# NEMO Validation Tools Kit (NVTK)

June 2008, version 1.2
Claude Talandier, Rachid Benshila (NEMO system team)
opatlod@locean-ipsl.upmc.fr

## Introduction:

The aim of the NEMO Validation Tools Kit (NVTK hereinafter) is to perform validation tests on the reference configurations to check the reliability and the continuity of the NEMO system are still ensured with new modifications/developments; furthermore NVTK allows also checks if they alter results relatively to the last reference version. These are fundamental steps before any modifications/developments integration into the reference code.

Validation tests, described in part **I-NEMO validation set**, are heavy and take time to be realised "by hand", so the NVTK allows to apply them quickly and simply on the NEMO's reference configurations, i.e. ORCA2_LIM , GYRE and GYRE_LOBSTER.

NVTK relies on the classical modipsl environment under which the NEMO code is currently managed and is plugged as an additional "layer"; its description and installation are done in **II-NVTK environment / organisation** with information to modify/improve it. It is based on Makefiles to control the compilation process and ksh shell scripts to manage jobs submission. The whole process is executed with few commands and ended with an assessment report which summarize the success or not of validation set.

NVTK allows to perform tests simply and quickly on new developments/modifications before there implementation into the reference code. Developers are strongly encouraged to use it to test their work before sending them to the NEMO team. NVTK can be used on the following platforms: NEC-SX8 and IBM-SP4 (French IDRIS center) and MAC (based on xlf compiler) and will require changes for any new target.

# I – NEMO validation set

The aim of this NEMO Validation Tools Kit  is to perform fixed standard tests (detailed below) to ensure the reliability and continuity of the NEMO system even with new modifications and before incorporating them into the reference code. For the reference configurations ORCA2_LIM, GYRE and GYRE_LOBSTER the following behaviour are controled:

- the **reproductibility**,
- the **restartability**,
- the **memory**,
- the **performances**,

## I-a Reproductibility

This point is ensured as soon as the output solver.stat (see the methodology detailed below) file from a MPI or Open-MP multi-processors simulation equal bit-to-bit to the one obtained in using a single processor. So to valid this feature, at least 2 simulations must be performed: one Single Processor Run (SPR hereafter, which is the reference one) and one MPI (or Open-MP) Multi-Processor Run (MPR hereafter). We define what we called in this document the “run type” which corresponds to the number of processors used:

- **mon:** mono processor run
- **mpi**: multi-processors run based on Message Passing Interface (MPI)
- **omp**: multi-threads run based on Open-MP

**Note**: the NVTK environment allows to test only one domain decomposition in one step. A **MxN** decomposition is the default one, but it will be maybe usefull to complete it with a **Mx1** and **1xN** decomposition to test specific cases as the North fold condition of ORCA’s family grid.

## I-b Restartability

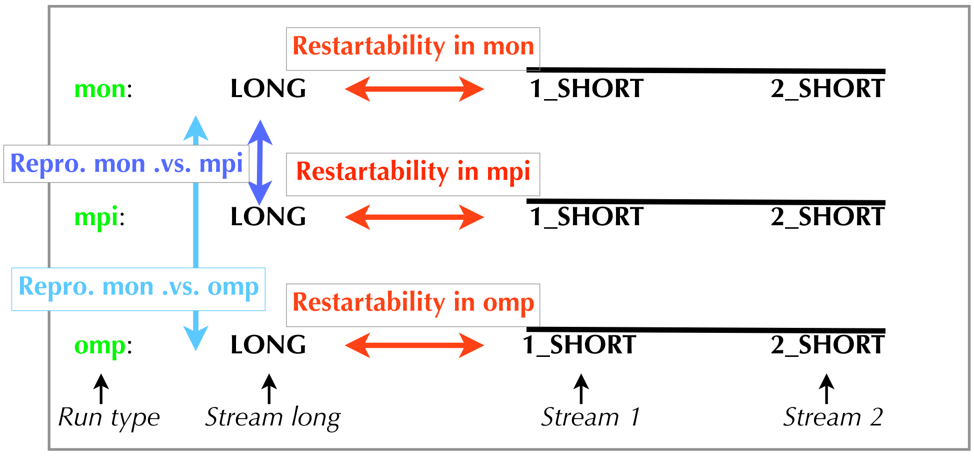
It relies on the strict equality bit-to-bit of the solver.stat file whatever the way the time period of the simulation is realised; i.e. in one single stream or many sub-stream (using restart files) so that a single stream time = sum (sub-stream time) for a given period. For a given run type **mon**, **mpi** or **omp** 3 runs are performed:

- **LONG** : long run (*stream long*)
- **1_SHORT**: short run (*stream 1*)
- **2_SHORT**: short run using restart files from the **1_SHORT** run (*stream 2*)

Time length simulations are hard coded into jobs scripts of each configurations (see in *./modipsl/config/NVTK/INSTALL/JOBS directory*):

- for ORCA2_LIM: 14 days (210 iterations) with 2 streams of 7 days for restartability check
- for GYRE/GYRE_LOBSTER: 30 days (360 iterations) with 2 streams of 15 days for restartability check

Reproductibility and restartability checks are summarised by the following schematic view:



The **reproductibility** is checked between run types (mon .vs. mpi & mon .vs. omp) over “LONG” streams
The **restartability** is checked for each run type (mon, mpi & omp) between stream long and Stream 1 + Stream 2

**Methodology to check both reproductibility and restartability:**
It is based on the comparison between solver.stat output ASCII files from each run type (mon .vs. mpi and/or mon .vs. omp); this file gather information from the elliptic solver used and give 3 information: number of iterations, the residue of the convergence and the second member of the system solved values.
These 3 fields must be the same bit-to-bit (under some restrictions such as not using key_tradmp cpp key and set the namelist parameter nbit_cmp=1) whatever the run type realised. This is possible only when using the Red Black Successive.Over.Relaxation S.O.R algorithm which has the property to perform the computation exactly the same way in mon, mpi or omp (the PCG elliptic solver does not have this behaviour). So all runs are realised in setting the nsolv parameter in the ocean namelist to 2 (II-c Launching jobs and assessment report).

**Note:** For configurations tested, this method can be applied because the filtered free surface numerical option is used (default option) and requires to solve an elliptic system. If using the time-splitting option, no elliptic solver is used and the only way to check both features will be to compare directly fields from restart files (fields are saved in double precision).

## I-c Memory

This check is done to ensure that the whole memory does not increase too much when adding new developments. This is closely related to the configuration since it depends on physical/numerical cpp keys activated. The global domain of a configuration must be considered here, i.e. when compiling a configuration in **mon** run type, i.e. without MPI domain decomposition.

3 memory levels are summarised:
- ***Stack memory***: corresponds to the *lower bound memory* since it does not take into account local variables i.e. those inside subroutines,
- ***Static memory***: corresponds to the *upper bound memory* since every hard coded variables are considered even code sections in which the configuration would not go through,
- ***Effective memory***: it is the maximum memory used during the run taking into account dynamical allocation and automatic arrays are also listed

Stack and Static memory result from the UNIX command "*size*" applied to the binary after the compilation process; usually the default compilation option is set to store variables in Stack mode whereas an explicit compilation option must be used (-P Static on NEC, -qsave on IBM) to access the Static memory.
The effective memory is obtained in using environment variables which depends on the considered target. This is a current data when making global timing of a code (see hereafter).

## I-d Performances

One ensure that time consuming and optimisations are not completely changed or simply removed with new modifications. 2 performance levels can be distinguished:
- global level: the total time consuming,
- subroutines level: detail the time used by each subroutines

Only the first one is checked here, the subroutine level is performed when over time consuming is detected or when a specific subroutine should be optimised. We perform an one year simulation for each configuration's run type to bypass initialisation step effects. Job/results is/are launched/stored from/into the **GTIME** directory.

At present performances are checked in setting environment variables (specific to each target) just before running one configuration. They summarise at least basic information such as, the real time and the CPU time consumed; the maximum memory used and the MFLOPS.

In the forthcoming version of NEMO, an internal timing will be available and will allow to check performance at subroutines levels.

The 4 validation points imply the following structure to manage simulations results:

```
NEMO_VALID/WNAME_CONFIG/
|-- mon                      --> SPR run
|    |-- 1_SHORT
|    |-- 2_SHORT
|    |-- LONG
|    `-- GTIME
|-- mpi                      --> MPR run
|    |-- 1_SHORT
|    |-- 2_SHORT
|    |-- LONG
|    `-- GTIME
`-- omp                      --> MPR run
     |-- 1_SHORT
     |-- 2_SHORT
     |-- LONG
     `-- GTIME
```

## II – NVTK environment / organisation

### II-a Download and install (via *ins_nvtk.ksh* script ) the appropriate environment

This step must be done only once under each modipsl directory. It addresses the download of necessary scripts/files for the validation and the build of end directories to launch jobs and to store results.

**A - Downloading phase:**

i) In one step, user can download: modipsl architecture, the latest NEMO code (SVN trunk), the reference configurations and NVTK under the TEST directory.

```
> mkdir TEST
> cd TEST
> svn_ano                   (download the modipsl structure)
> cd modipsl/util
> ./model NEMO_DEV          (download the NEMO code)
```

**Note:** *svn_ano* is an alias' associated to modipsl structure download. It refers to the following command: 'svn co http://forge.ipsl.jussieu.fr/igcmg/svn/modipsl/trunk modipsl '

Description of the NVTK sub-directories and scripts:

```
NVTK/
|-- 2TEST
|-- INSTALL
|   |-- CONFIG_FILES
|   |-- JOBS
|   |-- MODIPSL_FILES
|   `-- ins_nvtk.ksh
|-- Makefile
|-- cfg.txt
|-- fait_AA_make
`-- use_cfg
```

The **NVTK** directory:
this is the core of NVTK; Makefile allows to control the tests to perform (see I - NEMO standard tests procedure), usual scripts used to build dependencies BB_make file for a configuration (*fait_AA_make*) or to build a new configuration under the same modipsl architecture (*use_cfg, cfg.txt);* **2TEST** is a storage directory for Fortran modules to be tested.

# **INSTALL**: contains the shell script *ins_nvtk.ksh* to be use to install the necessary environment

# **INTALL/CONFIG_FILES:** contains *AA_make** and *BB_make.ldef** for each configuration modified and that will be used to build Makefiles

# **INTALL/JOBS:** contains jobs (*job_configname.ksh*; headers and scripts to treat timing results for each platforms (*jhd_target_runtype, CPU_time_target.ksh*); *lance_batch.ksh*, *cron_jobs.ksh*, *assessment.ksh* are respectively scripts dedicated to launch jobs, check the existence of jobs in the batch queue and build an assessment report to be sent to the user's e-mail

# **INTALL/MODIPSL_FILES:** stores the modipsl *ins_make* lightly modified script used to build Makefiles

Build the appropriate environment:

The *ins_nvtk.ksh* script allows to install the necessary environment through 5 steps detailed below (they are performed for the 4 configurations ORCA2_LIM, GYRE, GYRE_LOBSTER and ZAGRIF):

I. **Install modified/new shell scripts in appropriate directories of modipsl**
II. **Build configuration's environment with new directories WORK and MY_SRC and BB_make file dependencies**
III. **Build sub-directories from where jobs will be launched and in which results will be stored**
IV. **Modify *lance_batch.ksh*, *assessment.ksh* scripts and *job_CONFIGNAME.ksh* depending target and user's input information**
V. **Performs preliminary compilations for IOIPSL & AGRIF packages**

**B - <u>Installing phase:</u>**

Before executing the *./modipsl/config/NVTK/INSTALL/ins_nvtk.ksh* shell script, user has to fill the following part in the header of this script:

```
###########################################################
##### Begin Users modifications
###########################################################
# OUTDIR   : working directory from which jobs will be launched & results stored
# INPUTD   : directory where to get ORCA2_LIM_nemo_v2.tar
# DECOMP   : total number of processors which will be used
# MAIL     : give your e-mail
# UAGRIF   : one of the standard configurations is based on AGRIF yes/no
###########################################################
OUTDIR=/workdir/rech/eee/reee831
INPUTD=/u/rech/eee/reee831/IO_NEMO_ORCA2_LIM/
DECOMP=8
MAIL="your_email@xxxxxx"
UAGRIF=yes
###########################################################
##### End   Users modifications
###########################################################
```

**Note**:
- the **OUTDIR** & **INPUTD** must be directories accessible through classical UNIX *cd, cp ...etc* commands.
- the total number of processors DECOMP could be modified later directly in scripts
- the MAIL variable corresponds to the e-mail address to which assessment report will be sent

The 5 steps above are detailed just below:

## I. Install modified/new shell scripts in appropriate directories of modipsl

- From *./modipsl/config/NVTK/INSTALL/MODIPSL_FILES* directory:
the *ins_make* script includes some modifications to be able to build the Makefile of a specific directory in using it as: "*ins_make* -w DIRECTORY NAME". It is copied in the ./modipsl/util directory and replace the standard one.
- From *./modipsl/config/NVTK/INSTALL/CONFIG_FILES* directory:
the *AA_make/AA_make.ldef* files are copied under each ./config/CONFIG_NAME directory; they both include modifications which manage the compilation process and the launch of batch jobs.
the *BB_make.ldef_CONFIG_NAME* include cpp keys of each configuration

## II. Build configuration's environment with new directories WORK, MY_SRC and BB_make file dependencies

```
GYRE/
|-- AA_make
|-- AA_make.ldef
|-- EXP00
|   |-- AA_job
|   `-- namelist
|-- MY_SRC
|   `-- par_oce.F90_keep
|-- scripts
|   |-- BB_make
|   `-- BB_make.ldef
`-- WORK
```

For each configuration:
- build the WORK directory directly under the *./modipsl/config/CONFIG_NAME* in making appropriate links depending the necessary components OPA_SRC, ...etc
- build the MY_SRC directory with the par_oce.F90_keep file in which the user has to fix the MPI domain decomposition
- rebuild dependencies file ./modipsl/config/CONFIG_NAME/scripts/*BB_make* to take into account modifications of the environment.
The final tree for each configuration looks like the left figure, e.g. GYRE.
The EXP00 directory remain unchanged and gather all basic input files such as namelist files which will be used for runs. These files are modified directly in the script job associated to the configuration tested.

## III. Build sub-directories from where jobs will be launched and in which results will be stored

Following the validation test procedure (see I - NEMO validation tests procedure), the following tree is built under the OUTDIR directory specified by the user for each configuration:

```
NEMO_VALID/WNAME_CONFIG/
|-- mon                               --> MONO run
|   |-- 1_SHORT/REF
|   |-- 2_SHORT/REF
|   |-- LONG/REF
|   `-- GTIME/REF
|-- mpi                               --> MPI run
|   |-- 1_SHORT/REF
|   |-- 2_SHORT/REF
|   |-- LONG/REF
|   `-- GTIME/REF
`-- omp                               --> Open-MP run
    |-- 1_SHORT/REF
    |-- 2_SHORT/REF
    |-- LONG/REF
    `-- GTIME/REF
```

If a reference tag is specified by the user (see the variable **REF_TAGV** in the Makefile under *./config/NVTK/Makefile*); *solver.stat*, *memory_size.txt* and *runtypeCONFNAME_err* files, associated to this tag are retrieved and stored under each stream *REF* directory. They are used to compare the current version with reference run to check if results changed are not.

## IV. Modify *lance_batch.ksh, assessment.ksh* scripts and *job_CONFIGNAME.ksh* jobs

Information gave by user in the *ins_nvtk.ksh* script header (in blue below) are used into the files:
- *lance_batch*.ksh*:* **IODIR** = ***INPUTD*** and **PRC** = ***DECOMP*** variables
- *assessment.ksh*: **TARGET*** and **EMAIL =** ***MAIL*** variables
- *job_CONFIGNAME.ksh*: **TARGET*** variable
* this variable is filled using the script called *./modipsl/util/w_i_h* which gets the current target name

A description of these files is done in part *II.c. Launching jobs and assessment report*

## V. Performs preliminary compilations for IOIPSL & AGRIF packages

This step is necessary here since files compilation of one configuration is based on UNIX multi-process possibility; for that external libraries must be already compiled.

## II-b Compilation process

The leitmotiv is to be able to test specific modifications (numericals, physicals ...) on all standard configurations in only few commands (both the compilation and the running processes). This is possible because the tests to perform are fixed. We make hereafter the description of ***A) Launching compilation phase*** which is of main interest, followed by ***B) Chaining Makefiles*** and finished with ***C) Compilation options and cpp keys***.

```
NVTK/
|-- 2TEST
`-- Makefile
```

The *./modipsl/config/NVTK* directory has been added to the modipsl environment and is a kind of control location for the user. Indeed, only 3 steps are required to launch compilations and runs whatever the number of configuration and the run type specified (supposing the target machine specifications have been added, see part II-d Add a new target ? ). All the compilation process relies on Makefiles:

**A - *Launching the whole process : compilation, jobs and final report***

i) store files under *./modipsl/config/NVTK/2TEST*
ii) set the 7 following variables in the *./modipsl/config/NVTK/Makefile* file:

- **NAM_V:** name of the test (this information will be used in the assessment report at the end of all simulations)
- **LISTE_CONF:** name of standard configurations to test with modified routines under 2TEST directory
- **JOBS_2LAUN:** select jobs to launch
- **BUILD_MAKE:** compilation type (run type) to perform: mono-processor and/or multi-processors MPI and/or Open-MP (this list is applied to each configuration specified in the LISTE_CONF variable). *If **mpi** is specified, the user must set the processor cutting (jpni, jpnj & jpnij parameters) in the par_oce.F90_keep module stored in each ./config/CONFIG_NAME/MY_SRC directory. This MPI decomposition must be the same for the 3 configurations*
- **MAK_TIME:** perform a timing check
- **MAK_MEMO:** perform a memory check
- **REF_TAGV**: (optional) reference tag version to which compare results

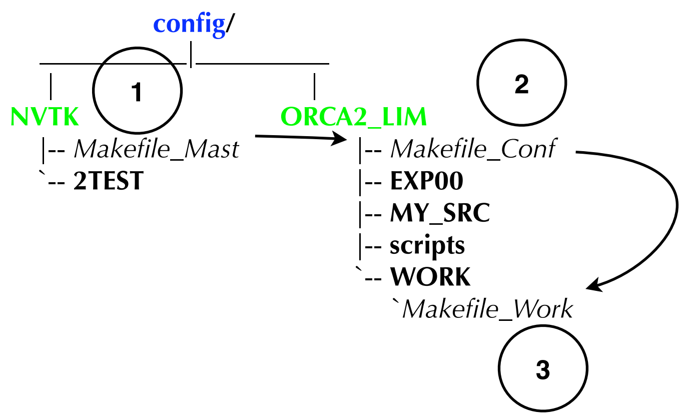
iii) launch the compilation process using *gmake* command from *./modipsl/config/NVTK/2TEST*

```
#- Name of the test
NAM_V = test1
#
#- Configurations to be tested
LISTE_CONF = ORCA2_LIM  GYRE  ZAGRIF
#-
#- Jobs to launch use keyword: nojob, all, long,
short or gtime
JOBS_2LAUN = all
#
#- Compilation list type to perform, mon (mono)
#  &/or mpi (MPI) &/or omp (Open-MP)
BUILD_MAKE = mon mpi
#-
#- Proceed to a timing, use key word 'timing' or
'notiming'
MAK_TIME = notiming
#-
#- Proceed to a memory check, use key word
'memo' or 'nomemo'
MAK_MEMO = nomemo
#
#- Reference Tag version (optional)
REF_TAGV = nemo_v2_3
#
```

**Note:** *since this environment has been built to use UNIX multi-process possibilities, it is possible to speed-up the compilation in using the gmake option -j : "gmake -j number_of_configurations_to_compile"*

**B - Chaining Makefiles:**

The *./modipsl/config/NVTK/Makefile* can be considered as a "master" Makefile (hereafter *Makefile_Mast*) since it controls 2 other ones located under each configuration directory; for commodity in the following we distinguish them with extension name respectively *Makefile_Conf* and *Makefile_Work*. See the sketch below for the **ORCA2_LIM** configuration as example:



The *Makefile_Mast* controls, for a given configuration, the *Makefile_Conf* located just under the *./config/CONFIG_NAME* directory which itself controls the *Makefile_Work* under *./config/CONFIG_NAME/WORK*. Now we detail majors tasks assigned to each of them:

***Makefile_Mast:*** from ./config/NVTK directory
performs 3 steps for each configuration set in the **LISTE_CONF** variable,

- *Step 1:* build links from *./config/NVTK/2TEST* to *./config/CONFIG_NAME/MY_SRC* directory
- *Step 2:* build *Makefile_Conf* & *Makefile_Work* Makefiles under respectively *./config/CONFIG_NAME* and *./config/CONFIG_NAME/WORK* directories using the command *ins_make*
- *Step 3:* launch the compilation of each configuration set in **LISTE_CONF**, i.e. launch gmake under each *./config/CONFIG_NAME* giving as argument of *Makefile_Conf* the compilation list type fixed in **BUILD_MAKE** variable, i.e. mono and/or mpi and/or omp.

***It allows to launch the compilation & the tests for many configurations with only one command***

***Makefile_Conf:*** from ./config/CONFIG_NAME directory
is the same whatever is the considered configuration,
This Makefile relies on 4 main targets depending on the compilation type set in the **BUILD_MAKE** variable input:

- **memo:** compilation to check the memory size required in “pure” static mode
- **mpi:** mpi compilation
- **omp:** Open-MP compilation
- **mon:** mono-processor compilation

So the **memo**, **mpi**, **omp** and **mon** compilations are executed in this specific order skipping one of them if not mentioned in the **BUILD_MAKE** variable.
For all compilations listed just above, it is the *Makefile_Work* under *./config/CONFIG_NAME/WORK* directory which is launched with the target name as argument.

For a given run type (procedure is the same for the others) e.g. **mpi**, once the compilation ended with *Makefile_work*, the following section (in the Makefile) prepare and launch the **mpi** simulation: in copying executable, shell scripts and jobs from the *./config/NVTK/INSTALL/JOBS* directory in the appropriate directory built at the installation of the NVTK environment (see part II-a Download and Install).

***Makefile_Work:*** from *./config/CONFIG_NAME/WORK* directory
This Makefile allows to compile one configuration in mono-processor, in MPI or in Open-MP depending the target name given in argument, i.e. one of the 4 targets listed above.

Furthermore each compilation is performed in parallel, i.e. in using many UNIX processes (currently fixed to 6). This can be changed through the **NBPRC** variable in *./config/CONFIG_NAME/scripts/BB_make.ldef* file. This compilation speed-up has a memory cost since all objects and modules (*.o & *.mod) files are stored in the *./modipsl/lib/oce_CONFIG_NAME/target* directory and transformation lists are also stored in the *./modipsl/ltmp_CONFIG_NAME/target*.

**C - <u>Compilation options and cpp keys:</u>**

As in the classical modipsl architecture:

- compilation options are fixed in the *./config/CONFIG_NAME/scripts/BB_make* dependencies file*;* it is built each time the *./config/NVTK/fait_AA_make* script is used, i.e. when a new cpp key or module is created. The best way to change compilation options is to do them in the *fait_AA_make* script and execute it from one *./config/CONFIG_NAME* directory using the *../NVTK/fait_AA_make* command.
- cpp keys for one configuration are set in the *./config/CONFIG_NAME/scripts/BB_make.ldef* file.

To allow the *Makefile_Work* to compile whatever the input argument is, i.e. memo, mpi, omp or mon, a section has been written (in *./config/NVTK/fait_AA_make* script) to add a specific compilation option or cpp key depending the input argument. For example with the input **mpi** argument, the key_mpp_mpi cpp key is added to the list of cpp keys associated with the configuration compiled.

## II-c Launching jobs and assessment report

This phase is automatically executed when the compilation ended as seen in the previous section *II-b Compilation process*. The 3 run types, i.e. mon, mpi and omp, allow to check the reproductibility whatever is the number of processors used; for each of them the restartability is checked and requires 3 sub-runs stored under *./NEMO_VALID/WCONFIG_NAME/* LONG, 1_SHORT (stream 1) and 2_SHORT (stream 2) directories described in previous section. In this part are addressed how jobs are built and launched through ***A) Jobs preparation and launching*** followed by the ***B) Assessment report***.

**A - <u>Jobs preparation and launching:</u>**

This step relies on 3 shell scripts described below and stored in *./config/NVTK/INSTALL/JOBS* directory:

- ***lance_batch.ksh:*** build jobs and launch them
- ***job_CONFIG_NAME.ksh:*** manage input files, launch the execution and save output files
- ***jhd_targetname_runtype:*** header associated to a platform to submit batch jobs

For a configuration CONFIG_NAME, when a run type compilation (i.e. mon, mpi or omp) ended, these 3 scripts are copied under the *NEMO_VALID/WCONFIG_NAME/runtype* directory. The location of the *NEMO_VALID* directory has been specified through the **OUTDIR** variable in the *ins_nvtk.ksh* script by user at the NVTK installation, see II-a Download and install.

***lance_batch.ksh:*** script to build and launch jobs

this script is executed (after the compilation) under the *NEMO_VALID/WCONFIG_NAME/runtype*, with 4 input arguments, for one configuration and for each run type (mon, mpi or omp): > ./ *lance_batch.ksh* "*CONFIG_NAME*" "*timing/notiming*" "nojobs/all/long/short/gtime" "*reference_tag*".
Basically *lance_batch.ksh* modifies (see below) both *job_CONFIG_NAME.ksh* and *jhd_targetname_runtype* files and concatenate them to give 4 jobs which are successively launched;

- *job_runtype_long.ksh*: long run
- *job_runtype_1_short.ksh*: short run (stream 1)
- *job_runtype_2_short.ksh*: short run (stream 2) using restart of the stream 1
- *job_runtype_gtime.ksh*: one year run (built only if the "*timing*" argument is specified

Note that the job job_runtype_2_short.ksh is launched only when the stream 1 is finished.
in *job_CONFIG_NAME.ksh:* some lines associated to a target are selected and variables are filled
in *jhd_targetname_runtype:* number of processors to use and the job name are adapted

The last step in *lance_batch.ksh* is the execution of the *cron_jobs.ksh*. Its goal is to launch the *assessment.ksh* script as soon as all runs associated to a configuration are finished i.e. mon mpi and omp. See section *B- Assessment report*.

***job_CONFIG_NAME.ksh:*** script to manage the pre-run, the run itself, and the post-run phase

It is used to build each of the 4 jobs described just above. It contains all information to:

- get input files: namelist, forcings, executable, restarts if needed
- launch the execution of the model
- save output files under *./NEMO_VALID/WCONFIG_NAME/runtype/* directories LONG, 1_SHORT (stream 1), 2_SHORT (stream 2)

- launch the restart run (stream 2) if necessary
- retrieve reference tag solver.stat and memory_size.txt files (4th argument if specified when launching *lance_batch.ksh* script)

Some instructions are hard-coded in this script to change some namelist variables. Modifications are put there to ensure, for one configuration, identical simulations by time, i.e. same iterations, advection scheme used ..etc

***jhd_targetname_runtype:*** header associated to one platform

it corresponds to the header of a job to enable batch job on the target name and for a given run type. It gather at least the required CPU time, the memory size, and the job name. This file must be provided by user for each new computer and for each run type since instructions of the header might be different for each of them.

## B - <u>Assessment report:</u>

The aim of the NVTK environment is to check that no modifications occur in the results of standard configurations when making changes or using new modules. To check that in a simple way, an e-mail is sent to the user with conclusions on all runs performed.

The assessment report is realised with the *cron_jobs.ksh* and *assessment.ksh* scripts which are copied under the *./NEMO_VALID/WCONFIG_NAME* directory at the compilation end by the *./config/CONFIG_NAME/Makefile* file. A short description of these 2 scripts is given below:

***cron_jobs.ksh:***

checks that no more jobs associated to a given configuration is still in the batch queue. It requires the CONFIG_NAME as input argument.

2 cases:

- if jobs still exist, it sleeps few minutes before relaunch itself,
- if no more jobs exist for a configuration, it launchs the *assessment.ksh* script from the *./NEMO_VALID/WCONFIG_NAME* directory

**Note:** *cron_jobs.ksh* script is launched **ONLY** <u>if the mon</u> run type is executed; see *lance_batch.ksh* script.

***assessement.ksh:***

do a final report (from the *./NEMO_VALID/WCONFIG_NAME* directory) which is sent to the user when all jobs (run type mon, mpi, omp) for a configuration are finished; it requires the CONFIG_NAME as input argument and use information specified in the specifs.txt file. All necessary user's modifications are done in this script at the installation step of the NVTK environment through the *ins_nvtk.ksh* script (see *II-b Download and install* ). It checks the following features (see part *I-NEMO Validation tests procedure*):

- memory size
- timing (if specified)
- reproductibility: mon .vs. mpi and mon .vs. omp
- restartability: mon, mpi and omp  run type
- furthermore if a reference tag is specified in the *./config/NVTK/Makefile*, results for one configuration and one run type obtained with tested files are compared to those resulting from this reference tag to check differences. This check is also based on solver.stat and ocean.output files comparison.

An assessment report example for ORCA2_LIM (on NEC SX8 platform) is given on the next page. The current version tested is the reference tag ***nemo_v3_beta*** with memory and CPU time compared to the ***nemo_v2_3*** one, the following figure shows results concerning the reproductibility and restartability. Note that files used for the comparison are listed with their respective creation date and also the total number of iterations performed for each run to ensure respectively that they correspond to the latest ones and that the run reached the end correctly.

***CPU_time_targetname.ksh:***

this script is charged to format all timing outputs in a readable format; it is called by the *assessment.ksh* script if necessary. Since timing outputs have not a universal output format,  this script must be provided for each new target.

```
#############################
CURRENT VERSION: nemo_v3_beta
#############################


#############################
CHECK EXECUTABLE MEMORY SIZE
#############################

 List memory_size.txt files and check date creation:
 -rw-------  1 reee831 257 Jun 27 17:41 mon/LONG/memory_size.txt
 -r-xr-x---  1 reee831 257 Jun 27 17:41 mon/LONG/REF/memory_size.txt
 -rw-------  1 reee831 19437 Jun 27 17:41 mon/LONG/monORCA2_LIM_err
 -r-xr-x---  1 reee831 17545 Jun 27 17:41 mon/LONG/REF/monORCA2_LIM_err

                              Version
         Memory         Current /      \ nemo_v2_3       Variation
       Stack     (Mo) :        576.1       |       587.0       |      -1.8 %
       Static    (Mo) :      1737.1       |     1751.8       |      -0.8 %
       Max. used (Mo) :    976.0       |       928.0       |        5.2 %


#######################
CHECK CPU TIME USED
#######################


  ---> Timing for the mon run :
       ----------------------------

       List monORCA2_LIM_err files and check date creation:
       -rw-------  1 reee831 19472 Jun 27 18:05 mon/GTIME/monORCA2_LIM_err
       -r-xr-x---  1 reee831 18233 Jun 27 18:05 mon/GTIME/REF/monORCA2_LIM_err

                         Current version    nemo_v2_3 version    Variation
       Real Time (sec)        :     1357.29     |    1264.29   |      7.36 %
       User Time (sec)        :    1146.45      |   1165.05    |     -1.60 %
       MFLOPS                 :    6512.08      |   6183.92    |      5.31 %
       VLEN                   :      224.73     |     220.00   |      2.15 %
       Memory Size (MB)   :      976.03     |      928.03   |      5.17 %

  ---> Timing for the mpi run :
       ----------------------------

       List mpiORCA2_LIM_err files and check date creation:
       -rw-------  1 reee831 37053 Jun 27 15:24 mpi/GTIME/mpiORCA2_LIM_err
       -r-xr-x---  1 reee831 31573 Jun 27 15:24 mpi/GTIME/REF/mpiORCA2_LIM_err

       Global Data of 8 processes  :  Current version   nemo_v2_3 version      Variation
       ==========================
       Real   Time (sec)          :       509.12    |       544.67    |       -6.53 %
       User   Time (sec)          :       440.28    |       469.32    |       -6.19 %
       MFLOPS                     :    2181.89    |     1976.04    |      10.42 %
       Average Vector Length   :      161.43    |       150.30    |        7.40 %
       Memory size used (MB) :      451.79    |       406.12    |      11.25 %

       Overall Data:               Current version   nemo_v2_3 version       Variation
       =============
       Real   Time (sec)                 :       509.29     |       544.68    |      -6.50 %
       User   Time (sec)                 :     3522.26     |     3754.57    |      -6.19 %
       GFLOPS (rel. to User Time)  :         17.45     |          15.81    |      10.42 %
       Memory size used (GB)         :           3.53     |            3.17    |      11.25 %
```

```
######################
 CHECK REPRODUCTIBILITY
 ######################


  ---> Reproductibility mon .vs. mpi ? :
       -------------------------------------
       List solver.stat files and check date creation:                Num. time steps:   done  /  expected
       -rw-------  1 reee831 15960 Jun 27 17:41 mon/LONG/solver.stat               0210  /     0210
       -rw-------  1 reee831 15960 Jun 27 15:13 mpi/LONG/solver.stat               0210  /     0210

       YES YES YES YES YES YES YES YES for the current version

 ####################
 CHECK RESTARTABILITY
 ####################


  ---> mon restartability ? :
       -------------------------
       List solver.stat files and check date creation:                Num. time steps:   done  /  expected
       -rw-------  1 reee831 7980 Jun 27 17:42 mon/1_SHORT/solver.stat            0105  /    0105
       -rw-------  1 reee831 7980 Jun 27 18:25 mon/2_SHORT/solver.stat            0210  /    0210

       YES YES YES YES YES YES YES YES for the current version i.e. LONG stream = ( 1_SHORT + 2_SHORT ) streams


  ---> mpi restartability ? :
       -------------------------
       List solver.stat files and check date creation:                Num. time steps:   done  /  expected
       -rw-------  1 reee831 7980 Jun 27 15:14 mpi/1_SHORT/solver.stat            0105  /    0105
       -rw-------  1 reee831 7980 Jun 27 15:25 mpi/2_SHORT/solver.stat            0210  /    0210

       YES YES YES YES YES YES YES YES for the current version i.e. LONG stream = ( 1_SHORT + 2_SHORT ) streams
```

## II-d Add a new target ?

The whole NVTK environment is currently used on 2 platforms: IBM-SP4 (aix, ax_mono) and NEC-SX8 (sx8brodie) from the IDRIS French center. If the user's target is not one of them, he must follow instructions below to be able to add its specifications. It concerns only the NVTK environment installation and use under the modipsl architecture. All files to be adapted or added must be saved under the *./config/NVTK/INSTALL/JOBS* directory. We list hereafter ***A) Scripts to adapt*** and ***B) Scripts to add***.
**Note:**
#1 the modification of scripts, listed below, for a new platform must be done ***before*** the NVTK installation
#2 we suppose here that specifications associated to the new target are already available in scripts associated to modipsl such as compiler name, compilation options ..etc (typically ./modipsl/util/*AA_make.gdef*, ./modeipsl/util/*w_i_h* scripts)

**A - <u>Scripts to adapt:</u>**

At least 5 scripts must be adapted to the new platform:

**lance_batch.ksh:** located under ./NVTK/INSTALL/JOBS directory
this script is executed at the end of the compilation process to build and launch jobs for each configuration CONFIG_NAME and run_type.
- the following variables must be added for each new target, they are used in final jobs:

```
# (1)  - W_XX    = name of target; e.g.  "#-T- sx8brodie" or "#-T- aix" ..
#      - LAUN    = name of jobs launcher command; e.g. qsub, ..
#      - LLJOBS  = name of jobs listing command; e.g. qstat, Qstat ..
#      - LSUB    = name of running command; e.g. "mpirun -np ${PRC}"
#      - LPERF   = name of timing command; e.g. "hpmcount -o perfs_mon.txt"
#      - LJTIM   = required time per CPU for a one year job
#      - LJTIMJ  = required time per job for a one year job
#      - LPERF   = name of timing command; e.g. "hpmcount -o perfs_mon.txt"
#      - CMDGET = name of specific command to retrieve files; e.g. mfget, cp
#      - SYMBOL = identifier associated to a platform which has been used to name all outputs
#                 files of a reference tag
#                 Ex: For Brodie, this symbol could be "B_" so output file names looks like
#                 B_solver.stat, B_ocean.output ...
```

*(1) the W_XX variable is used to select lines associated to a target in using sed UNIX command applied to the job_CONFIG_NAME.ksh script*
- a section must be added to change following items in the *jhd_targetname_runtype* header:
  - the name of the job,
  - the total number of processors to be used in mpi or omp
  - the CPU time for long run (gtime)

**job_CONFIGNAME.ksh:** located under ./NVTK/INSTALL/JOBS directory
In these jobs, additional lines associated to the new target must be added (for each run type) to:
- get and store the memory size of the executable (using the UNIX command "size") in *memory_size.txt* file. This is done only for mon run type.
- export variables for global profiling when they exist, e.g. "*F_PROGINF=detail*" on NEC

These lines must begin with the **W_XX** variable filled in *lance_batch.ksh* script; it will allow to select appropriate lines depending the target name. An example is given hereafter:

```
#-T- sx8brodie export F_PROGINF=detail
#-T- sx8brodie echo 'Stack memory:'          >> memory_size.txt
#-T- sx8brodie size opa_${CONF}_${RUN} >> memory_size.txt
#-T- sx8brodie echo 'Static memory:'         >> memory_size.txt
#-T- sx8brodie size opa_${CONF}_memo    >> memory_size.txt ;;
```

**fait_AA_make:** located under ./NVTK directory
In this script, which is used to build dependencies files, platform's compilation options are set

**AA_make:** located under ./NVTK/INSTALL/CONFIG_FILES directory
In this script, which is used to build dependencies files, platform's compilation options are set

**BB_make.ldef_CONFIGNAME:** located under ./NVTK/INSTALL/CONFIG_FILES directory
In this script, where cpp keys are set, add the prefix used to pass cpp keys to the compiler, add a line which looks like:

**#-Q- sx8brodie** prefix = -D

**B - Scripts to add:**

Only 4 scripts must be added:

***jhd_targetname_runtype*:** located under ./NVTK/INSTALL/JOBS directory
In addition specific headers linked to the new computer and declined for each run type must be created. They are used to build final batch jobs and gather at least the required CPU time, job time and memory. These headers are modified by the *lance_batch.ksh* script to give an adequate job name, and total number of processors to be used in mpi or omp run.

***CPU_time_targetname.ksh*:** located under ./NVTK/INSTALL/JOBS directory
Since profiling outputs are not in a standard format a specific script (based on awk commands) must be created with the syntax file name to treat and reformat basic information as CPU time, Elapse time, maximum memory used

***MEM_size_targetname.ksh*:** located under ./NVTK/INSTALL/JOBS directory
Since memory outputs requirements are not in a standard format a specific script (based on awk commands) must be created with the syntax file name to treat and reformat basic memory information

***BIO_targetname.ksh*:** located under ./NVTK/INSTALL/JOBS directory
Only used for the GYRE_LOBSTER configuration, this script (based on awk commands) output tracers statistics from the ocean.output file and save them into the tracer.stat file before it is used to check restartability and/or reproductibility  must be created in taking into account of number of tracers.

# III – Quick starting guide

**Context:**
User made modifications in few modules of the NEMO code; he wants to check if these changes fit basic features such as the reproductibility, the restartability, or to check the global memory and/or the time consuming but also if results of the reference configurations ORCA2_LIM, GYRE and GYRE_LOBSTER remain unchanged.
The NEMO Validation Tools Kit (NVTK hereinafter) has been built to allow the user to perform a set of validation tests simply and quickly in using few commands.

Here are summarised basic steps the user has to go through:

a. **Download and install the NVTK environment** (*to be done once*)
b. **Compilation, job submission and assessment report**

The user has to refer to previous sections to get details about the general organisation and scripts. Here we suppose the platform used is either the NEC-SX8 either the IBM-SP4 from the French computing center IDRIS, else the user will have to adapt some scripts (following instructions in part *II-d Add a new target ?*) once the NVTK environment will be downloaded.

## III-a Download and install the NVTK environment

**A - <u>Download NVTK:</u>**

ii) In one step, user can download: modipsl architecture, the latest NEMO code (SVN trunk), the reference configurations and NVTK under the TEST directory.

```
> mkdir TEST
> cd TEST
> svn_ano                  (download the modipsl structure)
> cd modipsl/util
> ./model NEMO_DEV         (download the NEMO code)
```

**Note:** *svn_ano* is an alias' associated to modipsl structure download. It refers to the following command: 'svn co http://forge.ipsl.jussieu.fr/igcmg/svn/modipsl/trunk modipsl '

**B - <u>Install NVTK:</u>**

User has to fill following variables in the *./modipsl/config/NVTK/INSTALL/ins_nvtk.ksh* script header:

```
############################################################
##### Begin Users modifications
############################################################
# OUTDIR   : working directory from which jobs will be launched & results stored
# INPUTD   : directory where to get ORCA2_LIM_nemo_v2.tar
# DECOMP   : total number of processors which will be used
# MAIL     : give your e-mail
# UAGRIF   : one of the standard configurations is based on AGRIF yes/no
############################################################
OUTDIR=/workdir/rech/eee/reee831
INPUTD=/u/rech/eee/reee831/IO_NEMO_ORCA2_LIM/
DECOMP=8
MAIL="your_email@xxxxxx"
UAGRIF=yes
############################################################
##### End   Users modifications
############################################################
```

**Note**:
- the **OUTDIR** & **INPUTD** must be directories accessible through classical UNIX *cd, cp ...etc* commands.
- the total number of processors DECOMP could be modified later directly in scripts
- the **MAIL** variable corresponds to the e-mail address to which assessment report will be sent
- **UAGRIF** to compile/or not the AGRIF package before using it.

i) Once the ins_nvtk.ksh header is changed, go under NVTK/INSTALL directory and execute the script.

```
> cd  NVTK/INSTALL
> ./ins_nvtk.ksh
```

ii) when ins_nvtk.ksh stop, set the appropriate domain decomposition values for jpni, jpnj & jpnij parameters. To be done in par_oce.F90_keep file under each ./config/CONFIG_NAME/MY_SRC directory

```
> cd  ./config/CONFIG_NAME/MY_SRC/
> vi par_oce.F90_keep

 INTEGER, PUBLIC, PARAMETER ::    &  !:
   jpni  = 2,           &  !: number of processors following i
   jpnj  = 4,           &  !: number of processors following j
   jpnij  = 8              !: nb of local domain = nb of processors
   !                       !  ( <= jpni x jpnj )
```

**Note:**
- The total number of processors **jpnij** must be the same for all reference configurations and in coherence with the **DECOMP** variable fixed in the *ins_nvtk.ksh* script just above.
- To change the number of processors to be used during runs, change in the *./config/NVTK/INSTALL/JOBS/lance_batch.ksh* the **PRC** variable before launching the compilation process.

## III-b Compilation, job submission and assessment report

This could be performed in using just one command as described in part A or manage each step separately "by hand" in part B. A schematic overview of the whole process can be found in page 18.

**A - *Launching the whole process : compilation, jobs and final report***

i) store files under *./modipsl/config/NVTK/2TEST*
ii) set following 7 variables in the *./modipsl/config/NVTK/Makefile* file:
- **NAM_V:** name of the test (this information will be used in the assessment report at the end of all simulations)
- **LISTE_CONF:** name of standard configurations to test with modified routines under 2TEST directory
- **JOBS_2LAUN:** select jobs to launch
- **BUILD_MAKE:** compilation type (run type) to perform: mono-processor and/or multi-processors MPI and/or Open-MP (this list is applied to each configuration specified in the LISTE_CONF variable). *If **mpi** is specified, the user must set the processor cutting (jpni, jpnj & jpnij parameters) in the par_oce.F90_keep module stored in each ./config/CONFIG_NAME/MY_SRC directory. This MPI decomposition must be the same for the 3 configurations*
- **MAK_TIME:** perform or not a timing check
- **MAK_MEMO:** perform or not a memory check
- **REF_TAGV**: (optional) reference tag version to which compare results

iii) launch the compilation process using *gmake* command from *./modipsl/config/NVTK/2TEST*

```
#- Name of the test
NAM_V = test1
#
#- Configurations to be tested
LISTE_CONF = ORCA2_LIM  GYRE  ZAGRIF
#-
#- Jobs to launch use keyword: nojob, all, long, short or gtime
JOBS_2LAUN = all
#
#- Compilation list type to perform, mon (mono)
#  &/or mpi (MPI) &/or omp (Open-MP)
BUILD_MAKE = mon mpi
#-
#- Proceed to a timing, use key word 'timing' or 'notiming'
MAK_TIME = notiming
#-
#- Proceed to a memory check, use key word 'memo' or 'nomemo'
MAK_MEMO = nomemo
#
#- Reference Tag version (optional)
REF_TAGV = nemo_v2_3
#
```

**Note 1:** *since this environment has been built to use UNIX multi-process possibilities, it is possible to speed-up the compilation in using the gmake option -j : "gmake -j number_of_configurations_to_compile"*
**Note 2:** the **REF_TAGV** variable is usefull only in the case you already performed runs on your own platform for one of the NEMO tag, i.e. nemo_v2 or nemo_v2_3 for instance.

Links are created between *./config/NVTK/2TEST* and *./config/CONFIG_NAME/MY_SRC* for each configuration set in the **LISTE_CONF** variable. During the compilation process, 2 logbooks are created for a given configuration:

- ***NAME_CONFIG_step.txt*** file under *./config/NVTK* directory: list the run types compiled, jobs launched and jobs in the queue
- ***NAME_CONFIG_logbook_runtype.txt*** file under *./CONFIG/CONFIG_NAME* directory: list mainly cpp keys used and all modified files used for the compilation (stored under *./config/NVTK/2TEST*)

## B - ***Launching "by hand" either the compilation either the jobs or the report for one configuration***

i) Compilation:
For a debugging step, it is usefull to launch manually the compilation for a given configuration. Since modified files are stored in *./config/NVTK/2TEST* and links are done between this directory and the *./config/CONFIG_NAME/MY_SRC* one, the compilation is handled under the *./config/CONFIG_NAME/WORK* directory with the command: "***gmake COMP=run_type***", run_type being mon and/or mpi and/or omp.

ii) Jobs launching:
If it is necessary to launch only jobs for one configuration and one run_type, you must be under the directory (*$(OUTDIR)/NEMO_VALID/WCONFIG_NAME/run_type*) you specified in the **OUTDIR** variable in the *ins_nvtk.ksh* script; 2 ways to proceed:

- to launch the 3/4 runs : ***./lance_batch.ksh "CONFIG_NAME" "timing/notiming" "all/short/long/gtime" "REFERENCE_TAG"***. The last optional argument refers to a reference tag for which results already exist and have been saved. If specified, solver.stat, ocean.output, memory_size.txt and runtypeCONFIGNAME_err files will be retrieved under *./NEMO_VALID/WCONFIG_NAME/run_type/REF* directory.
- launch one specific job, e.g: "***qsub job_mpi_1_short.ksh***"

User can also make ***local*** modifications in scripts such as:

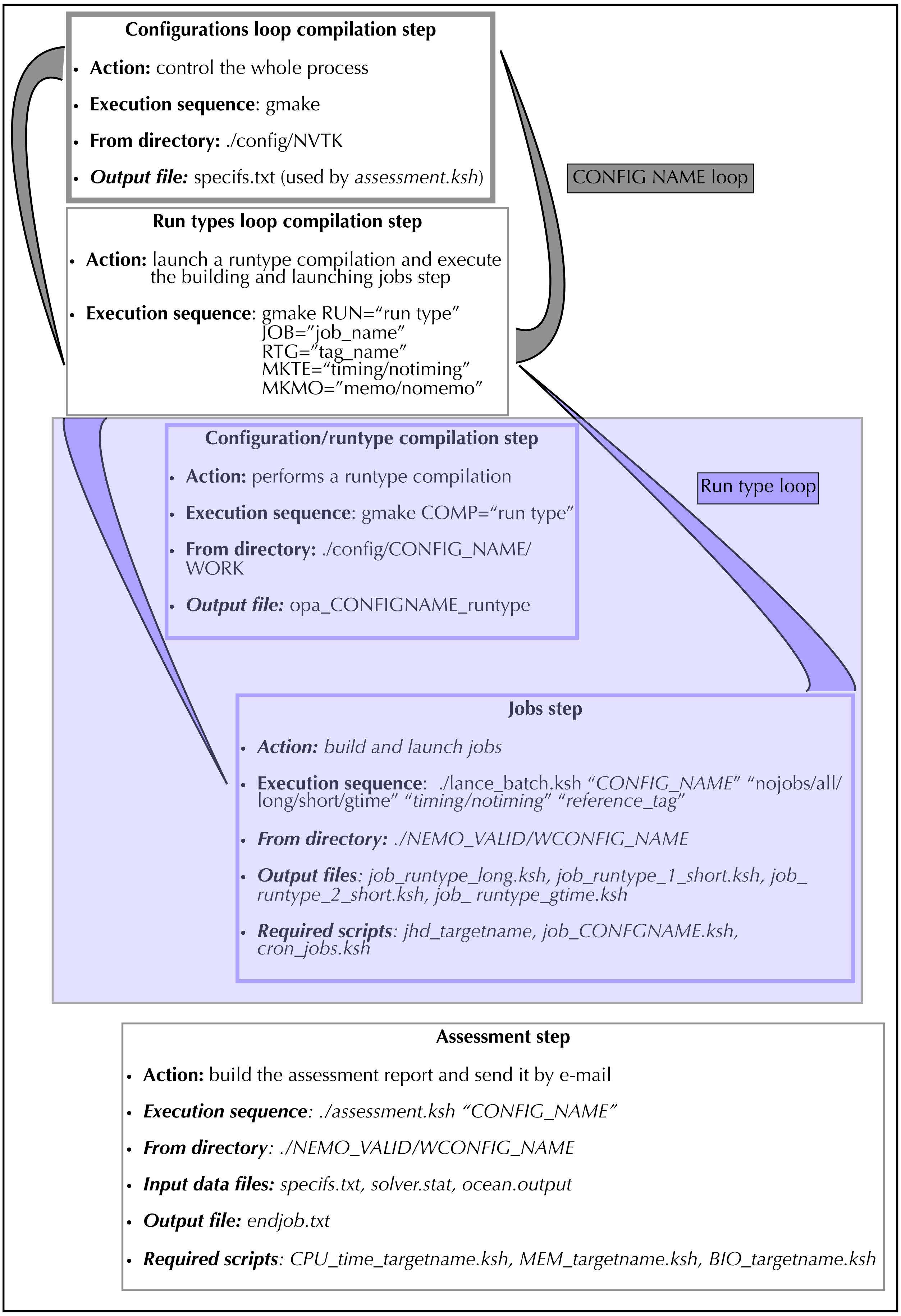
- the header (e.g jhd_sx8brodie_mpi for the SX8 target and mpi run type) about the CPU time, the memory required ..etc
- the main job (e.g. job_ORCA2_LIM.ksh) in adding new files to get/save, changing physics/numerics in namelist

If these changes should be permanent, modified scripts must be saved under the *./config/NVTK/INSTALL/JOBS* directory to be systematically used.

iii) Assessment report:
In case jobs are launched "by hand", i.e. without the *lance_batch.ksh* script as defined just above, no assessment report will be build "automatically". So the user has to check that for a given configuration all jobs ended (mon, mpi or omp) and then execute the following command from the *../NEMO_VALID/WCONFIG_NAME* directory: ***./assessment.ksh "CONFIG_NAME"***

**Note:** The assessment report, for one configuration, is built "automatically" **ONLY** *if the mon run type* is specified in the **BUILD_MAKE** variable in the *./config/NVTK/Makefile* or if it is explicitly launched with the lance_batch.ksh script from the *./NEMO_VALID/WCONFIG_NAME/mon* directory.

Schematic overview of the whole compilation, jobs execution and assessment report process



**Configurations loop compilation step**

- **Action:** control the whole process
- **Execution sequence**: gmake
- **From directory:** ./config/NVTK
- ***Output file:*** specifs.txt (used by *assessment.ksh*)

**Run types loop compilation step**

- **Action:** launch a runtype compilation and execute the building and launching jobs step
- **Execution sequence**: gmake RUN="run type" JOB="job_name" RTG="tag_name" MKTE="timing/notiming" MKMO="memo/nomemo"

**Configuration/runtype compilation step**

- **Action:** performs a runtype compilation
- **Execution sequence**: gmake COMP="run type"
- **From directory:** ./config/CONFIG_NAME/ WORK
- ***Output file:*** opa_CONFIGNAME_runtype

**Jobs step**

- ***Action:*** *build and launch jobs*
- **Execution sequence**: ./lance_batch.ksh "*CONFIG_NAME*" "nojobs/all/ long/short/gtime" "*timing/notiming*" "*reference_tag*"
- ***From directory:*** *./NEMO_VALID/WCONFIG_NAME*
- ***Output files****: job_runtype_long.ksh, job_runtype_1_short.ksh, job_ runtype_2_short.ksh, job_ runtype_gtime.ksh*
- ***Required scripts****: jhd_targetname, job_CONFGNAME.ksh, cron_jobs.ksh*

**Assessment step**

- **Action:** build the assessment report and send it by e-mail
- ***Execution sequence****: ./assessment.ksh "CONFIG_NAME"*
- ***From directory****: ./NEMO_VALID/WCONFIG_NAME*
- ***Input data files:*** *specifs.txt, solver.stat, ocean.output*
- ***Output file:*** *endjob.txt*
- ***Required scripts****: CPU_time_targetname.ksh, MEM_targetname.ksh, BIO_targetname.ksh*