdz.xml"/>
...

<! Define domains and groups of domains >
<domain_definition>
    <domain id="dom_glo" data_dim="2" />
</domain_definition>

<! Define groups of vertical axes >
<axis_definition>
    <axis id="presnivs" standard_name="Vertical levels" unit="Pa"/>
    ...
</axis_definition>

<grid id="grid_glo_presnivs">
    <domain domain_ref="dom_glo" />
    <axis axis_ref="presnivs" />
</grid>

</context>
```

# LMDZ outputs



## XIOS in LMDZ : the xml files

### field\_def\_lmdz.xml :

```
<field_definition level="1" prec="4" operation="average" freq_op="1ts" enabled=".true."
default_value="9.96921e+36">

  <field_group id="fields_2D" domain_ref="dom_glo">
    <field id="phis"      long_name="Surface geop.height"      unit="m2/s2" />
    <field id="ffonte"    long_name="Thermal flux for snow melting"    unit="W/m2" />
    ...
  </field_group>
  <field_group id="fields_3D" domain_ref="dom_glo" axis_ref="presnivs">
    <field id="tke"      long_name="TKE"      unit="m2/s2" />
    ...
  </field_group>

  <field_group id="fields_NMC" domain_ref="dom_glo" axis_ref="plev">
    <field id="ta" long_name="Air temperature" unit="K" />
    ...
  </field_group>

  <field_group id="fields_COSP_CALIPSO" domain_ref="dom_glo" freq_op="3h">
    <field id="cllcalipso"      long_name="Lidar Lowlevel Cloud Fraction"      unit="1" />
    ...
  </field_group>
</field_definition>
```



# LMDZ outputs



## XIOS in LMDZ : the xml files

### file\_def\_histday\_lmdz.xml :

```
<file_definition>
  <file_group id="defile">
    <file id="histday" name="histday" output_freq="1d" output_level="2" enabled="TRUE"
compression_level="4">
      <! VARS 2D >
        <field_group operation="average" freq_op="1ts">
          <field field_ref="phis" level="1" />
          ...
          <field field_ref="ffonte" level="10" />
          ...
          <field_group operation="average" freq_op="1ts"
detect_missing_value=".true.">
            <field field_ref="u850" level="7" />
            ...
          </field_group>
        </field_group>
      <! VARS 3D >
        <field_group operation="average" freq_op="1ts" axis_ref="presnivs">
          <field field_ref="cldtau" level="5" />
        </field_group>
      </file>
    </file_group>
  </file_definition>
```



## COSP outputs

### COSP (CFMIP Observation Simulator Package) :

- To evaluate the representation of cloud process in the models :  
The simulator is a diagnostic code that computes pseudosatellite observations from model variables in order to compare them to real satellite observations.
- COSP has simulators for these satellite cloud products :  
ISCCP, CALIPSO, CLOUDSAT, PARASOL, MISR, MODIS

# LMDZ outputs



## COSP outputs

COSP implemented in LMDZ model :

Directory : `.../libf/phy/md/cosp`

Outputs routines :

`.../libf/cosp/cosp_output_mod.F90`

`.../libf/cosp/cosp_output_write_mod.F90`

To run LMDZ simulation with COSP simulator :

Compile LMDZ model with option :

`-cosp none/v1/v2`

Activate COSP in LMDZ simulation :

`ok_cosp=y`

in \*def file

# LMDZ outputs



## COSP outputs

Organization of COSP output files : the same as LMDZ history files.

3 outputs files for COSP include cloud diagnostics :

*hismthCOSP.nc, histdayCOSP.nc, histhfCOSP.nc*

The control keys of files and variables :

✓ With IOIPSL library :

<code>cosp_outfilenames</code>	=	<i>hismthCOSP.nc</i> ,	<i>histdayCOSP.nc</i> ,	<i>histhfCOSP.nc</i>
<code>cosp_outfilekeys</code>	=	y	y	n
<code>cosp_ecritfiles</code>	=	1mth	1day	3h
<code>cosp_outfiletypes</code>	=	ave(X)	ave(X)	inst(X)
<code>cles_cllcalipso</code>	=	y	y	n
<code>name_cllcalipso</code>	=	LowCldMth	LowCldDay	LowCldHf

# LMDZ outputs



## COSP outputs with XIOS

*Organization of COSP output files : the same as LMDZ history files.*

The control keys of files and variables :

✓ With XIOS library :

Variables defined in XML file:

**field\_def\_lmdz.xml**

Content of files defined in XML files :

**file\_def\_histmthCOSP\_lmdz.xml,**  
**file\_def\_histdayCOSP\_lmdz.xml,**  
**file\_def\_histhfcOSP\_lmdz.xml**



## « Restart » files

The dynamical and physical modules each write their own restart file (**restart.nc** et **restartphy.nc**). These files save the state variables that the model needs at each time step so that the model can be restarted without losing continuity (in practical terms this is known as «**1+1=2**»)

Routines involved in this process are :

**.../libf/dyn3d/dynredem.F** for the dynamical module (the restart state being read in by **.../libf/dyn3d/dynetat0.F**)

**.../libf/physlmd/phyredem.F** for the physics module (corresponding routine in **.../libf/physlmd/phyetat0.F**)

These routines do not use the **IOIPSL** library and are interfaced directly with the **NetCDF** library.

# LMDZ outputs



## Controlling output messages :

Most of control outputs and messages are written to standard output (the screen) by the use of commands such as:

**print\***, ... or **write(\*,\*) 'ma variable =',... .**

A mechanism exists to output these messages to a file rather than the screen.

One just needs :

- to include in any new routine, the `iniprint.h` file which defines and shares 2 parameters :

**lunout** : a unit number corresponding to the output file

(if `lunout`  $\neq$  6 ==> a `lmdz.out` file is created and assigned to this number)

**prt\_level** : an output level

The value of these two parameters can then be modified in the `run.def` file

- To use them in the routine you then just need to add lines such as :

```
IF (prt_level>9) WRITE(lunout,*) 'pas de convection'
```

While keeping small values of **prt\_level** for really important messages

We also have introduced a routine called `prt_alerte` to print informative alert messages :

see [https://lmdz-forge.lmd.jussieu.fr/LMDZPedia/HowTo:\\_Print\\_alert\\_messages](https://lmdz-forge.lmd.jussieu.fr/LMDZPedia/HowTo:_Print_alert_messages) for information

# LMDZ outputs



## Output messages : what use are they ?

If the model crashes or does not seem to run properly, these output messages should give you an indication of what's going on.

The first thing to do is

```
grep 'Houston, we have a problem ' output_file
```

as the model will output this string with an indication of the problem it encountered, when the problem has been anticipated by the developers (might be a configuration problem, a temperature that's suddenly out of range, ...).

If the string is not found and no obvious error (segmentation fault, memory violation, floating point exception) can be found in the output messages, the best bet is to recompile the model with the `-debug` option and run it again. It should now give you an indication of the line and the routine that is causing the crash. Once found, you can start debugging the routine or call for help with some vital information.