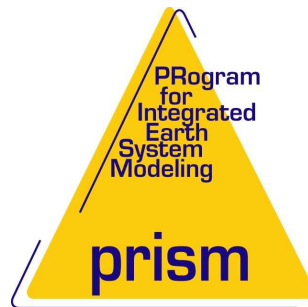


PRISM
An Infrastructure Project for Climate Research in Europe



The PRISM
Standard Running Environment
Handbook

Edited by:
Heinrich Widmann, Stephanie Legutke, Irina Fast, and Veronika Gayler

PRISM-Report Series-05

3. Edition (release prism_2-5)
(last change: 25 February, 2009)

Copyright Notice

© Copyright 2006 by MPI-HH M&D

All rights reserved.

No parts of this document should be either reproduced or commercially used without prior agreement by MPI-HH M&D representatives.

How to get assistance?

Assistance can be obtained as listed below.

PRISM documentation can be downloaded from the WWW PRISM web site under the URL: <<http://prism.enes.org>> or from the SVN PRISM repository.

Phone Numbers and Electronic Mail Addresses

Name	Phone	Affiliation	e-mail
Heinrich Widmann	+49-40-41173-292	MPI-HH, M&D	heiner.widmann@zmaw.de
Stephanie Legutke	+49-40-41173-104	MPI-HH, M&D	stephanie.legutke@zmaw.de
Irina Fast	+49-40-41173-330	MPI-HH, M&D	irina.fast@zmaw.de

Contents

About this handbook	1
Introduction	2
1 The SRE directory structure	5
1.1 The <i>/root/tools</i> directory	5
1.2 The <i>/root/arch</i> directory	6
1.3 The <i>/root/data</i> directory	6
1.4 The <i>/root/util/running</i> directory	6
1.5 The <i>/root/experiments</i> directory	7
2 Experiment configuration and tasks	9
2.1 Create_TASKS.frm and basic settings	9
2.2 The setup	11
2.3 Completion of setup	16
2.4 Generation of the tasks	16
2.5 The suite of tasks	16
2.5.1 Running	18
2.5.2 Pre processing	20
2.5.3 Monitoring	20
2.5.4 Post processing	21
2.5.5 Archiving	22
2.5.6 Data base filling	22
2.6 The functions	22
2.7 The calendar tools	25
3 IT environment and site specific configurations	27
3.1 Nodes and servers	27
3.2 Queueing systems	28
3.3 Remote data management	28
4 Running a coupled model	29
4.1 Step-by-step usage	29
5 Enlarging the System	33
5.1 Adding a new coupled model	33
5.1.1 Adaptation of Create_TASKS.frm	33
5.1.2 Include files depending on the coupled model combination	35
5.1.3 Include files depending on the component model	35
5.1.4 Providing the input data	36
5.1.5 Providing adjunct files	37

5.2	Adaptation of the SRE to a new site	39
5.2.1	Node dependent include files	39
A	Example for a setup file	41
B	Example for a run script	47
C	Examples	71
C.1	COSMOS-ao on SX-6 and the cross platform at DKRZ	71
C.2	COSMOS-asob on the Linux cluster tornado at ZMAW	74
Index		79

List of Figures

1.1	Standardized directory structure	6
1.2	The directory <i>root/util/running</i>	6
1.3	The directory <i>root/experiments</i>	7
2.1	Schematic view on the tasks generation using <i>Create_TASKS.frm</i>	11
2.2	Structure of the setup file	12
2.3	The suite of tasks of an experiment	18
2.4	The structure of a run script	19
2.5	The structure of a monitoring script	20
2.6	The structure of a post processing script	21
2.7	The structure of an archiving script	22
2.8	The structure of a data base filling script	23
3.1	The data flow of an experiment	28
4.1	The workflow of an experiment	31
C.1	Example for a total time series plot as shown on a html page if monitoring is active	72
C.2	Example for the three generated figures of yearly time series over 12 monthly means	73
C.3	By clicking on one figure, the corresponding year is shown	73

List of Tables

1	Definition of the variables used in the handbook	3
2.1	Usage message of <code>Create_TASKS.frm</code>	10
2.2	List of the optional task switches	13
2.3	List of time control settings	14
2.4	List of the Unix commands defined in the setup	16
2.5	Variables defined in setup and <code>complete_setup</code>	17
2.6	Date and time formats supported by calendar tools	25
3.1	Error message if a not supported queueing system is entered	28
5.1	List of by IMDI supported coupled models	34
5.2	Variable names of the model components	34
5.3	Content of the initial data tar file for the coupled model <code>ECHO</code>	37
5.4	Variables used in the <code>namcouple</code> base file	38
5.5	List of by IMDI supported 'sympolic nodes'	39
C.1	Output of the first call of <code>Create_TASKS.frm</code>	74
C.2	Status of experiment	75

About this handbook

The PRISM infrastructure software provides support for the full suite of steps that is needed to perform an experiment with an earth system model. The suite starts with the source code retrieval and ends with the visualization of model diagnostic output. This handbook describes only one aspect of the full system, the SRE (Standard Running Environment), i.e. the configuration and creation of run scripts and experiment execution. For the first part, the SCE (Standard Compile Environment), which deals with retrieval of the necessary source code and tool box from the repository, generation of Makefiles and compile scripts, and compilation, consult the handbook on SCE by [Legutke et al. \(2007\)](#).

This edition describes the software release tagged as indicated on the cover page. The SRE development is an ongoing process as is the writing of this handbook. These activities will be synchronized as much as possible. Therefore, new editions of the handbook will be provided in about the same intervals as new software is released with major modifications of the SRE. If model adaptation activities are started, it should thus be ensured that the newest edition of the handbook and software is used. It can be downloaded from the PRISM web site (<http://prism.enes.org>) or, together with the PRISM software, from the PRISM SVN source code repository (http://svn-prism.zmaw.de/svn/prism/tags/tag/util/running/doc/PRISM_SCE.pdf).

Acknowledgment

The design of the SRE profited much from discussions with colleagues and their input: Martina Schubert and Karin Meier-Fleischer (M&D at the MPI-M).

Introduction

The PRISM project aimed at the establishment of a climate research network in Europe. An important step towards this goal is the development of a common flexible, easy-to-use, and portable infrastructure for earth system modelling.

Keeping in mind the large number of models and platforms used in Europe for climate modelling, and taking into consideration the quick development of both software and hardware, it is obvious that the infrastructure must be extendable to accommodate new models and platforms, and must facilitate the replacement of component models, while still being low in maintenance.

In order to meet the requirements of portability, flexibility and extendibility at low maintenance costs, the SRE software is highly modularized and thereby gives the user a common look&feel for all activities possible within the SRE, independent of the models or platforms. Earth system models, on the other hand, have to meet a minimum of standards in order to allow the use of the SRE tools for model execution and automatic data processing. This makes it possible to have a portable toolbox for performing experiments with coupled models which consist of an arbitrary number of component models on different computing platforms.

The running environment comprises a standard directory structure which is described in detail in Chapter 1. It is closely connected to the source code directory structure of the SCE (Legutke et al., 2007).

The SRE includes a comprehensive set of utilities to configure model experiments and generate standardized tasks, i.e. scripts for model integration and for data management. These tasks are generated from a base of partly platform and/or model dependent include files, which allows to perform experiments with complex model configurations within different and heterogenous IT environments from initialization up to data storage and visualization. This method furthermore enables for easy adaption of new coupled models or new platforms since model and platform/site dependent sections are clearly identified.

Experiment configuration, the tasks and their generation is topic of Chapter 2. In Chapter 3 we show how the experiment workflow and data management can be adapted to given IT environments. Chapter 4 gives step-by-step instructions on how to perform simulations with an existing coupled model. Detailed information on how to extend the system, i.e. how to adapt a new component or coupled model or to include a new platform/site, is provided in Chapter 5.

The latest released version of the SRE can be downloaded either from the central PRISM repository (<http://svn-prism.zmaw.de/svn/prism/>) or from the repository of M&D (<http://svn-mad.zmaw.de/svn/mad/Model/IMDI/>) where the SRE is developed and maintained. Both repositories are based on the Subversion versioning system (SVN)¹. How to download the whole SCE and SRE directory structure or only parts of the environment is described in detail in Legutke et al. (2007).

If you plan to adapt a new coupled model combination to the SRE or to port the system to a new platform/site please do not hesitate to contact us. We are interested in your experiences and are happy to answer your questions. Besides, it is possible to integrate your changes into the central SVN system.

¹For details about the version control system Subversion see <http://subversion.tigris.org/>

Notes on usage of this handbook

Typewriter font is used to indicate file and directory names, Unix commands or standard input of programs. Variable parts of these strings are written in italics. For example, the file `input_model.nc` contains input data for the model with model name *model*. Table 1 gives a list of the variables used in the following chapters.

Variable	Description
<i>model</i>	component model name
<i>cplmod</i>	coupled model name
<i>node</i>	(symbolic) node name of the computing host
<i>res</i>	horizontal resolution of a component model
<i>cmp</i>	abbreviation for the model components: atm, che, srf, oce, ice, bgc
<i>c</i>	one letter abbreviation for the model components: a,c,s,o,i,b

Table 1: Definition of the variables used in the handbook.

Chapter 1

The SRE directory structure

A central feature of the modelling environment is a standardized directory structure. It interfaces the Standard Compile Environment (SCE) and the Standard Running Environment (SRE).

The checkout of a coupled model from the SVN repository provokes the inflation of the standard directory structure with a root directory called as the tagged revision, if no explicit target directory name is used during checkout. The absolute path of this top directory is called `/root` in the following. It is possible to rename this directory, but renaming or moving any other subdirectory of the tree may cause errors.

The root directory contains several sub directories, which are displayed in Figure 1.1. Here, we focus on the directories that are part of SRE. Some of the directories are described in more details in the following sections.

- In `/root/tools` scripts are provided for adaption of new component model, for conversion of FORTRAN source code into HTML format with forward and backward linked subprogramm calls as well as for download of a certain coupled model from a SVN repository (Legutke et al., 2007). Furthermore, several tools for automatic experiment configuration and drivers for running tasks are provided.
- In `/root/arch` the binaries and model executables are stored. The executables can be generated within SCE or provided from another source.
- `/root/data` contains tar-files with initial input data for coupled models.
- `/root/xml` provides tools and templates for generation of XML-formatted experiment meta data files.
- `/root/util/running` includes all utilities, tools, name lists, header files and functions needed for task generation and experiment execution. Furthermore the generated setup files, within the experiment configuration is specified, reside here in the subdirectory `setup`.
- `/root/experiments` contains scripts and directories needed to run the whole experiment workflow or parts of it.

1.1 The `/root/tools` directory

Here are general tools and drivers provided, which allow to trigger specific parts of the tasks, to test the basic functionality, or to check whether e.g. processing and archiving of model output was successful. E.g. the script `check_archived_data` checks, if model output data are stored in a given archive path.

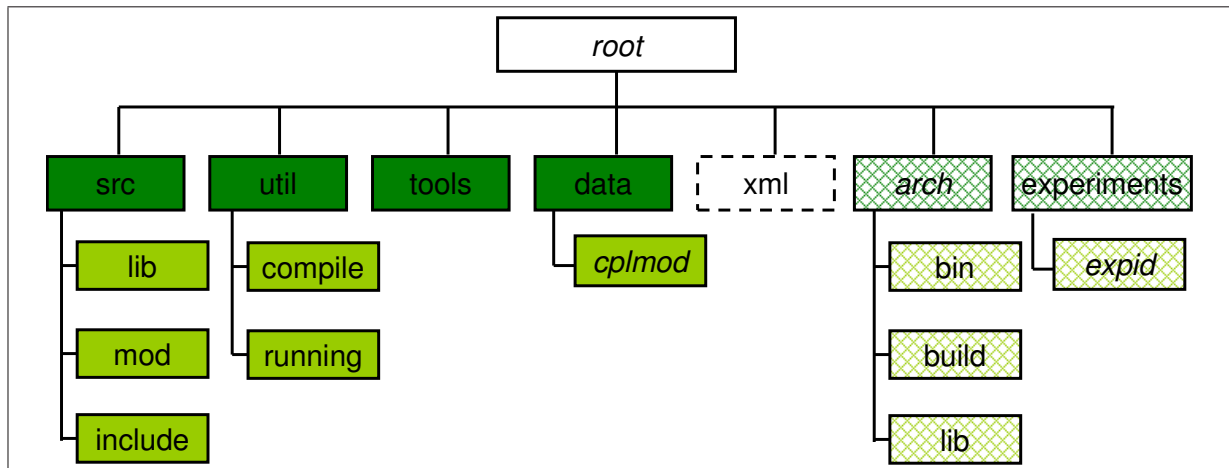


Figure 1.1: Full standardized directory structure. The subdirectory *arch* is created at compile time. The subdirectory *experiments* is created at script creation time.

1.2 The */root/arch* directory

This directory is not available from the SVN repository and is created at the first compilation action. Here reside the binaries and model executables. Model executables generated not within SCE should be put in the directory */root/arch/bin* and named as *model_id[.MPI1,2].x*. For coupled models the coupler is regarded as component too and should be named *coupler.MPI1,2.x*.

1.3 The */root/data* directory

Tar-files with input data for a specific coupled model combination can be stored in */root/data/cplmod*. From the SVN repository tar-files with initial input data are available for TOYCLIM and TOYOA4 only.

1.4 The */root/util/running* directory

The */root/util* directory, also introduced in the SCE handbook (Legutke et al., 2007), contains tools and scripts for compilation (*/root/util/compile*) and for running (*/root/util/running*) of coupled models. We focus here on the running branch, which is shown in Figure 1.2.

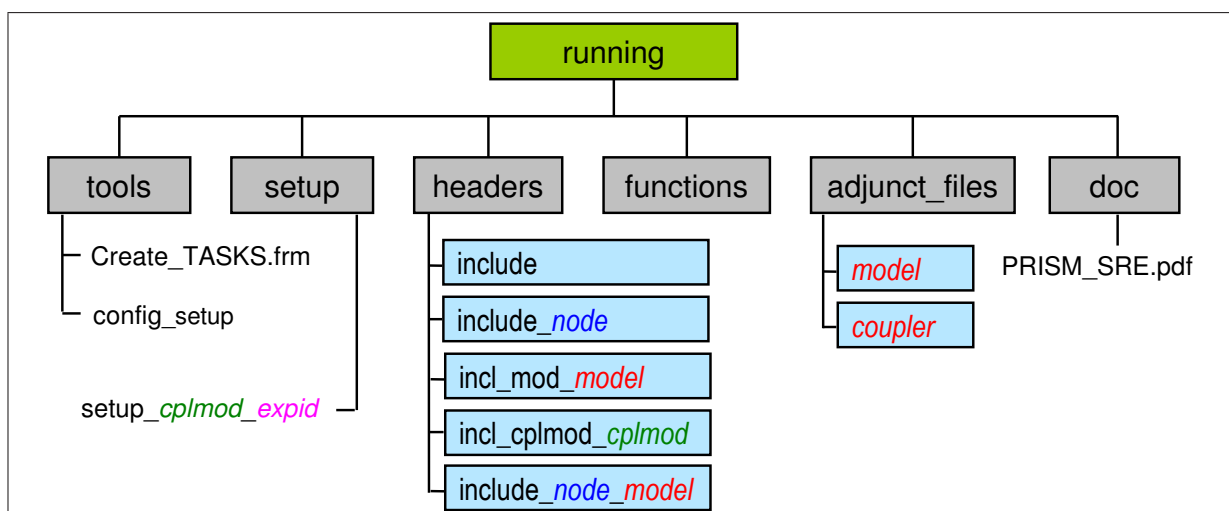


Figure 1.2: The directory */root/util/running*.

The subdirectory `tools` contains a script to generate the tasks called `Create_TASKS.frm`. It uses several include files residing in subdirectories of `headers` and depending on the computing host `node`, the coupled model `cplmod` or the model component `model`.

The setup files of different experiments defined by the user are stored in the subdirectory `setup`. A detailed description of the tasks and their generation is given in Chapter 2.

The subdirectory `functions` contains auxiliary shell functions and scripts called during experiment execution.

Some models need ASCII input files (namelists etc), which for example control the model configuration and integration duration. They are kept in subdirectory `adjunct_files`. Most prominent among the adjunct files is the input file of the coupler OASIS3 called `namcouple`. A special `namcouple` version is distributed with each of the coupled models in the file `adjunct_files/oasis3/namcouple_cplmod`. For more information on `namcouple` please read Section 5.1.5.

The subdirectory `doc` contains documentation on SRE.

1.5 The /root/experiments directory

The `experiments` directory is not available from the SVN repository but is installed when the user defines a new experiment. All user experiments are stored in subdirectories labelled with the experiment ID (`expid`). The directory `/root/experiments/expid` contains all that is needed to run the specific experiment, as shown in Figure 1.3 for an experiment `expid2`.

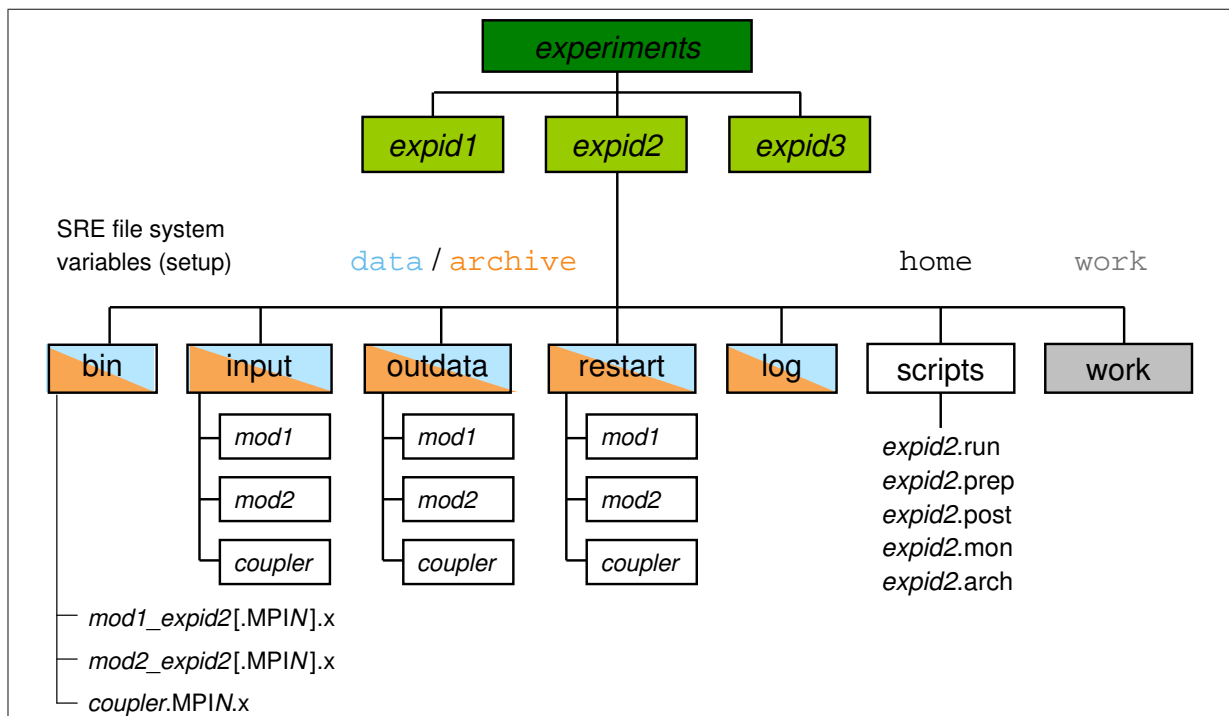


Figure 1.3: The SRE subdirectory `experiments`. As example the unfolded subdirectory tree for an experiment with `expid2` is shown. Variables `home`, `data`, `archive`, and `work` refer to configurable variables of the experiment setup.

As compilation, execution, post processing, monitoring and archiving not always happens on a single machine, the directory structure might span several file systems on several machines. E.g. the variables

data, *home*, and *work* determine paths on the compute node, but e.g. *archive* assign the archive host and path, where model output is archived. All these settings refer to configurable variables of the experiment setup as described in section 2.2.

The executables of the component models and the coupler are placed in */root/experiments/expid/bin*. They are copied from the directory */root/arch/bin* into *experiments/expid/bin* at the beginning of the experiment and stay unchanged on the computing host until the end of the experiment, even if a model was recompiled.

The input data needed for experiment *expid* are stored in */root/experiments/expid/input*. Each of the component models and the coupler have separate input data directories. Analogously, separate directories for the output data (*experiments/expid/outdata/model*) and the restart files (*experiments/expid/restart/model*) of the components are defined. The adjunct files of the component models and of the coupler are copied from the */root/util/running/adjunct_files* directory to the models input directories at the time the tasks are created.

The directory */root/experiments/expid/log* contains the log files created by the model components at run time and the standard output files of the runs.

The scripts (tasks) used to execute the experiment *expid* can be found in */root/experiments/expid/scripts*. Depending on the setup of the experiment there might be scripts for running, preprocessing, post processing, monitoring, archiving, and data base filling. At run time two additional files will be created and placed into this script directory, namely *expid.log* and *expid.date*. Both files are important for the course of the experiment and must not be removed.

The directory */root/experiments/expid/work* is the working directory at run time. At the beginning of each run the content of this directory will be deleted automatically. Then the executables and input files used for the run are moved to here. When the run is over the output is transferred from there to the experiment's output data directories (*/root/experiments/expid/outdata/model*).

Chapter 2

Experiment configuration and tasks

Usually, model experiment execution is partly model and site dependent, even though the main part of the underlying scripts is identical for all platforms. Some parts (as e.g. functions or the calendar) can be used with all model combinations. To minimize the effort of portability and maintenance IMDI distributions do not contain run scripts or other ready-to-use tasks, but a tool to generate the tasks from a collection of short include files.

The usage and syntax of this central script `Create_TASKS.frm` and the basic settings are described in the next section 2.1.

The first call of `Create_TASKS.frm` creates a setup file. The setup file, where you can configure your experiment, is subject of Sections 2.2 and 2.3.

The execution of a model experiment consists of different tasks, one following the other in order of launching. These tasks are created by the second and subsequent call of `Create_TASKS.frm` with the same experiment ID - corresponding to the settings in the setup file. Details about task generation are given in Section 2.4. The features of the different tasks are described in Section 2.5, and the usage and functionality of the functions are explained in Section 2.6.

2.1 `Create_TASKS.frm` and basic settings

The assemblage of include files is performed by a script called `Create_TASKS.frm`. It makes use of the GNU M4 macro processor. The script resides in the directory `/root/util/running/tools`. `Create_TASKS.frm` needs at least two input parameters and the others are in most circumstances optional :

- The name of the coupled model `cplmod` the tasks are created for is mandatory and the only position parameter
- The option `--id | -i EXPID` is mandatory and assigns the given ID to the experiment
- The option `--node | -n NODE` specifies the compute node, for which the tasks will be generated, and is set by default to `$(uname -n)` of the machine where `Create_TASKS.frm` runs on.
- The option `--compiler | -c COMPILER` is only required if there are more than one compiler is available for the specified node
- By `--file | -F FILE` you can choose another setup file instead the by default generated setup file
- With the option `--metadata | -m` given meta data will be written in XML files

By entering `Create_TASKS.frm -h -v` one gets the usage message, as shown in Table 2.1

The first call of `Create_TASKS.frm` for a specific coupled model and experiment ID leads to the generation of a setup file (`/root/util/running/setup/setup_cplmod_exp_id`). This file contains

```

NAME
    Create_TASKS.frm - generate whole set of tasks and running scripts
    for a coupled model run of supported IMDI model combination.

SYNOPSIS
    Create_TASKS.frm [OPTION]... --id, -i EXPID CPLMODEL

DESCRIPTION
    Creates by the first call for the specified CPLMODEL and corresponding EXPID
    a setup file. By subsequent calls tasks and run scripts are generated according
    the corresponding setup file setup_CPLMODEL_EXPID

.
    CPLMODEL
        required command line parameter which must be acronym of a coupled model.
        If the typed acronym is not a valid model, list of all locally available
        coupled models is printed and execution of the script is stopped.

OPTIONS
    --help, -h
        show the built-in help text and exit.
    --verbose, -v
        set verbosity on. In conjunction with --help option
        display this more detailed help text.
    --id, -i EXPID
        (required) use the specified configuration or experiment EXPID to tag the
        generated compile scripts, build directories and executables
        (except for OASIS3/4 and TOYCLIM). The default value is model
        dependent.
    --node, -n NODE
        symbolic node of compute host. This is by default 'uname -n', except
        running the script on cross (node name 'ds?' at DKRZ), where the tasks
        are created by default for the compute host hurrikan.dkrz.de.
        It is required if
        - site and OS specifications should be taken, which are different
          from the specification for the default 'uname -n', i.e. from the platform
          where it is created, or
        - the login host is not explicitly supported by its hostname ('uname -n') as
          compute host. In this case you have to specify the appropriate
          architecture instead.
        If you type a not supported platform or architecture, a list of all supported
        'symbolic node' names will be printed.
    --compiler, -c COMPILER
        acronym (pgi, nag etc.) of the compiler the used model executables are
        compiled with. This is required if more than one available, as for
        architectures as linux86, crayx1, linux-x64, aix or sunos
    --file, -F FILE
        the file FILE in the setup directory is used as setup file
        instead the by default generated setup_CPLMOD_EXPID
    --metadata, -m
        fill xml-formatted metadata files.
        Note that this is only one step in the full suite.
        The option has to be activated in all scripts in order
        to retrieve all metadata known in IMDI.
        Use the verbose option for meta data generation control messages.
        Note: the meta data generation is not yet active. No use to use it.

```

Table 2.1: Usage of Create_TASKS.frm. The text was produced by typing “./Create_TASKS.frm -h -v” in the directory /root/util/running/tools.

a list of all configurable variables for the selected coupled model. It needs to be edited according to the experimental design. Comments give a short description of the variables and an overview of the choices. More information on the setup file is given in the next Section 2.2.

To generate the tasks `Create_TASKS.frm` needs to be called a second time with the same parameters. A check of the setup is performed. If the selections are not consistent, variables are missing or unknown the setup check will fail. In this case the user needs to correct the setup and run `Create_TASKS.frm` again. When the setup check is passed successfully the tasks are created using the experiment setup file. The tasks are transferred to the computing host in the permanent script directory `$home/experiments/expid/scripts`. At the same time the adjunct files needed for the run are transferred to the component models input directories on the computing host. Figure 2.1 gives a graphical overview of the usage of `Create_TASKS.frm`.

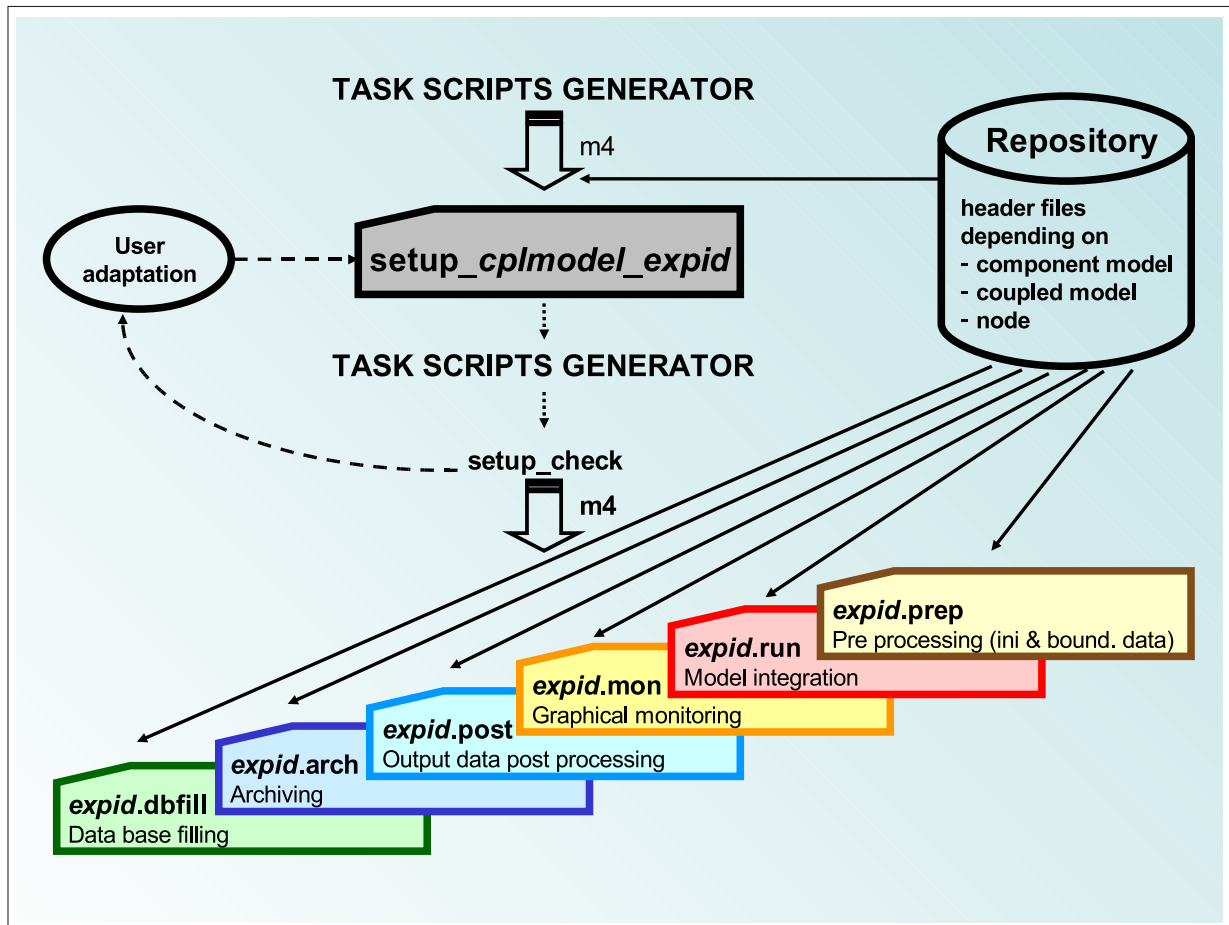


Figure 2.1: Schematic view on the tasks generation using `Create_TASKS.frm`.

2.2 The setup

The setup of an experiment contains the users choices of all configurable variables of a coupled model. It gives a precise description of the experiment configuration which is also valuable for documentation of the experiment. The setup is created by `Create_TASKS.frm` (see above) and is used for task generation.

This section gives an overview of the structure of the setup file and explains the variables that need to be defined. As the tasks the setup itself is made up from several include files depending on the coupled model, the component models or the node, as shown in figure 2.2.

An example for a setup file can be found in the Appendix A.

	Section	Include file name
<i>user interface</i>	0.1 Experiment settings	config_experiment.h
	0.2 Model components	config_components_ <i>cplmod</i> .h
	1.1 Coupled model	config_cplmod_ <i>cplmod</i> .h
	1.2 Tasks specification	config_tasks_ <i>cplmod</i> .h
	1.3 Time control	config_timecontrol.h config_timecontrol_ <i>cplmod</i> .h
	1.4 Initialization	config_initial.h
	1.5 Message passing	config_mpi_ <i>node</i> .h
	1.6 File systems	config_file_systems.h
	1.7 Restart control	config_restart.h config_restart_ <i>cplmod</i> .h
	1.8 Platform specific specif.	config_site.h config_site_ <i>node</i> .h
	1.9 Unix commands	config_commands.h config_commands_ <i>node</i> .h
	1.10 Remote processing	config_remote.h
	1.11 Pre Processing	config_preprocessing[_ <i>model</i>].h
	1.12 Post Processing	config_postprocessing[_ <i>model</i>].h
	1.13 Monitoring	config_monitoring[_ <i>model</i>].h
1.14 Archiving	config_archive.h	
1.15 Data base filling	config_dbfill[_ <i>model</i>].h	
2. Complete setup	complete_setup_ <i>cplmod</i> .h	

Figure 2.2: Structure of the setup. The file is composed of several header files, some of them depending on the coupled model combination (*cplmod*), the component model (*model*), or the platform/site (*node*). The framed part of the setup can be modified by user to configure an experiment.

0. BASIC SETTINGS

How the basic settings as `expid`, `cplmod` and `node` are set by `Create_TASKS.frm` is discussed already in the last section 2.1. They should not be changed in the setup file.

1.1 COUPLED MODEL SETTINGS

The include file `config_cplmod_cplmod.h` builds the first section of the setup file, which can be modified by the user. This section lists all variables defining the component models of the selected coupled model combination. Only variables that give a choice are mentioned. For COSMOS-AO (ECHAM5 + MPIOM) for example the horizontal and vertical model resolutions need to be specified, whereas the toy model resolutions of TOYCLIM are fixed and do not appear in the user interface.

There are some specifications needed for the coupler with all model combinations, as the three character job ID (`jobname`) or the standard output extent (`nlogprt`). Other variables (e.g. `gridswr` stating whether or not new grid description files should be generated at run time) depend on the coupled models flexibility.

1.2 TASK SPECIFICATION

In this part the generation and running of optional tasks (preprocessing, postprocessing, monitoring, archiving, dbfill) can be switched on or off as shown in the table 2.2. Detailed description of the tasks is found in 2.5.

Task switch	Default setting	Description and dependance
<code>preprocessing</code>	yes for CLM	This task is only available for the regional models as CCLM and is called within the run script before model execution.
<code>monitoring</code>	no	Monitoring is submitted direct after model execution is finished and/or the postprocessed data to plot are available.
<code>postprocessing</code>	yes	Post processing is as well submitted from the run script after model execution and is highly model component dependant.
<code>archiving</code>	no	Archiving can only executed if post processing is performed and is submitted at the end of the task POST
<code>dbfill</code>	no	Database filling can only executed if post processing is finished, but in parallel with archiving

Table 2.2: List of the optional task switches. They can be switched on and off by setting to yes or no

1.3 TIME CONTROL

The section time control is identical for all coupled model combinations and all nodes.

The calendar type `caltype` used by the component models and by the coupler needs to be the same. I.e. the corresponding choices are given by the OASIS3 coupler.

An experiment spans a long (simulated) time period and is divided into several runs (or chunks) of equal length. The experiment length is given by `initial_date` and `final_date`.

The duration of each run (or chunk) can be specified as number of years (`nyear`), months (`nmonth`), days (`nday`), hours (`nhour`), minutes (`nminute`), seconds (`nsecond`) or component model time steps

(`nstep_model`) within a run. Depending on the calendar type not all combinations of these variables are allowed.

The possible choices for the time control variables are listed in the table 2.3.

Variable	Settings	Description
<code>caltype</code>	0	Gregorian calendar without leap year (365 days per year)
	1 (default)	Gregorian calendar with leap years
	n	Equal length of months (e.g. 30 for 30 days months)
<code>initial_date</code>	YYYY-MM-DD	Starting date of the experiment
<code>final_date</code>	YYYY-MM-DD	Ending date of the experiment
<code>nyears</code>	n	Length of one chunk given in years
<code>nmonths</code>	n	Length of one chunk given in months
<code>nyears</code>	n	Length of one chunk given in days
<code>nstep_model</code>	n	Length of one chunk given in component model time steps

Table 2.3: List of time control settings

1.4 INITIAL DATA AND SETTINGS

Initial data for a coupled model run are distributed in a tar-file containing the initial data of all the component models. In case the variable `use_initial_tarfile` is set to `yes` this tar-file will be transferred from the archive, defined by the variable `archive_in`. If the initial data are already available no input data tar-file needs to be used. The variable `tag` allows to specify the version of the initial data tar-file.

1.5 MESSAGE PASSING

The section about the message passing method used for OASIS communication is specific to each platform. So, the defaults are appropriate for the machine you are working on. In this section you define the launching mode of the coupled model: `message_passing_method=MPI2` if coupled model should be spawned by coupler and `MPI1` otherwise and whether buffered MPI send (`bsend`) should be used. Also, the MPI library implementation (e.g. `mpich`, `mpich2`, `openmpi`), paths to `mpiexec`, and the number of processors used for the `mpiexec` (`nprocmpl`) can be specified. On most machines `nprocmpl` is zero, but for example `fujitsu` needs a processor exclusively for the MPI.

1.6 FILE SYSTEMS

The variables defined in the file system section are the roots of the PRISM standard directory tree (1.3). The defaults given here correspond to a setup with only one file system on one machine (a common root for the whole tree). For a different setup the root directories of the following file systems need to be adapted.

home: The home file system defines a permanent file system on the computing host. The tasks are placed there. If `Create_TASKS.frm` is not run on the computing host the variable is used for the transfer of the tasks via ftp. In this case it is not possible to use environment variables (as `$HOME`).

data: The data file system resides on the computing host. It needs to be capacious enough for large data files and should conserve the data at least for some days. It should have a fast connection to the temporary working directory. Input, output, and restart data for the experiment are stored on this file system and will be transferred to the working directory at run time.

archive: The archiving file system is used for archiving of the input and output data of an experiment for a long time. At the end of each run the data will be saved first on the `data` file system and then (if desired) in the `archive`. The archiving file system does not need to reside on the computing host. If it resides on a remote host files will be transferred by ftp or by another user defined method of file transfer (compare section Unix commands).

archive_in: The `archive_in` file system is a file system to store initial data that can be used for several experiments of several users. At the beginning of an experiment it is looked for the input data first in the experiments input directories on the `data` file system. If the data is not available the `archive_in` is checked. The `archive_in` file system must reside on the same host as the `archive` file system (`archiving_host`).

work: The working directory resides on file system `work`. Its time horizon does not need to be longer than a run. At runtime all files needed are transferred to the (temporary) working directory.

compile_server and **compile_path:** The variable `compile_server` gives the node name of the compiling host. The `compile_path` defines the directory on the compiling host where the executables are placed. Both variables are only important for the first run of an experiment. At the beginning of an experiment the executables needed are transferred from the compiling host to the experiments `bin` directory on the `data` file system. This assures that the latest versions of the executables is used.

1.7 RESTART CONTROL

All the components of the selected coupled model combinations can either start from initial conditions (climatology) or from restart files of a previous experiment. There are three variables to be specified per component: `cmp_restart` is needed to specify whether (1) or not (0) to start from restart files. If one of the component models starts from a restart file the filename has to be specified in `cmp_restart_file`. Besides, for some models the age of the restart file is needed (`cmp_age`). This is the number of simulated years since the start from initial conditions. In this case it is not possible to start an experiment from a date other than first of January.

1.8 PLATFORM DEPENDENT SPECIFICATIONS

This section is intended for system specific variables (e.g. queueing system, name of the batch queue, environment variables). They might not be relevant on other platforms. Generally, the header file comes with reasonable defaults.

1.9 UNIX COMMANDS

In this section some Unix commands are defined (see Table 2.4). This allows to specify paths to the commands if versions other than the default need to be used. Besides, it is possible to specify special options for a command that might not be available on all platforms. The defaults in this section are specific for the platform the tasks are created for.

1.10 REMOTE DATA PROCESSING

1.11 PRE PROCESSING

1.12 POST PROCESSING

The setup only contains a post processing section if post processing is needed for output data of one of the experiments component models. The variables depends on component models and external postprocessing tools used.

Command	Description
<code>mkdir</code>	create a new directory, possibly with missing parent directories (-p)
<code>cp</code>	copy, possibly without changing the time stamp (-p)
<code>rm</code>	remove a file or directory
<code>rtp</code>	remote transfer protocol (e.g. ftp, gridftp, ssh), possibly with options
<code>put_archive</code>	special command to put data to band archive (e.g. dsmc)
<code>get_archive</code>	special command to get data from band archive (e.g. dsmc)
<code>gunzip</code>	unzip a file that was zipped using gzip
<code>job_account</code>	receive a job account at the end of the run (not needed on all platforms)
<code>qsub</code>	submit a job in batch mode. To run the job interactively the variable has to be defined empty ("") or set to nohup.
<code>cdo</code>	climate data operator
<code>python</code>	python

Table 2.4: List of the Unix commands defined in the setup. The site specific default values of these variables are defined in `config_commands_node.h`.

1.13 MONITORING

1.13 ARCHIVING

The variable `archiving_host` defines the name of the machine used for archiving. If execution and archiving take place on the same machine this variable can stay empty.

1.15 DATA BASE FILLING

The variable `archiving_host` defines the name of the machine used for archiving. If execution and archiving take place on the same machine this variable can stay empty.

2.3 Completion of setup

The setup file contains all configurable parameters (compare section 2.2). However, depending on the model and on the platform there are parameters that need to be defined without the user having a choice. These parameters are defined in `/root/util/running/headers/include_cplmod_cplmod/complete_setup`. This file also contains derived variables. The `setup` and `complete_setup` together contain the complete list of variables needed for the different tasks (Table 2.5).

2.4 Generation of the tasks

As mentioned above the tasks are assembled from several include files depending on the coupled model, the component model or the node. Others can be used for all models on all sites.

2.5 The suite of tasks

An experiment is composed of one or several consecutive runs. Each run consist of one or more tasks. The tasks currently supported within the SRE are running (i.e. particular integration of the coupled model), preprocessing, monitoring, post processing, archiving, and data base filling. At the beginning of an experiment the run script is submitted. When the model integration of the first run is completed, the

Predefined Variables	
<code>expid</code>	experiment ID (input parameter for <code>Create_TASKS.frm</code>)
<code>cplmod</code>	coupled model name (input parameter for <code>Create_TASKS.frm</code>)
<code>node</code>	node name of the computing host (input parameter for <code>Create_TASKS.frm</code> or <code>\$(uname -n)</code>)
<code>components</code>	component models making up the coupled combination
<code>atmmod</code>	atmosphere model
<code>chemod</code>	atmospheric chemistry model
<code>srfmod</code>	surface model
<code>ocemod</code>	ocean model
<code>icemod</code>	ice model
<code>bgcmod</code>	bio-geo-chemistry model
<code>coupler</code>	coupler
Variables for all Components	
<code>res_cmp</code>	horizontal resolution; grid acronym
<code>vres_cmp</code>	vertical resolution; number of vertical layers
<code>cmpvers</code>	component model version; used in the executable name
<code>ncdt</code>	model time step of the component [sec]
<code>nproccmp</code>	number of MPI processors used by the model
<code>nthreadcmp</code>	number of OpenMP threads used by the model
<code>ncplproccmp</code>	number of mpi processors communicating with the coupler
Variables for the Coupler	
<code>dto2a</code>	Exchange time step from the ocean to the atmosphere [sec]
<code>dta2o</code>	Exchange time step from the atmosphere to the ocean [sec]
<code>dtb2a</code>	Exchange time step from the bio-geo-chemistry to the atmosphere [sec]
Number of Processors	
<code>ntproc</code>	total number of processors
Time Control	
<code>caltype</code>	calendar type
<code>initial_date</code>	initial date of the experiment
<code>final_date</code>	final date of the experiment
<code>nyear</code>	number of years per run (chunk)
<code>nmonth</code>	number of months per run (chunk)
<code>nday</code>	number of days per run (chunk)
<code>nhout</code>	number of hours per run (chunk)
<code>nminute</code>	number of minutes per run (chunk)
<code>nsecond</code>	number of seconds per run (chunk)
<code>nstep_cmp</code>	number of seconds per run (chunk)
Restarting Options	
<code>cmp_restart</code>	flag stating whether or not a component is starting from restart files
<code>cmp_age</code>	number of years since start from initial files for the component
<code>cmp_restart_fileext</code>	restart file name of the component; use extension if more than one restart file is needed
<code>message_passing</code>	message passing method for launching of the coupled model: MPI2 if spawned by <code>coupler</code> , MPI1 otherwise
<code>bsend</code>	flag stating whether or not to use buffered sent with MPI
<code>nprocmpl</code>	number of processors reserved for mpi

Table 2.5: Variables of `setup` and `complete_setup`. Depending on whether or not the user has a choice on a variable it is defined either in `setup_cplmod_expid` or in the `complete_setup_cplmod.h`

run script submits itself again for the integration of the next run. Besides, it submits the next tasks. In the case displayed in Figure 2.3 these are the post processing and monitoring tasks. At the end of the post processing script the archiving task is submitted. In case of errors the archiving script is run again until all files are saved successfully.

Not all tasks are mandatory for a successful experiment. Currently, preprocessing is supported as an extra task for regional models (e.g. CLM, CCLM) only. Postprocessing, monitoring, and database filling are optional. These tasks are highly model component dependent. It is up to user to adapt the code for his needs. Whether or not archiving is required is site dependent. Furthermore, some preconditions should be fulfilled for special tasks.

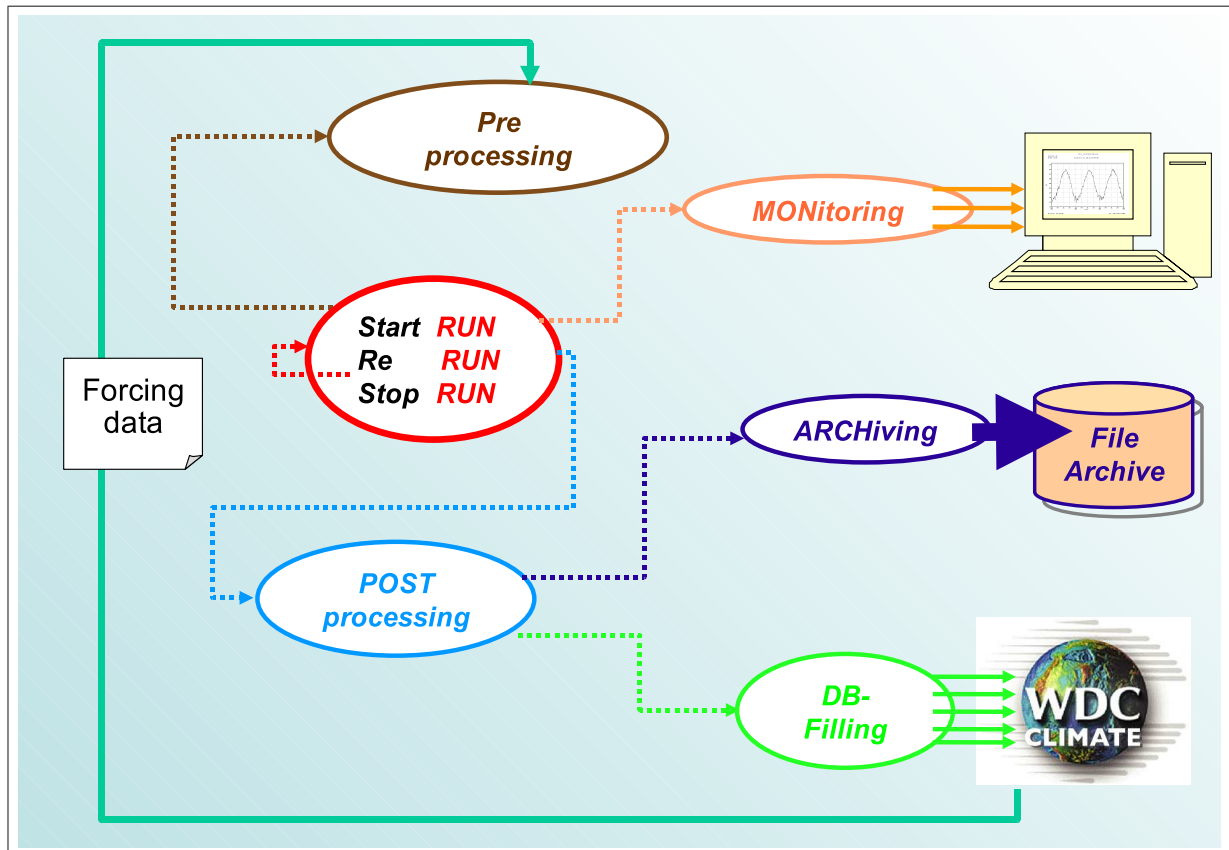


Figure 2.3: The suite of tasks of an experiment. Here the tasks for running, post processing, monitoring and archiving are shown.

2.5.1 Running

The run script (*expid.run*) manages the integration of a coupled model and submits the other tasks as shown in Figures 2.3 and 4.1.

At the beginning of an experiment executables of the model components and coupler are transferred from */root/arch/bin* directory on compile host to the *data* directory on computing host. Also, input and restart data are put in *\$data/input* and *\$data/restart* directories. After the model integration output data and log files are moved from the working directory *\$work* into the *\$data/output* and *\$data/log* directories. Finally, the run scripts resubmits itself for the next run and eventually submits the follow-on tasks. Figure 2.4 shows the structure of the run script. As the other tasks it is composed of several include files depending on either the coupled model, the node or the component model.

The run script starts with introductory comments on the experiment followed by the commands for the queuing system. The next section contains a list of variable definitions. This is described in detail in

Comments	comments_running.h comments_cplmod.h
Job directives	nqs_commands.h
Prologue	prologue.h
Settings and	setup_cplmod_expid.run
Definitions	complete_site_cplmod_node.h complete_setup_cplmod.h job_directory_node.h calendar.h define_functions.h define_directories.h
Preparation	save_logfile_node.h get_tarfiles.h create_workdir.h get_executables_cplmod.h get_input_ini.h
[Preprocessing submission]	[submit_preprocessing.h]
Get input	[get_input_coupler_cplmod.h] get_input_model.h namelist_ini.h [namelist_coupler_cplmod.h] namelist_model.h
Integration	launching_ini.h launching_cplmod_node.h
Output saving	save_output_ini.h [save_output_coupler_cplmod.h] save_output_model.h
Job submission	final_check_saving.h submit_next_job.h [submit_monitoring.h] [submit_postprocessing.h] [submit_archiving.h]

Figure 2.4: The structure of a run script. It consists of several include files, some of them depending on the coupled model (*cplmod*), the platform/site (*node*) or the component (*model*). Include files depending on the model components are listed only once in the figure. However there is one file of the same kind for each component model and for the coupler. Inclusion of files in square brackets depends on coupled models and user-selected optional tasks.

sections 2.2 and 2.3. The run script makes use of several KornShell functions and scripts (see Sections 2.6 and 2.7).

The preprocessing phase starts with the creation of directories. All directories needed for the experiment are generated at run time during the first run. This includes directories on a remote archiving host. The executables of the component models and of the coupler as well as input data are transferred to the working directory. Name lists needed by the component models are generated as here-documents. Execution of the coupled model is triggered by MPI commands in `launching_cplmod_node.h`. Within the post processing phase model output is saved. Finally, the run script submits the next tasks.

2.5.2 Pre processing

2.5.3 Monitoring

The output of an ongoing experiment can be visualized with the optional monitoring task (`expid.mon`). Within this task time series of output variables are generated, plotted, and assembled to HTML-formatted pages, which can be used to monitor the experiment using the web browser. Figure 2.5 shows the structure of the monitoring script.

Comments	<code>comments_monitoring.h</code>
Job directives	<code>comments_cplmod.h</code>
Prologue	<code>nqs_commands_monitoring.h</code>
Settings and	<code>prologue.h</code>
Definitions	<code>time_parameters.h</code>
	<code>setup_cplmod_expid.mon</code>
	<code>complete_setup_cplmod.h</code>
	<code>job_directory_node.h</code>
	<code>define_functions.h</code>
	<code>define_directories.h</code>
[,Remote' Processing commands]	[<code>processing_commands.h</code>]
Preparation	<code>get_tarfiles.h</code>
	<code>create_workdir.h</code>
	<code>get_executables_cplmod.h</code>
	<code>get_input_ini.h</code>
[Preprocessing submission]	[<code>submit_preprocessing.h</code>]
Monitoring	[<code>get_input_coupler_cplmod.h</code>]
	<code>monitoring_model.h</code>
	<code>monitoring_coupler.h</code>
	<code>final_check_monitoring.h</code>
Archiving submission	[<code>submit_archiving.h</code>]

Figure 2.5: The structure of a monitoring script. It consists of several include files, some of them depending on the coupler (`coupler`) or model components (`model`), corresponding to what output fields should be monitored. Include files depending on the model components are again listed only once in the figure. Inclusion of files in square brackets depends on user-selected optional tasks.

The parameter `monitoring` determines whether or not this task is generated. If no graphical monitoring is wanted set (`monitoring=no`) the parameters in the monitoring section of the setup are irrelevant in this case.

The time series are updated after each run period and merged to plot daily, monthly, seasonal and yearly means. This provides on one hand controlling and monitoring the current experiment status, but also a basic analysis of the whole experiment at the end or a zoom on a certain time scale.

To plot the data the graphics package `Madplot` (Meier-Fleischer, 2004) is used to interface `GrADS` and `GMT`. The sources of `madplot` are provided in the functions directory `/root/util/running/functions` or tool directory `/root/tools` respectively. `GrADS` and `GMT` should be installed on the machine, where the monitoring task will be run. The produced images are in GIF format. The python script

`thumbTab4www.py` is used to generate the HTML pages. On these pages the images are organized into tables with thumbnail previews. By clicking on a thumbnail the corresponding large size image is displayed.

Monitoring can be carried out on the compute server or on an extra processing host where the visualization software is installed. In the latter case all output data needed for the monitoring are copied to this host. The monitoring strongly depends on the content of the output files of the component models. The monitoring section of setup allows to specify a list of variables to be displayed. Common to all coupled models are the OASIS3 output files produced by the `mpp-io` library with the options `EXPOUT` or `IGNOUT` (compare Valcke et al., 2004a).

In the output directory of the coupler you can find - at least if the first chunk of your experiment is finished several html files, which can be loaded and reloaded in a web browser. E.g. the figures C.1, C.2 and C.3 show for a concrete experiment the html pages and plots continuously generated during the run. By clicking of one of the figures, there will be pop up the e.g. the chosen year.

2.5.4 Post processing

The postprocessing task is switched on with `postprocessing=yes` in the experiment setup file. This task is highly model dependent. So far, post processing is supported for the COSMOS model family, CLM and CCLM. Figure 2.6 shows the structure of the post processing script.

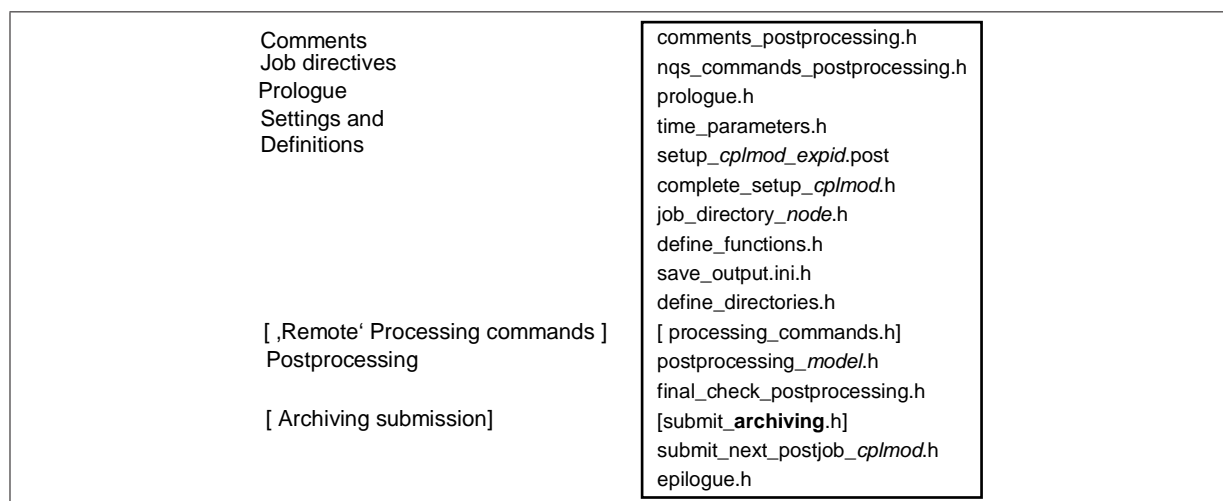


Figure 2.6: The structure of a post processing script. It consists of several include files, some of them depending on the coupler (*coupler*) or model components (*model*), corresponding to which model output should be post processed. Include files depending on the model components are again listed only once in the figure.

The external software used during the postprocessing should be installed on the processing host. These are `afterburner`¹ (only for ECHAM5), `CDO`², and `NCO`³ (for CLM and CCLM only).

Post processing can be performed on the compute host as well as on a remote host. For the latter case the processing host must be set in `postprocessing_host`, and furthermore in `pqsub` the submit command can be specified, because it may differ from the command `qsub`, the run script is submitted with.

¹Standard post processor for ECHAM developed at MPI-M. For more details see <http://www.mpimet.mpg.de/afterburner/>

²Climate Data Operators developed at MPI-M. For more details see <http://www.mpimet.mpg.de/fileadmin/software/cdo/>

³netCDF Operator, see <http://nco.sourceforge.net/>

2.5.5 Archiving

The archiving task (*expid.arch*) is used to save model output in a permanent archiving file system. This file system can reside on the compute server or on a remote archiving host. In the saving phase of the run script the model output is transferred from the working directory to the *data* file system (compare section file systems of section 2.2). This file system should have fast access to the working directory but is not necessarily permanent. The archiving task transfers the data from the *data* file system to the permanent archive. Figure 2.7 shows the structure of the archiving script.

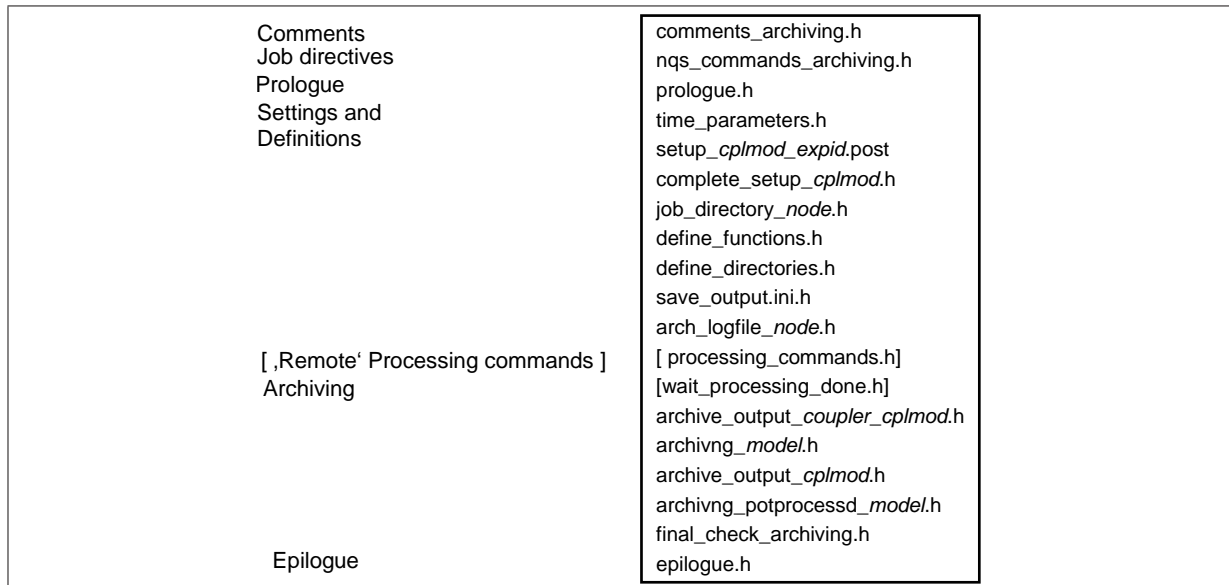


Figure 2.7: The structure of a archiving script. It consists of several include files, some of them depending on the model components (*model*), corresponding to which model output should be filled. Include files depending on the model components are again listed only once in the figure.

Archiving is the final task of the tasks family of a run. After each file transfer the size of the archived file is checked. If one or more files are incomplete or missing the archiving script is resubmitted until all files are archived completely.

The variable *archiving_host* defines the name of the machine used for archiving. If execution and archiving take place on the same machine this variable can stay empty. With the variable *archiving_job* one defines whether archiving is part of the run script or whether a separate job is generated to save the output data on an archiving host. The latter is mandatory if output data post processing or monitoring take place.

2.5.6 Data base filling

The data base filling task (*expid.dbfill*) is currently supported for the COSMOS model family only. To run this task write access to the CERA DB is required. Figure 2.8 shows the structure of the data base filling script.

2.6 The functions

To keep the tasks as clear as possible several KornShell functions and scripts, which are stored in */root/util/running* are used. The functions and scripts are part of the tasks for all models on all platforms. Their functionality is described below.

Comments	comments_dbfill.h
Job directives	nqs_commands_dbfill.h
Prologue	prologue.h
Settings and	time_parameters.h
Definitions	setup_cplmod_expid.dbfill
	complete_setup_cplmod.h
	job_directory_node.h
	define_functions.h
	define_directories.h
	dbfill_.ini.h
	[processing_commands.h]
[,Remote' Processing commands]	[wait_processing_done.h]
Data base filling	dbfill_model.h
	final_check_dbfill.h
Epilogue	epilogue.h

Figure 2.8: The structure of a data base filling script. It consists of several include files, some of them depending on the model components (*model*), corresponding to which model output should be filled. Include files depending on the model components are again listed only once in the figure.

function `get_model_resolution`

This function is called to get the acronym of a grid a component model is running on from the model name. The function does not need an input parameter but uses what is currently defined by the variable `model`. It returns the variable `res`, which is the grid acronym for a component model and the name of the coupled model for the coupler. The function is needed to define the paths to in- and output directories of the archive.

function `get_tarfile`

The initial data for a coupled model run is distributed in a tar-file. For coupled toy models TOYCLIM and TOYOA4 the tar-files are available from the SVN repository in the directory `/root/data/cplmod`. Depending on the user specifications in the `setup` at the beginning of an experiment the initial data can be transferred from the remote or local archiving host. The initial data will be un-zipped and un-tarred in the input data directories of the specific experiment. If the data is already available in the input directories the user can suppress the data transfer by specifying `use_initial_tarfile=no`. As the data transfer, the un-zipping and the un-tarring is time consuming function `get_tarfile` is called only at the first run of an experiment.

function `get_file`

At the beginning of each run all input data including restart files and name lists need to be available in the working directory. Function `get_file` is called to transfer these files. The function first checks for the input file in the `input/model` directories of the experiment on the `data` file system. If the file is not available it checks the archive (`archive_in`) that can be located either on the computing host or on a remote machine (`archiving_host`).

The executables of the component models are transferred to the working directory using function `get_file` as well. If the executables are not available on the `data` file system they are transferred from the compile server.

Optional input files (e.g. re-mapping matrices for OASIS SCRIP interpolation) will be transferred to the working directory if they are available. If not the run will continue without these files.

The function prints a short log message about the file transfer to standard output.

function `save_file`

The function `save_file` is used to save output and restart data at the end of each run. The exact functionality of `save_file` is depending on whether it is called in the run script or in an archiving script. At the end of a run the output data is transferred to the *data* file system on the computing machine. If the archiving file system (*archive*) is different from the *data* file system and if no separate archiving task exists, the data is subsequently saved in the *archive*. This file system can be located on the compute server or on a remote archiving host, which has to be assigned by *archiving_host*. The call of `save_file` in a separate archiving task will cause the file transfer from the *data* to the *archive* file system. When all data is transferred successfully to the *archive* file system it will be by default removed from the *data* file system.

Restart files usually should not be archived at the end of a run as they are needed for the next run. It is possible to give the name of the restart file of the previous run as an additional input parameter of `save_file`. Then at the end of the run the current restart file will be stored in the *data* file system but the restart file of the previous run will be archived.

```
#
#-- Log file
#
save_file ${atmmod} log lmdz.x.prt0 \
           ${date}_${enddate}_lmdz.x.prt0
#
#-- Restart file
#
save_file ${atmmod} restart restart.nc \
          ${expid}_${enddate}_restart.nc \
          ${expid}_${prevdate}_restart.nc
```

Two calls of function `save_file` to archive a log and a restart file of the atmosphere model LMDz. The lines are part of the include file `save_output_lmdz.h`. Note that for archiving the restart file `save_file` is called with the name of the restart file of the previous run as a fifth parameter.

Before the file transfer starts the date of the output file is compared with the date of a reference file that was created right before model submission. If the output file is older than the reference file it was not updated at runtime. In that case the function will wait some minutes to assure that the output file is written completely before saving it. This check turned out to be mandatory on some platforms.

As `get_file` function `save_file` prints a log message to standard output.

The following two scripts substitute the complex functionality of function `save_file`, whereby the data processing parts is splitted in three two steps :

1. Merge files together, e.g. concatenate all monthly model output files of a datastream in one yearly file or collect all restart files of several model components of the coupled model in one tar file
2. Transfer the given file to the archive

shell script `merge_files`

Merge files together, e.g. concatenate all monthly model output files of a datastream in one yearly file or collect all restart files of several model components of the coupled model in one tar file

shell script `archive_file`

Transfer the given file to the archive

shell script `remote_transfer`

The `remote_transfer` script is used for data transfer from one machine to another via file transfer protocols as `ftp`, `lftp`, `gridftp`, `ssh`, `ecpopy` or local file system commands. The script replaces the formerly used function `ecfill`. The script is called from functions `get_tarfile`, `get_file` and `save_file`.

function `check_size`

After each file transfer it is checked whether the file was saved completely. The size of the source and the target file are compared using function `check_size`. The function requires six input parameter. These are source file name, target file name, file type, remote transfer protocol, remote host name, remove flag. With this input can be specified whether or not the source and target files are located on the same machine and whether or not the source file should be removed when archiving was successful.

function `submit`

The function carries out the submission of a subsequential job. It depends on the queueing system and whether or not the job is run interactively.

2.7 The calendar tools

Time control of the tasks is accomplished by calendar tools: shell scripts `calc_date`, `days_in_month`, `days_in_year`, `format_date`, `later_date`, `time_between`. These shell scripts are stored in the directory `/root/util/running/functions`.

Except for the initial date of the current run all date and time parameters are available from the user defined setup (compare Section 2.2). The initial date of the current run is read from a short ASCII file called `expid.date`. This file is updated automatically at the end of each run.

Date and time formats supported by calendar tools are listed in the Table 2.6.

Format number	Format
0	<code>yearMMDD[_hh[mm[ss]]]</code>
1	<code>year-MM-DD[_hh[:mm[:ss]]]</code>
2	<code>year-MM-DD[Thh[:mm[:ss]]]</code>
3	<code>year-MM-DD[hh[:mm[:ss]]]</code>
4	<code>year MM DD[hh[mm[ss]]]</code>
5	<code>DD Mon year[hh[:mm[:ss]]]</code>
6	<code>yearMMDD[_hh[:mm[:ss]]]</code>
7	<code>year-MM-DD[_hh[mm[ss]]]</code>
8	<code>YYYYMMDD[hh[mm[ss]]]</code>
9	<code>yearMMDD[_hh[mm[ss]]]</code>

Table 2.6: Date and time formats supported by calendar tools. The default format is 0; format 2 corresponds to ISO 8601; the using of four-digits year specification YYYY is mandatory for format 8.

Chapter 3

IT environment and site specific configurations

SRE is used at several sites and with different IT environments and workflows. This means e.g.

- model execution maybe performed on a specified computing host, which is *not* the submit host, from where you submit your jobs,
- submission of jobs can be performed by a chosen queueing system or can be done interactively
- data management can be done remotely with chosen transfer protocols and dedicated data and archiving servers.

How to adapt the experiment workflow to the underlying IT environment and your specific needs is described in the following.

3.1 Nodes and servers

To enable launching and submission of the tasks of an experiment to specified hosts and servers, the following settings are provided :

- *node* : The compute node on which the model integration is performed. It can be specified with the option `--node | -n node` of `Create_TASKS.frm`. By default it is set to the machine name where you execute `Create_TASKS.frm` (``uname -n``). If you enter a not supported compute node, you get a list of supported nodes as shown in 5.5.
- *submit_host* : The submission or front end host, from which the experiment is launched by the user by launching of `expid.run`. The default setting is the compute host *node* itself. It is required, if the host, from which `expid.run` is *submitted* is not the same as the compute host *node*, where it is *executed* or another task is performed on a remote host. See e.g. the following setting and 3.3
- *data_rem=[processing_host:]processing_path* : Location of the file system where output data are processed, whereby *processing_host* is an optional data server and *processing_path* the absolute path to the data directory. By default this is the same as the 'local' work share path *data*. If post processing is performed on a remote data server *processing_host* and *submit_host* must be set accordingly.
- *archiving_host* : The archiving host, where data are archived. With *archiving=yes* after post processing model output data are transferred onto the host *archiving_host* in the directory tree *archive* by the specified transfer protocol *rtp*.

3.2 Queueing systems

You can launch your experiment by submitting the run script interactively, i.e. just enter `./expid.run`. Usually this is recommendable only for testing or very short jobs. But if you want to execute the model integration on a multi processor HPC server, the run script is submitted by a queueing or job management system to the compute node. This queueing system is assigned by the setting `queueing_system` and the default is site specific.

Not all by IMDI supported queueing systems are available or configured for each site. E.g. the queueing system SGE (Sun Grid Engine) is not supported for NEC SX (node=sx platforms. So, if you change the default setting in the setup file to `queueing_system=SGE`, you get an error message :

```
> Create_TASKS.frm --id SXtest -n sx cosmos-ao
.....
Only NQS2 and INTERACTIVE supported by now...
Please modify SRE header file 'define_queue_run_sx.h'
```

Table 3.1: Error message if a not supported queueing system is entered

In the case you perform remote data processing sometimes another submit command is used for the associated task. This is assigned by the setting `qsub_rem`, which is by default `qsub`, the submit command, which corresponds to `queueing_system`.

3.3 Remote data management

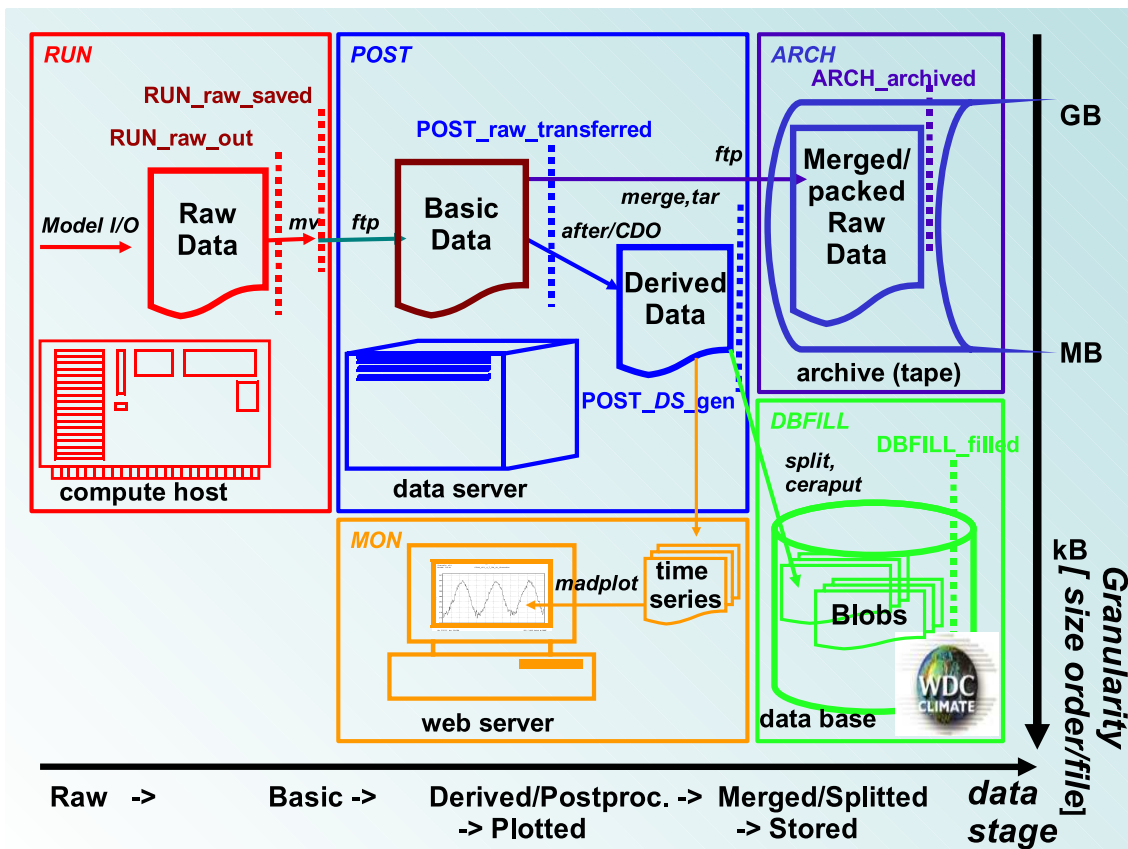


Figure 3.1: The data flow of an experiment *expid*.

... to continue ... !!!

Chapter 4

Running a coupled model

This chapter gives a brief to-do list for running a coupled model within SRE. We assume here that the coupled model *cplmod* and the compute node *node* used are supported by SRE (check the corresponding lists in the tables 5.1 and 5.5). Otherwise you have to adapt SRE to the new coupled model respectively new site as described in chapter 5.

It is furthermore assumed that:

- you have access by SVN to the MaD repository, to check out a IMDI version from there in your working directory as described in chapter 1,
- the executables of all model components and of the coupler are available in the standard directory (*architecture/bin*) on the compute node server having the standard names (*model[_submodel]_cmpvers.MPI[1,2].x*). For details on model compilation please check the PRISM SCE handbook (Legutke et al. (2007)).
- all input data is available
 - either on the compute node as a tar file in *root/data/cplmod* or as separate files distributed over the component models' input directories (compare section 5.1.4).
 - or in the location set by *archive_in*

In the following section we describe in general, how an experiment is performed step by step. Afterwards three concrete examples for common coupled models and supported sites follow.

4.1 Step-by-step usage

These are the steps that need to be taken to run a coupled model within the running environment.

1. Login on the compute node
2. Check out the IMDI ans model sources
3. Go to the directory *util/running/tools*
4. Run the script *Create_TASKS.frm* to generate a setup file for your experiment. Type *Create_TASKS.frm -h -v* for help. More information is given in section 2.4.
5. The setup file (*util/running/tools/setup/setup_cplmod_exp_id*) contains all configurable parameters of the coupled model experiment. The defaults listed in the setup file are reasonable for the coupled model combination and the computing host you are working on. Edit the setup file according to the design and configuration of your experiment (section 2.2).
6. Run *Create_TASKS.frm* again with the same input parameters. The script will check your specifications in the setup. If a parameter is not supported or a combination of parameters does not make sense *Create_TASKS.frm* gives an error message and stops. In this case you have to correct your setup file.

7. Repeat the last two steps, until setup check is passed successfully and the task scripts as *expid.run* etc. are generated in the directory *experiments/expid/scripts*. If you generated the scripts not on the compute host, you have to transfer them into the corresponding directory.
8. Interactive model execution :
 - (a) Login on the compute server (if different from the compile server).
 - (b) Change to the directory *home/expid/scripts*. (The file system *home* was defined in the setup.)
 - (c) Launch the experiment by *./expid.run*.
9. Launch the experiment by submit of the run script by *qsub* (rsp. the corresponding submit command used on the compute node)
 - (a) Login on a submit host from which you can submit jobs onto the compute host (if different from the compile server).
 - (b) Change to the directory *home/expid/scripts*. (The file system *home* was defined in the setup.)
 - (c) Launch your experiment by submitting *qsub expid.run*.

The current state of the experiment can be observed in the working directory *work/expid/work*. The status and course of the experiment can be observed by checking the logfiles *expid.date* and *expid.log* in the scripts directory *home/expid/scripts*.

At the end of each run the output files are moved to the *data* or *archive* file system depending on the setup. The runsript resubmits itself for the execution of the follow-up run until the end of the experiment is reached. If applicable, scripts for postprocessing and archiving are launched automatically at the end of each run.

The workflow for such an experiment - here with monitoring and data base filling performed on a remote host - is shown schematically in figure 4.1.

Concrete Examples for coupled model configurations and IT environements supported by SRE you can find in the appendix C.

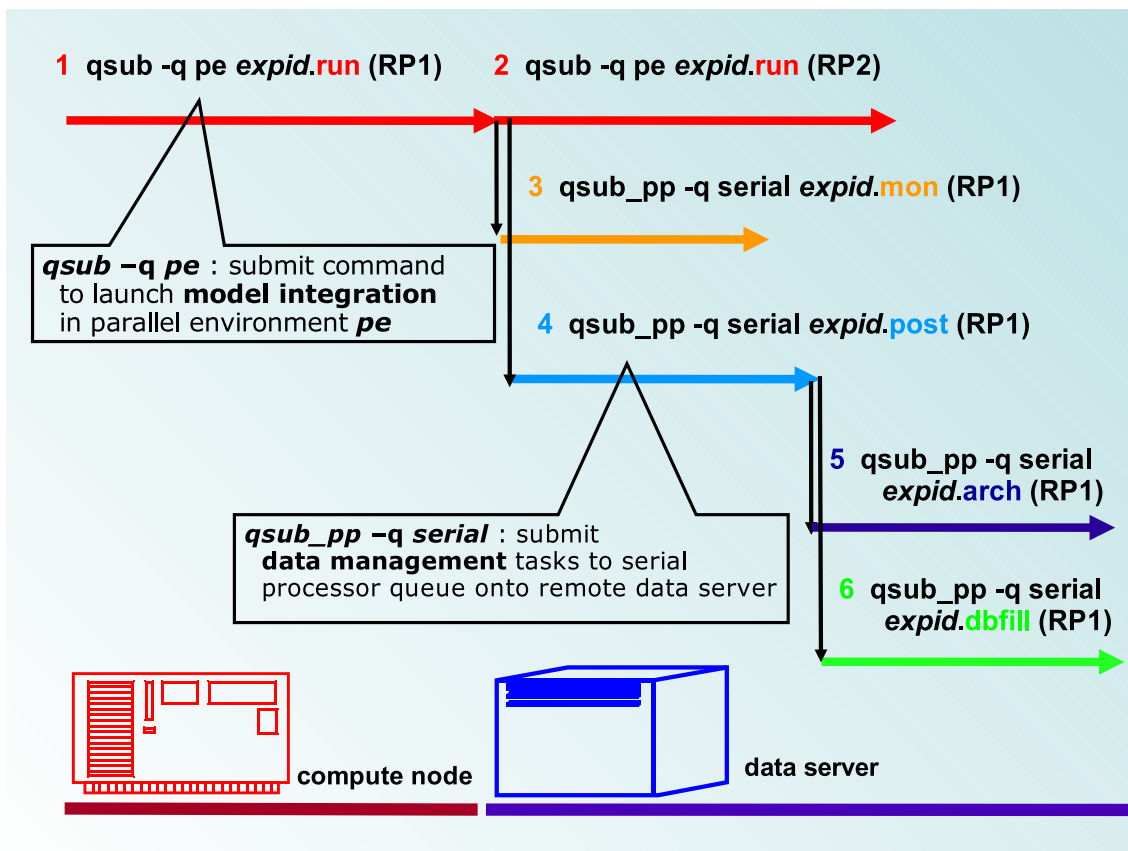


Figure 4.1: The workflow of an experiment *expid*.

Chapter 5

Enlarging the System

The PRISM system facilitates the assemblage of new coupled models and minimises the effort of portability. This is achieved by dividing the tasks into short header files depending on either the model, the coupled model combination or the site. These files are collected to create a runscript specific to the coupled model and site. For a detailed description of the method please consult section 2.4.

The PRISM SCE handbook (?) describes how to adapt model source code to the SCE. This chapter demonstrates the adaption of a coupled model to the SCE (section 5.1) and to run it on a new site (section 5.2).

At the end of the PRISM project 12 component models (including 4 toy models) running in 10 coupled combinations on 12 sites have been adapted to the PRISM system. For the current numbers please check the web page <http://prism.enes.org>.

5.1 Adding a new coupled model

The ksh-script `Create_TASKS.frm` manages the assemblage of header files to tasks that are specific to the coupled model and the site (section 2.4). Running `Create_TASKS.frm` for a coupled model, which is not supported by SRE, leads to the error message and a list of supported models shown in 5.1.

To add a new model to the SRE one has to provide all model specific header files. This includes header files for each new component model and for the new coupled model combination. This section gives an overview on the include files that need to be created and shows their functionalities. It is helpful to compare include files of several adapted models to get an idea of the files compositions.

The include files are assembled using the gnu m4 preprocessor. Some of the include files contain variables that will be replaced by the preprocessor. These m4 variables can be recognised by a common syntax: They all begin with an underscore (`_`) and one capital character followed by lower case characters.

5.1.1 Adaptation of `Create_TASKS.frm`

One of the input parameters of `Create_TASKS.frm` is the coupled model configuration (`cplmod`). From this parameter `Create_TASKS.frm` identifies the model components. The script checks the correctness of the input parameters given. An unknown coupled model name will lead to an error message. Thus the name of the new coupled model (in lower case, including numbers, “-” or “_”) must be added to the list of supported model combinations. Besides it is helpful for other users to mention the new coupled model and its components some lines below when the text for the error output is defined.

The assignment of components to the coupled model name is realised in section “Definition of the model components” of `Create_TASKS.frm`. Please define here the components of the new coupled model using the variables listed in table 5.2. Please use lower case characters for the component model names.

```

Script generation for the coupled model xxx is not possible.
Possible selections are :
cplmod = atmmod + chemod + srfmod + ocemod + icemod + bgcmod : coupler
-----
cclm = cclm + + + + + :
clm = clm + + + + + :
cosmos-a = echam5 + + + + + :
cosmos-ac = + + + + + :
cosmos-ao = echam5 + + + mpiom + + : oasis3
cosmos-aob = echam5 + + + mpiom + + hamocc : oasis3
cosmos-as = echam5 + + jsbach + + + :
cosmos-aso = echam5 + + jsbach + mpiom + + : oasis3
cosmos-asob = echam5 + + jsbach + mpiom + + hamocc : oasis3
cosmos-o = + + + mpiom + + :
cosmos-ob = + + + mpiom + + hamocc :
cosmos-s = + + jsbach + + + :
echam5 = + + + + + :
echam5j = + + + + + :
ips1_cm4 = lmdz + + orchidee + opa + lim + : oasis3
jsbach = + + + + + :
morea = remo + + + roms + + : oasis3
mpi-ao-pisces = echam5 + + + mpi-om + + pisces : oasis3
mpi-aob-ham = echam5 + + + mpi-om + + hamocc : oasis3
mpi-o-pisces = + + + mpi-om + + pisces :
mpiom = + + + + + :
opatoy = toy4opa + + + opa + lim + : oasis3
psmopa = psmodel + + + opa + lim + : oasis3
remo = remo + + + + + :
remo-toy4remo = remo + + + toy4remo + + : oasis3
roms = + + + roms + + :
toyclim = toyatm + toyche + + toyoce + + : oasis3
toymozart3 = toy4mozart3 + mozart3 + + + + : oasis3
toyoa4 = atmoa4 + + lanao4 + oceoa4 + + : oasis4

```

Table 5.1: You get a list of by IMDI supported coupled models as error message returned by `Create_TASKS.frm --id ID xxx` if an unknown coupled model is specified (in this case 'xxx')

Variable	Description	<i>cmp</i>	<i>c</i>	TOYCLIM	COSMOS	CLM
atmmod	atmosphere model	atm	a	toyatm	echam5	clm
chemod	atmospheric chemistry model	che	c	toyche	-	-
srfmod	surface model	srf	s	-	jsbach	-
ocemod	ocean model	oce	o	toyoce	mpiom	-
icemod	ice model	ice	i	-	-	-
bgcmod	bio-geo-chemistry model	bgc	b	-	hamocc	-
coupler	coupler	-	-	oasis3	oasis3	oasis3

Table 5.2: Variable names of the model components. The variables are used to define the components of a coupled model in `Create_TASKS.frm` and in the include files building up the tasks.

5.1.2 Include files depending on the coupled model combination

The include files specific to a coupled model can be identified from the extension `_cplmod`. The node dependent include files are located in directory `util/running/tools/include_node`, the files relevant on all platforms can be found in directory `util/running/tools/include`. Here a short description of all include files depending on the coupled model combination.

config_cplmod.h: All include files starting with `config` are part of the setup. Detailed information on these files is given in section 2.2. The file `config_cplmod.h` defines the configurable variables of the coupled model combination.

config_timecontrol_cplmod.h: This file defines the variables needed for time control. The defaults given are depending on the coupled model. Some models for example only support 30 day months, others only the Gregorian calendar. The variables defined in the time control section of the setup are explained in detail in section 2.2.

complete_setup_cplmod.h: Setup variables that are not configurable for the coupled model are listed here (compare section 2.2).

check_setup_cplmod.h: As the name implies the include file is needed to check the user's selections in the setup. Not all combinations of parameters make sense for a coupled model. These inconsistencies should be identified by the check. The setup check for the coupled model ECHO is considered as a good example.

check_setup_cplmod_node.h: Setup checks specific to the coupled model and to the site are performed in this include file. For example, if it is not possible to run a model on one CPU at a specific site, the check should prevent the users from making this selection.

comments_cplmod.h: The comments file gives a brief introduction to the coupled model combination. It should list at least the model components and the developing institutes. Besides it gives room for any additional information.

define_queue_cplmod_node.h: The file is used to calculate the parameters needed for the queuing system, i.e. time and memory consumption and the number of CPUs on the specific computing host.

get_executables_cplmod.h: The header file contains the commands to copy the executables of the components from the experiment's `bin` directory to the working directory. If the binaries are not available in that directory they are transferred from the compiling host. This is performed by calls to the function `get_file`. To ensure that the latest version of the executables are used, at the beginning of the first run of an experiment the executables are removed from the `bin` directory and transferred anew from the compiling host.

launching_cplmod_node.h: The command to launch the coupled model is defined in this site dependent include file. Commands to generate a profiling protocol can be added.

5.1.3 Include files depending on the component model

The include files specific for a component model can be identified from the suffix `_model`. All header files depending on the component are listed below. For each component of the coupled model an extra version of the files needs to be provided. The include files for the coupler depend on the coupled model combination. Thus the suffix `_model` in the header file names is replaced by `_coupler_cplmod` for the coupler specific include files of this section.

config_postprocessing_model.h: The include file is part of the setup and contains a list of the variables needed for postprocessing of the models' output. Currently postprocessing is supported only with ECHAM5. If one of the component models needs postprocessing the file `post-processing_model.h` has to be generated. Besides `Create_TASKS.frm` has to be adapted:

There is an inquiry on whether or not postprocessing is an option for a component model. The new model has to be added.

config_visualization_model.h: As the file above `config_visualization_model.h` is used to generate the setup (compare section 2.2). Variables needed for visualization of the component models output files are listed. Not included are variables needed for the visualization of the files written by the PSMILe library when the options EXPOUT or IGNOUT are set. The latter configuration is common to all models (see `visualization_coupler.h`).

check_setup_model.h: The include files with `check_setup_` in their names are part of the setup check. As explained in detail in section 2.2 the setup contains all configurable variables of the experiment. The include files `check_setup_model.h` and `check_setup_restart_model.h` scan the model dependent variables and perform consistency checks on them. Please use existing header files as examples.

check_setup_restart_model.h: A setup check specific to the restart variables is performed by this include file.

get_input_model.h: The component models include files are provided in this section. The header file mainly contains a series of calls to function `get_file` (compare section 2.6). All input data files needed for the model component should be listed. Not included are namelists. If the model needs different input files depending on the configuration the filename should contain variables (as e.g. the resolution). If-statements can be used to list files only needed in certain coupled configurations.

namelist_model.h: The namelists are created from the runscrip as a here-document. All namelists for a model component are defined in the header file `namelist_model.h`. The generation of the namelist should be very flexible. All important variables should be configurable through the GUI. Modifications in `namelist_model.h` should be exceptional.

save_output_model.h: The header file comprises a series of calls to function `save_file` (compare section 2.6). Included are output data files, the output written by the PSMILe library (`*.prt*`) and ASCII log files. To avoid overwriting of output files the names should include the beginning and final date of the run (i.e. `filename.date_enddate.nc`). Restart files should not be moved to a remote archive at the end of the current run but at the end of the next run. This is achieved by calling function `save_file` with the restart file name of the previous run as an additional parameter. The header file `save_output_model.h` is part of the runscrip and part of the archiving scrip.

postprocessing_model.h: As mentioned above postprocessing is not supported for all models. The header file `postprocessing_model.h` only needs to be provided if postprocessing is an option for the model. The file is highly dependent on the postprocessing utilities for the model.

save_postprocessing_model.h: The postprocessed output data is archived as defined in `save_postprocessing_model.h`. As in `save_output_model.h` function `save_file` is used.

visualization_cplmod.h: The include file is part of the visualization task. It calls a python script for low end visualization called `LE_parameter.py`. The script and some environment variables need to be set on the visualization host. For information on low end visualization please consult the relevant PRISM report (Meier-Fleischer (2004)).

5.1.4 Providing the input data

Initial data for a PRISM experiment is distributed in tar files. These tar files are located in directory `data/cplmod` of the standard directory tree. They are available from the central PRISM CVS repository. Expanding the tar file leads to the creation of the directory `input` with subdirectories `model` for each component model participating in the experiment.

The input data depending on the horizontal or vertical resolution should contain the resolution acronyms in their names. In particular this is true for input data depending on the resolution of several model

```

> tar tf input_echo_t21_grob_prism_2-2.tar
input/
input/oasis3/
input/oasis3/areas_grob_t21_frac.nc
input/oasis3/grids_grob_t21_frac.nc
input/oasis3/masks_grob_t21_frac.nc
input/oasis3/nweights_grob_t21_frac
input/oasis3/rmp_atmo_to_oces_BILINEA_grob_t21.nc
input/oasis3/rmp_atmo_to_oceu_BICUBIC_grob_t21.nc
input/oasis3/rmp_atmo_to_ocев_BICUBIC_grob_t21.nc
input/oasis3/rmp_oces_to_atmo_CONSERV_FRACAREA_grob_t21.nc
input/echam5/
input/echam5/T21L19_jan_spec.nc
input/echam5/T21grob_jan_surf.nc
input/echam5/T21_amip2sst_clim.nc
input/echam5/T21_amip2sic_clim.nc
input/echam5/T21_O3clim2.nc
input/echam5/T21grob_VLTCLIM.nc
input/echam5/T21grob_VGRATCLIM.nc
input/echam5/T21_TSLCLIM.nc
input/echam5/surrta_data
input/echam5/hdpara.nc
input/echam5/hdstart.nc
input/mpi-om/
input/mpi-om/BEK_grob
input/mpi-om/anta_grob.ext8
input/mpi-om/arcgri_grob.ext8
input/mpi-om/topo_grob
input/mpi-om/SURSAL_grob.ext8
input/mpi-om/INITEM_grob.L20.ext8
input/mpi-om/INISAL_grob.L20.ext8

```

Table 5.3: Content of the initial data tar file for the coupled model ECHO. Component models are ECHAM5 running in the horizontal resolution T21 with 19 vertical levels, MPI-OM in the horizontal resolution grob with 20 vertical levels and OASIS3.

components. For example the ECHAM5 input file `T21grob_VLTCLIM.nc` depends on the ECHAM5 resolution (T21) and on the MPI-OM resolution (grob). The file names themselves and the grid acronyms correspond to the names given by the model developers. Note that they do not necessarily match the PRISM standard (lower case).

At run time the input data is transferred from the input directory to the working directory. At the same time the files are renamed to match the file names requested by the component models.

As mentioned in chapter 1 the input data is stored additionally in an input data archive `archive_in/-data`. This input data archive can be located on a remote archiving machine and might be used by several users for several experiments with various model combinations. Directory `archive_in/data` has a subdirectory for each component model and for the coupler. The model directories again have subdirectories giving the model horizontal resolution. The coupler has a subdirectory for each of the coupled model combinations.

When the input data tar file of a coupled model is expanded at runtime the input data archive is filled automatically.

5.1.5 Providing adjunct files

Some model components have namelist like input files in ASCII format. These adjunct files are not distributed within the input data tar-files coming with all coupled models. The adjunct files are closely related to the model source code and profit from CVS version control.

Regular namelists are generated as here-documents at run time. Relevant include file for writing the namelists are `namelist_model.h`. It is recommended to generate the adjunct files as here-documents as well. If this is not possible for some reason the files are placed in the adjunct files directory `util/running/adjunct_files/model`.

The script `Create_TASKS.frm` is generating the tasks and transfers them to the experiments directory on the compute server. At the same time the adjunct files are transferred to the input directory of the specific experiment.

The namcouple

The coupler OASIS3 reads the information on the coupling algorithm from a file called `namcouple`. Detailed information on this file is given in the OASIS3 User Guide (Valcke et al. (2004a)). Each of the coupled model combination uses a special version of this file. The `namcouple` depends on many of the configurable variables of the setup. For that reason the coupled models are provided with a `namcouple` base file in the adjunct files directory. This base file contains variables that are replaced at runtime. The related source code of the runscript can be found in the include file `namelist_oasis3_cp1mod.h`. Namcouple variables are identified from a leading `#` followed by a capital letter. A list of commonly used namcouple variables is presented in the table below (table 5.4).

Variable	Description
<code>#Nmseq</code>	Maximum number of fields exchanged sequentially
<code>#Channel</code>	Message passing method
<code>#Mod1procs</code>	Number of processors used by model 1, Number of processors of model 1 involved in the OASIS3 communication, optional argument
<code>#Mod2procs</code>	Number of processors used by model 2, Number of processors of model 2 involved in the OASIS3 communication, optional argument
<code>#Mod...</code>	
<code>#Jobname</code>	Job ID composed of three characters
<code>#Runtime</code>	Runtime in seconds
<code>#Inidate</code>	Initial date of the run (yyyymmdd)
<code>#Nlogprt</code>	Parameter controlling the standard output extent
<code>#Caltype</code>	Calendar type
<code>#Dta2o</code>	Exchange interval in seconds (atmosphere to ocean)
<code>#Dto2a</code>	Exchange interval in seconds (ocean to atmosphere)
<code>#Dtb2a</code>	Exchange interval in seconds (bio-geo-chemistry to atmosphere)
<code>#Dt...</code>	
<code>#Stat_field01</code>	Status of the exchange field no 1
<code>#Stat_field02</code>	Status of the exchange field no 2
<code>#Stat_field...</code>	
<code>#Cnfileaw</code>	Restart file containing the atmosphere fields
<code>#Cnfileow</code>	Restart file containing the ocean fields
<code>#Cnfile...</code>	
<code>#Lona, #Lata</code>	Dimensions of the atmosphere model grid
<code>#Lono, #Lato</code>	Dimensions of the ocean model grid
<code>#Lon..., #Lat...</code>	
<code>#Norma</code>	Option for the normalization with SCRIP remapping
<code>#Order</code>	Order of conservative SCRIP remapping
<code>#Iseq</code>	Sequential field index for sequential exchange
<code>#Laga2o</code>	Time lag of field exchange (atmosphere to ocean)
<code>#Lago2a</code>	Time lag of field exchange (ocean to atmosphere)
<code>#Lag...</code>	
<code>#Extrapwr</code>	Enforce rewriting of the extrapolation matrix

Table 5.4: Variables used in the `namcouple` base file. At runtime these variables are updated with the current values. More information on the variables is given in section 2.2 and in the OASIS3 User's Guide Valcke et al. (2004a).

5.2 Adaptation of the SRE to a new site

Running `Create_TASKS.frm` on a new machine or specifying a not supported node name with option `-n node` leads to the error message shown in 5.5.

```

No include file directory include_XXX exists for the machine specification XXX
Supported symbolic nodes or platforms are
aix
crayx1
cs
elnino
hpcsun
linux-x64
linux86
```

Table 5.5: You get a list of by IMDI supported coupled models as error message returned by `Create_TASKS.frm` `--id ID -n XXX cosmos-asob` if an unknown node name is specified (in this case 'XXX')

The first step to enlarging the SRE for a new machine is the adaption of `Create_TASKS.frm`. The script contains a function called `get_node_name`. From the node name of the calling machine (or if available from the third input parameter of `Create_TASKS.frm`) this function defines four variables that are needed for script generation:

node_fullname: The complete node name of the computing host as obtained from the command `uname -n`. This node name is needed for ftp.

node: The node name of the computing host; on multi node machines only the part of the name that is identical for all nodes. This node name is part of the site dependent include file names of the SRE.

node_compile: The complete node name of the compile server as obtained from the command `uname -n`. This node name is needed for ftp.

node_comp: The node name of the compile server; on multi node machines the part of the name that is identical for all nodes. This node name is part of the site dependent include file names of the standard compiling environment.

If the node names of the compute server and of the compile server are identical and if you are working on a single node machine nothing needs to be done. In that case all node name variables listed above are identical which is the default. On the other hand, for sites with separate machines for model compilation and model integration or sites with a multi node machine function `get_node_name` in `Create_TASKS.frm` has to be adapted. A new "if block" has to be introduced defining the four node name variables listed above.

At some sites the compile server and the compute server are separate machines, but as the file systems of both machines are mounted ftp transfer is not needed (or even not supported). This is true for the DKRZ machines cross (cross compilation) and hurrikan (compute sever). In this case the variables `node_fullname` and `node_compile` are identical (cs) to suppress the ftp. The variable `node_comp` is defined as ds, which is the extension of node dependent include files used in the SCE.

5.2.1 Node dependent include files

The next step is the creation of the directory `util/running/include_node` with `node` being the node name of the computing host (variable `node`). This directory contains all site dependent include files used for script generation. It might be helpful to compare the include files of other PRISM sites in the directories `include_node` to get an idea of the content and structure of the different include files. Comments in the files give additional help.

Here a list of the site dependent include files. The `node` variable in the file names indicates site dependency. It must be replaced by the node name of the computing host. Besides some of the include files

depend on the coupled model combination. These file names additionally contain the coupled model name (*cplmod*).

config_commands_node.h: The include files with `config` in their names are part of the setup. The setup is generated when `Create_TASKS.frm` is called for the first time. Detailed information on the setup can be found in section 2.2. The include file `config_commands_node.h` gives defaults for some important Unix commands.

config_mpi_node.h: Parameters having to do with message passing are defined in `config_mpi_node.h`. The defaults given are appropriate for the machine.

config_site_node.h: System specific variables are defined here. This includes variables defining the output buffer size or machine specific debugging options.

check_setup_cplmod_node.h: As the name implies this include file is used to perform a site dependent setup check. If for example a model component cannot run on more than one node on the specific site any other selection should lead to an error exit.

define_queue_cplmod_node.h: The queueing system of the compute server needs information on the CPU time consumption and memory usage of a coupled model running at the specific site. The requested resources depend on the length of the simulation, on the resolution and on the number of CPUs.

job_directory_node.h: This include file defines the name of the task and the directory in which it is located. The variables are used for submission of follow-on tasks. For interactive runs these variables are defined using the Unix commands `dirname` and `basename`.

save_logfile_node.h: The logfile of a run cannot be archived at the end of the run as it is not available until the run is over. The file is saved at the beginning of the following run. The name of the log file and the place where it is written are site dependent.

launching_cplmod_node.h: The command to launch a coupled model is given in `launching_cplmod_node.h`. It depends not only on the site and the coupled model but also on the message passing method. Besides the commands to generate a profiling protocol are listed.

Appendix A

Example for a setup file

```
#Nota bene!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
#Nota bene!
#Nota bene!   The USER INTERFACE of this setup file needs to be adapted to the
#Nota bene!   design of your experiment.
#Nota bene!   The comments give possible choices for all variables.
#Nota bene!
#Nota bene!   Warning: Create_TASKS parses this set up file.
#Nota bene!   Do not change anything but the variables.
#Nota bene!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
#####
#
#       SETUP OF EXPERIMENT D2
#
#####
#-- Experiment ID
#
export expid=D2
#
#-- Coupled model name
#
export cplmod=cosmos-asob
#
#-- Node name of the computing host
#
export node=linux-x64
#-----
# Setup of COSMOS-ASOB
#-----
#-- Component model names
#
atmmod=echam5
srfmod=jsbach
ocemod=mpiom
bgcmod=hamocc
coupler=oasis3
components="echam5 jsbach mpiom hamocc"
#####
#
#       1. USER INTERFACE
#
#####
#-----
#       1.1 COMPONENT MODELS
#-----
#
#-- ECHAM5
#
res_atm=T31      # horiozontal grid resolution
#                T21 / T31 / T42 / T63 / T85 / T106 / T159
vres_atm=19     # number of vertical levels
#                19 / 19 / 19 / 31 / 31 / 31 / 31
atmvers=I2      # atmosphere model version (used in executable name)
out_filetype=GRIB # output file format: GRIB / NETCDF
arch_format=RAW # archive file format: RAW / GRIB_SZIP
dt_write_atm=6  # time interval of output writing in hours
lhd=yes         # hydrological discharge model activated
co2_transport=true # true: prognostic CO2 mass mixing ratio
#               # false: uniform volume mixing ratio of CO2
millennium_ctrl=true # use namelist parameters for permanent year 800 (RADCTL)
volc_forc=false  # true: volcanic forcing: provide volc_data and rad_table
save_dblrad=false # true: handle instantaneous flux anomalies (accuflx)
#
#-- JSBACH
#
ntiles=4        # number of tiles
read_cpools=false # read cpools from file at the beginning of an initialized experiment:
#               # true/false
land_use=false  # use land use change (read yearly files of cover fraction)
#               # true/false
refyear=800     # reference year of initial data (e.g. cpools)
```

```

lctlivers=""          # lctlib file version. Default: "". Chose "albedo_snow"
                      # for millennium-like setup (no litter pools).
jsb_standalone=false
#-- MPIOM
#
res_oce=GR30         # horiozontal grid resolution (acronym)
                      # GR30 / GR15
vres_oce=40         # number of vertical levels
                      # 20/40 / 40
ocevers=I2          # ocean model version (used in executable name)
#-- HAMOCC
#
#-- COUPLER
#
jobname=D2x         # OASIS experiment-id (3 characters)
nlogprt=0           # Standard output extent:
                      # 0: little
                      # 1: much
                      # 2: very much std. output
ncplvers=""         # namcouple version
                      # "" for the default namcouple
                      # for usage of an altenative namcoule: place it in
                      # prism/util/running/adjunct_files/oasis3
                      # and append the namcouple version to its name:
                      # namcouple_'cplmod'ncplvers'
run_mode=concurrent # sequential / concurrent (=parallel)
# remapping parameters
scripwr=0          # writing of SCRIP remapping matrices
                      # 0: use SCRIP matrice if existing; else (re)calculation
                      # 1: unconditional (re)calculation of SCRIP matrices
gridswr=0          # writing of grid description files for OASIS
                      # 0: use grid descript. files if existing; else (re)generation
                      # 1: unconditional (re)generation of grid description files
extrapwr=1         # writing of extrapolation matrix (NINENN)
                      # 0: use of existing extrapolation matrix (error if not
                      # available)
                      # 1: unconditional (re)generation of the extrapolation matrix
                      # Note: if gridswr=1 extrapwr needs to be 1 too
# time interval of data exchange
dto2a=86400        # coupling time step from ocean to atmosphere (86400/43200) [s]
                      # Note: coupling time step from atmosphere to ocean is 86400 s
# treatment of coupling fields (see OASIS documentation for more information)
timtranso2a=AVERAGE # INSTANT / AVERAGE
export=EXPORTED    # EXPORTED / EXPOUT
# number of processors
integer nproca_atm nprocb_atm nproma_atm nthreadatm nproca_oce nprocb_oce nthreadoce
nproca_atm=8       # (8 is default on tornado / =5 for T63 on hurrikan)
nprocb_atm=3       # (3 is default on tornado / =1 on hurrikan) total number of MPI processors for the atmosphere
nproma_atm=64      # (64 is default on tornado / =1024 on hurrikan / =3840 for T63) vector length (http://svn.zmaw)
nthreadatm=1       # (1 is default) number of openMP threads for the atmosphere model
nproca_oce=3       # (3 is default on tornado / =1 on hurrikan)
nprocb_oce=9       # (9 is default on tornado / =1 on hurrikan / =3 for T63) total number of MPI processors for the ocean
nthreadoce=1       # (1 is default on tornado / =3 on hurrikan) number of openMP threads for the ocean model
#-----
# 1.2 TASK SPECIFICATION
#-----
#-- preprocessing: yes: create and run a script for preprocessing
#                  no: no preprocessing
preprocessing=no
#
#-- postprocessing: yes: create and run a script for postprocessing
#                  no: no postprocessing
postprocessing=yes
#
#-- splitting:     yes: split multi code output in codes (only possible, if postprocessing=yes)
#                  no: no splitting of codes
splitting=no
#
#-- dbfill:        yes: fill database according code lists (only possible, if postprocessing=yes and splitting=yes)
#                  no: no data base filling
dbfill=no
#
#-- monitoring :   yes: create and run a script for monitoring
#                  no: no monitoring
monitoring=yes
#
#-- archiving:     yes: create and run a script for archiving
#                  no: no archiving
archiving=no
#-----
# 1.3 TIME CONTROL
#-----
#-- declarations
integer nyear nmonth nday nhour nminute nsecond
#
#-- calendar type: Available calendar options:
# 0 : No leap year (365 days per year)
# 1 : Gregorian (365/366 days per year)
# n : Equal months of "n" days (30 for 30 day months)

```

```

caltype=1
#
#-- initial and final date of the experiment
#   Format: YearMMDD[_hh[mm[ss]]], Year-MM-DD[_hh[:mm[:ss]]] or
#           Year-MM-DD[Thh[:mm[:ss]]]
#   Note: The experiment will not stop within a run/chunk even if the
#         final date is reached.
initial_date=0800-01-01 # initial date of the experiment
final_date=0803-12-31  # final date of the experiment
#
#-- duration of a run/chunk
#   Specify the length of each run in one of the below units.
nyear=1      # number of years per run
nmonth=0     # number of months per run
nday=0       # number of days per run
nhour=0      # number of hours per run
nminute=0    # number of minutes per run
nsecond=0    # number of seconds per run
nstep_atm=0  # number of atmosphere model time steps per run
nstep_oce=0  # number of ocean model time steps per run
#-----
#   1.4 ARCHIVE
#-----
#-- use_initial_tarfile - Get initial data from a tar file
#   yes:  get initial data tar file from 'archive_in'
#   no:   all initial data is in place (short term archive 'data')
use_initial_tarfile=yes
#
#-- tag to distinguish between different input data files
#
tag=""
#
#-- file and directory permissions of the output
#
export dir_permits=755
export file_permits=644
#-- fill_archive_in - Store data from the tarfile as single files in
#                   'archive_in'. This allows for usage with different experiments
#                   (only with use_initial_tarfile=yes)
#   yes:  the data shall be stored as single files in 'archive_in'
#   no:   'archive_in' is not used
fill_archive_in=no
suppress_links=yes
#
#-- archiving_onlyraw
#   yes:  archive only raw files (default)
#   no:   archive as well postprocessed files
archiving_onlyraw=yes
#
#-- archiving_host - Node name of an archiving machine (if different from the
#                   compute or data processing platform )
# export archiving_host="cross.dkrz.de" setting if you run your model at on cross, but data processing is performed
# export archiving_host="gridftp.dkrz.de" setting for running on tornado and archiving by gridftp on the DKRZ
export archiving_host=""
#-----
#   1.5 MESSAGE PASSING
#-----
#-- message passing library with which the models and the coupler were compiled
#   (MPI1 or MPI2) and shall be launched (mpich/mpich2/openmpi/"")
mpi_library=openmpi
#
#-- launching mode (spawned by coupler: MPI2; MPI1 otherwise)
#
message_passing=MPI1
#
#-- buffered MPI Send
#   yes: buffered send
#   no:  simple send
bsend=no
#####
# MPIBIN according to site and compiler for calls of mpiexec/mpirun
#
#-- name of compiler (set according to Create_TASKS parameter specifications)
#
compiler=intel
case "${compiler}" in
  pgi)
MPIBIN=/sw/sles9-x64/ofed/openmpi-1.2.5-pgi7/bin
;;
  nag)
MPIBIN=/sw/sles9-x64/ofed/openmpi-1.2.5-nag/bin
;;
  sun)
  #MPIBIN=/opt/ofed/openmpi-1.2.5-sun12/bin      # GKSS
  #MPIBIN=/opt/openmpi-1.2.8/sun/bin           # DWD
MPIBIN=/sw/sles9-x64/ofed/openmpi-1.2.5-sun12/bin # ZMAW
;;
  intel)
MPIBIN=/sw/sles9-x64/ofed/openmpi-1.2.5-intel/bin
;;
  path)

```

```

#MPIBIN=/opt/scali/bin # GKSS
#MPIBIN=/sw/sles9-x64/ofed/openmpi-1.2.5-path/bin
*)
echo Compiler $compiler not supported.
exit 1
esac
#
#-- number of processors used for the mpiexec
#
nprocmpi=0
#-----
# 1.6 FILE SYSTEMS
#-----
#-- home: Permanent file system for the SCRIPTS on the COMPUTING HOST
#         (only needs to be specified if the tasks are NOT generated on the
#         computing host)
export home=/scratch/work/bm0021/k204019/cosmos-dev/experiments
#
#-- archive_in: Root directory of the LONG TERM INPUT data archive. It needs
#               to reside on the same machine as the output archive. This
#               archive is intended for input data that is needed with
#               several experiments, e.g. initial , forcing or restart files.
# - The parent-directory of ${archive_in} needs to exist before submission of the job-
export archive_in=/pool/data/COSMOS/cosmos2
#
#-- data: Root directory of the SHORT TERM data server.
#         Model INPUT and OUTPUT will be read from/written to
#         this file system of the computing host
# - The parent-directory of ${data} needs to exist before submission of the job
export data=/scratch/wrkshr/k204019/cosmos-dev/experiments
#
#-- archive: Root directory of the LONG TERM OUTPUT data archive.
#            - Either a filesystem of the computing host or of a remote archiving host.
#            If ${archive} differs from ${data} model output will be saved in
#            ${archive} and removed from ${data}.
# - The parent-directory of ${archive} needs to exist -
export archive=${PRJHOME}/arch/k204019/cosmos-dev/experiments
#
#-- work: Root directory for the temporary working directory
#         (for production runs use $TMPDIR on NEC)
#
work=/scratch/wrkshr/k204019/cosmos-dev/experiments
#
#-- Compilation
#
#         compile_server: Node name of the compile-server
#         compile_path: Directory where the executables are stored
#                       on the compile-server
compile_server=linux-x64
compile_path=/scratch/work/bm0021/k204019/cosmos-dev/x86_64-intel/bin
#
#-- Path to the IMDI function directory
#
export fpath=/scratch/work/bm0021/k204019/cosmos-dev/util/running/functions
export PATH=${fpath}:$PATH
#-----
# 1.7 RESTART CONTROL
#-----
#-- 'component'_restart: start from restart or initial files (climatology)
#         1 : start experiment from restart files for 'component'
#         0 : start experiment from initial conditions for 'component'
#         If the experiment starts from restart files you need to specify:
#
#         'component'_age: the age of the restart file used in years
#         'component'_restart_file: filename of the restart file (including path)
#
atm_restart=1
atm_age=0
atm_restart_file=${data}/${expid}/restart/echam5/rerun_echam.nc
atm_restart_tracer=${data}/${expid}/restart/echam5/rerun_tracer.nc
atm_restart_co2=${data}/${expid}/restart/echam5/rerun_co2.nc
atm_restart_accu=${data}/${expid}/restart/echam5/rerun_accuflx.nc
hd_restart_file=${data}/${expid}/restart/echam5/hdrestart.nc
srf_restart=1
srf_age=0
srf_restart_jsbach=${data}/${expid}/restart/jsbach/rerun_jsbach.nc
srf_restart_surf=${data}/${expid}/restart/jsbach/rerun_surf.nc
srf_restart_veg=${data}/${expid}/restart/jsbach/rerun_veg.nc
oce_restart=1
oce_age=0
oce_restart_file=${data}/${expid}/restart/mpiom/rerun_mpiom.ext
bgc_restart=1
bgc_age=0
bgc_restart_file=${data}/${expid}/restart/hamocc/rerun_hamocc.nc
cpl_restart=1
cpl_restart_file_sstocan=${data}/${expid}/restart/oasis3/sstocan.nc
cpl_restart_file_flxatmos=${data}/${expid}/restart/oasis3/flxatmos.nc
#-----

```

```

# 1.8 PLATFORM DEPENDEND SPECIFICATIONS
#-----
#-- set system stuff required:
#
if [ "${sitename}" = "dkrz.de" ]; then
  if [ "${MODULESHOME:-}" != "" ]; then
    . ${MODULESHOME}/init/ksh
    module load SunGridEngine/6.0u10 globus/4.0.3
  else
    echo "ERROR: module command not available."
    exit 1
  fi
fi
#
#-- batch queueing system ( PBS | SGE | LL | NQS2 | LSF )
#
#queueing_system=PBS ; n_nodes=24 # total number of nodes
queueing_system=SGE
#
#-- email address (for queueing system)
#
email=your@email
#
#-- account (for queueing system)
#
account=default # DKRZ account number ("default" for default account)
#
#-- queue (for queueing system)
#
queue=default # queue name ("default" for default queue)
#-----
# 1.9 UNIX COMMANDS
#-----
export mkdir="mkdir -p" # create a new directory
export cp="cp -p" # copy without changing the time stamp
export ln="ln -sf" # soft link (if no links are wanted: same as cp)
export rm=rm # remove
export rtp=gridftp # remote transfer protocol
export rtp_post=$rtp # transfer protocol to remote processing host
export put_archive="" # special command to put files to band archive (e.g. dsmc)
export get_archive="" # special command to get files from band archive (e.g. dsmc)
export gunzip="gzip -d" # unzip a file that was zipped using gzip
export job_account="" # command to receive job account at the end of the run
export cdo=/client/bin/cdo # climate data operator
export python=/client/bin/python # python
#-----
# 1.10 REMOTE DATA PROCESSING
#-----
#-- Perform data processing on remote host: yes/no
#
postprocessing_rem=no # yes/no
monitoring_rem=no # yes/no
dbfill_rem=no # yes/no
archiving_rem=no # yes/no
#
#-- data_rem: Location, where data processing (postprocessing etc.) should be performed.
# It is specified as [processing_host:][processing_path], e.g.
#
# data_rem=$data default, i.e. processing in the directory
# ($data) on the compute (frontend) host
# data_rem=mil00.zmaw.de:/mil00 TPP: processing on host mil00.zmaw.de
# in the working directory /mil00
data_rem=$data
[[ $(print ${data_rem} | grep :) = "" ]] && host_rem="" || host_rem=${data_rem%%:*}
path_rem=${data_rem#*:}
#
#-- qsub_rem: Submit command for the processing jobs on remote host
#
# qsub_rem=$qsub default, i.e. processing jobs are submitted
# with the same command as the run job ($qsub)
# qsub_rem=sge_qsub TPP
# qsub_rem='ssh [-l user] nohup' processing on remote host without
# queueing system (interactive)
qsub_rem=$qsub
qsub_rem_sync=$qsub_rem' -sync y' # wait for the job to complete before exiting
#
#-- Path to the IMDI function directory on remote processing host
#
fpath_rem=${path_rem}/${expid}/functions
[[ "${fpath}" = "${fpath_rem}" ]] || export PATH=$PATH:${fpath_rem}
#
#-- Submit host (i.e. compute or frontend node)
#
submit_host=cross.dkrz.de
#
#-- chron_proc: Data processing in chronological order: yes/no
#
chron_proc=yes
#-----
# 1.11 PRE PROCESSING
#-----
# 1.12 POST PROCESSING
#-----

```

```

# no suffixes for by CDO splitted files
export CDO_DISABLE_FILE_SUFFIX="1"
#
#-- afterburner: for information on the afterburner see
#      http://www.mpimet.mpg.de/en/extra/models/echam/prepost.php
afterburner=$(whence after) || afterburner=/pool/ia64/afterburner/bin/after
#
#-- Code lists for postprocessing of ECHAM5 model output (afterburner)
#
# 2-dimensional (BOT) variables included in postprocessing (monthly means)
codelist_echam5_BOT_mm="83,84,85,86,87,88,91,92,93,94,95,96,97,
102,103,104,105,106,107,108,109,110,111,112,113,
114,115,116,117,119,120,121,122,123,124,
134,137,139,140,141,142,143,144,145,146,147,150,151,160,161,164,165,166,
167,168,169,171,175,176,177,178,179,180,181,182,184,
185,186,187,188,193,197,203,204,205,208,209,
210,211,213,214,216,218,219,221,222,229,230,231,233,235,260"
# 3-dimensional atmosphere (ATM) codes/variables included in postprocessing (monthly means)
codelist_echam5_ATM_mm="130,131,132,133,135,138,148,149,153,154,155,156,157"
codelist_echam5_ATM_mm_type30="130,131,132,133,135,154,156,157,223"
# use type=70 in afterburner for codes 138,148,149,155
codelist_echam5_ATM_mm_type70="138,148,149,155"
# 6h derived data only generated in post postprocessing if splitting=yes set !
codelist_echam5_BOT_6h="83,84,85,86,87,88,91,92,93,94,95,96,97,
102,103,104,105,106,107,108,109,110,111,112,113,
114,115,116,117,119,120,121,122,123,124,
134,137,139,140,141,142,143,144,145,146,147,150,151,160,161,164,165,166,
167,168,169,171,175,176,177,178,179,180,181,182,184,
185,186,187,188,193,197,201,202,203,204,205,208,209,
210,211,213,214,216,217,218,219,221,222,229,230,231,233,235,237,259,260"
# 3d (ATM) 6 hourly
codelist_echam5_ATM_6h="130,131,132,133,135,138,148,149,153,154,155,156,157,259"
codelist_echam5_ATM_6h_type20="153,154,223"
codelist_echam5_ATM_6h_type30="130,131,132,133,135,156,157"
# use type=70 in afterburner for codes 138,148,149,155
codelist_echam5_ATM_6h_type70="138,148,149,155"
# 2-dim echam5 co2 stream variables
codelist_echam5_co2_mm="5,6,7,8,17,20"
# dm derived data only generated in post postprocessing if dbfill=yes set !
codelist_echam5_co2_dm="5,8"
# 2-dim echam5 tracer stream variables
codelist_echam5_tracer_mm="1"
# 2-dim echam5 flux anomalies for double radiation
[ "${save_dblrad}" = "true" ] && codelist_echam5_accuflux_mm="73,74,75,76,77,78"
# List of vertical pressure levels for the 3-dimensional ATM stream (afterburner)
levellist_echam5_ATM="100000,92500,85000,77500,70000,60000,50000,40000,30000,25000,20000,15000,10000,7000,5000"
# List of vertical model(!) levels for the 3-dimensional ATM stream (afterburner)
model_levellist_echam5_ATM="1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19"
#
#-- Code lists for postprocessing of JSBACH model output (monthly means)
#
# Note : code 13 and 125 are generated
codelist_jsbach_main_mm="21,22,55,107,124,126,127,148"
# Note : code 125 are generated
codelist_jsbach_land_mm="40,44,60,67,68,76,107,109,110,124,126,127,148,149"
codelist_jsbach_veg_mm="150,151,152,153,154,156,157,158,160,161,162,163,164,170,174,175,176"
[[ ${jsb_standalone} != true ]] && codelist_jsbach_surf_mm="203,204,205,206,207,212,213"
# dm and 24h derived data only generated in post postprocessing if dbfill=yes set !
codelist_jsbach_main_dm="21,22,55,107,124,126,127,148"
codelist_jsbach_land_dm="40,44,60,67,68,76,107,109,110,124,126,127,148,149"
codelist_jsbach_veg_dm="150,151,152,153,154,156,157,158,160,161,162,163,164,170,174,175,176"
codelist_jsbach_veg_24h="181,182"
[[ ${jsb_standalone} != true ]] && codelist_jsbach_surf_dm="203,204,205,206,207,212,213"
#
#-- Code lists for postprocessing of MPIOM model output
#
# Note : for MPIOM no code specific post processing exists
#
#-- Code lists for postprocessing of HAMOCC model output
#
# Note : for HAMOCC no code specific post processing exists
#
-----
# 1.13 MONITORING
# -----
# 1.14 DATA BASE FILLING
# -----
#####
#
# END OF THE USER INTERFACE
#
#####

```

Appendix B

Example for a run script

```
#!/bin/ksh
#####
#
#           D2.run
#   Automatically generated by Create_TASKS.frm using the m4 macro processor.
#####
#
#   R U N - Script for the model configuration cosmos-asob
#
#   COSMOS-ASOB is a coupled model configuration with the components
#   ECHAM5 - global atmosphere GCM (MPI-HH)
#   JSBACH - global land surface model (MPI-HH)
#   MPIOM  - global ocean GCM (MPI-HH)
#   HAMOCC - global ocean bio-geo-chemistry model (MPI-HH)
#
#####
#$ -S /bin/ksh
#$ -o $JOB_NAME.o$JOB_ID
#$ -j y
#$ -cwd
#$ -q serial
### Batch Queuing System is SGE
#$ -m n
#####
#-----
#
#           0. PROLOGUE
#-----
#
set -e
export task=RUN      # The task: RUN, ARCH, POST, VIS, MON, REM
print "\n This ${task} script\n - is started at\t$(date)\n - running on host\t$(hostname)"
#####
#
#           SETUP OF EXPERIMENT D2
#####
#
#-- Experiment ID
#
export expid=D2
#
#-- Coupled model name
#
export cplmod=cosmos-asob
#
#-- Node name of the computing host
#
export node=linux-x64
#-----
# Setup of COSMOS-ASOB
#-----
#-- Component model names
#
atmmod=echam5
srfmod=jsbach
ocemod=mpiom
bgcmod=hamocc
coupler=oasis3
components="echam5 jsbach mpiom hamocc"
#####
#
#           1. USER INTERFACE
#-----
#
#           1.1 COMPONENT MODELS
#-----
#
#-- ECHAM5
#
res_atm=T31          # horiozontal grid resolution
#                   # T21 / T31 / T42 / T63 / T85 / T106 / T159
vres_atm=19         # number of vertical levels
```

```

#           19 / 19 / 19 / 31 / 31 / 31 / 31
atmvers=I2      # atmosphere model version (used in executable name)
out_filetype=GRIB # output file format: GRIB / NETCDF
arch_format=RAW # archive file format: RAW / GRIB_SZIP
dt_write_atm=6  # time interval of output writing in hours
lhd=yes        # hydrological discharge model activated
co2_transport=true # true: prognostic CO2 mass mixing ratio
                # false: uniform volume mixing ratio of CO2
millennium_ctrl=true # use namelist parameters for permanent year 800 (RADCTL)
volc_forc=false  # true: volcanic forcing: provide volc_data and rad_table
save_dblrad=false # true: handle instantaneous flux anomalies (accuflx)
#
#-- JSBACH
#
nfiles=4        # number of tiles
read_cpools=false # read cpools from file at the beginning of an initialized experiment:
                # true/false
land_use=false  # use land use change (read yearly files of cover fraction)
                # true/false
refyear=800     # reference year of initial data (e.g. cpools)
lctlbvers=""    # lctlb file version. Default: ". Chose "albedo_snow"
                # for millennium-like setup (no litter pools).
jsb_standalone=false
#
#-- MPIOM
#
res_oce=GR30    # horizontal grid resolution (acronym)
                # GR30 / GR15
vres_oce=40     # number of vertical levels
                # 20/40 / 40
ocevers=I2      # ocean model version (used in executable name)
#
#-- HAMOCC
#
#-- COUPLER
#
jobname=D2x     # OASIS experiment-id (3 characters)
nlogprt=0       # Standard output extent:
                # 0: little
                # 1: much
                # 2: very much std. output
ncplvers=""     # namcouple version
                # "" for the default namcouple
                # for usage of an alternative namcouple: place it in
                # prism/util/running/adjunct_files/oasis3
                # and append the namcouple version to its name:
                # namcouple_'cplmod''ncplvers'
run_mode=concurrent # sequential / concurrent (=parallel)
# remapping parameters
scripwr=0      # writing of SCRIP remapping matrices
                # 0: use SCRIP matrice if existing; else (re)calculation
                # 1: unconditional (re)calculation of SCRIP matrices
gridswr=0     # writing of grid description files for OASIS
                # 0: use grid descript. files if existing; else (re)generation
                # 1: unconditional (re)generation of grid description files
extrapwr=1    # writing of extrapolation matrix (NINENN)
                # 0: use of existing extrapolation matrix (error if not
                # available)
                # 1: unconditional (re)generation of the extrapolation matrix
                # Note: if gridswr=1 extrapwr needs to be 1 too
# time interval of data exchange
dto2a=86400    # coupling time step from ocean to atmosphere (86400/43200) [s]
                # Note: coupling time step from atmosphere to ocean is 86400 s
# treatment of coupling fields (see OASIS documentation for more information)
timtranso2a=AVERAGE # INSTANT / AVERAGE
export=EXPORTED # EXPORTED / EXPOUT
# number of processors
integer nproca_atm nprocb_atm nproma_atm nthreadatm nproca_oce nprocb_oce nthreadoce
nproca_atm=8    # (8 is default on tornado / =5 for T63 on hurrikan)
nprocb_atm=3    # (3 is default on tornado / =1 on hurrikan) total number of MPI processors for the atmosphere
nproma_atm=64   # (64 is default on tornado / =1024 on hurrikan / =3840 for T63) vector length (http://svn.zmaw
nthreadatm=1    # (1 is default) number of openMP threads for the atmosphere model
nproca_oce=3    # (3 is default on tornado / =1 on hurrikan)
nprocb_oce=9    # (9 is default on tornado / =1 on hurrikan / =3 for T63) total number of MPI processors for the
nthreadoce=1    # (1 is default on tornado / =3 on hurrikan) number of openMP threads for the ocean model
#-----
# 1.2 TASK SPECIFICATION
#-----
#
#-- preprocessing: yes: create and run a script for preprocessing
#                  no: no preprocessing
preprocessing=no
#
#-- postprocessing: yes: create and run a script for postprocessing
#                  no: no postprocessing
postprocessing=yes
#
#-- splitting:     yes: split multi code output in codes (only possible, if postprocessing=yes)
#                  no: no splitting of codes
splitting=no
#
#-- dbfill:        yes: fill database according code lists (only possible, if postprocessing=yes and splitting=yes)
#                  no: no data base filling

```

```

dbfill=no
#
#-- monitoring :      yes: create and run a script for monitoring
#                   no:  no monitoring
monitoring=yes
#
#-- archiving:      yes: create and run a script for archiving
#                   no:  no archiving
archiving=no
#-----
# 1.3 TIME CONTROL
#-----
#-- declarations
integer nyear nmonth nday nhour nminute nsecond
#
#-- calendar type: Available calendar options:
# 0 : No leap year (365 days per year)
# 1 : Gregorian (365/366 days per year)
# n : Equal months of "n" days (30 for 30 day months)
caltype=1
#
#-- initial and final date of the experiment
# Format: YearMMDD[_hh[mm[ss]]], Year-MM-DD[_hh[:mm[:ss]]] or
#        Year-MM-DD[Thh[:mm[:ss]]]
# Note: The experiment will not stop within a run/chunk even if the
#       final date is reached.
initial_date=0800-01-01 # initial date of the experiment
final_date=0803-12-31  # final date of the experiment
#
#-- duration of a run/chunk
# Specify the length of each run in one of the below units.
nyear=1      # number of years per run
nmonth=0     # number of months per run
nday=0       # number of days per run
nhour=0      # number of hours per run
nminute=0    # number of minutes per run
nsecond=0    # number of seconds per run
nstep_atm=0  # number of atmosphere model time steps per run
nstep_oce=0  # number of ocean model time steps per run
#-----
# 1.4 ARCHIVE
#-----
#-- use_initial_tarfile - Get initial data from a tar file
# yes: get initial data tar file from 'archive_in'
# no:  all initial data is in place (short term archive 'data')
use_initial_tarfile=yes
#
#-- tag to distinguish between different input data files
#
tag=""
#
#-- file and directory permissions of the output
#
export dir_permits=755
export file_permits=644
#-- fill_archive_in - Store data from the tarfile as single files in
#                   'archive_in'. This allows for usage with different experiments
#                   (only with use_initial_tarfile=yes)
# yes: the data shall be stored as single files in 'archive_in'
# no:  'archive_in' is not used
fill_archive_in=no
suppress_links=yes
#
#-- archiving_onlyraw
# yes: archive only raw files (default)
# no:  archive as well postprocessed files
archiving_onlyraw=yes
#
#-- archiving_host - Node name of an archiving machine (if different from the
#                   compute or data processing platform )
# export archiving_host="cross.dkrz.de" setting if you run your model at on cross, but data processing is performed
# export archiving_host="gridftp.dkrz.de" setting for running on tornado and archiving by gridftp on the DKRZ
export archiving_host=""
#-----
# 1.5 MESSAGE PASSING
#-----
#-- message passing library with which the models and the coupler were compiled
# (MPI1 or MPI2) and shall be launched (mpich/mpich2/openmpi/"")
mpi_library=openmpi
#
#-- launching mode (spawned by coupler: MPI2; MPI1 otherwise)
#
message_passing=MPI1
#
#-- buffered MPI Send
# yes: buffered send
# no:  simple send
bsend=no
#####
#
# MPIBIN according to site and compiler for calls of mpiexec/mpirun

```

```

#
#-- name of compiler (set according to Create_TASKS parameter specifications)
#
compiler=intel
case "${compiler}" in
  pgi)
MPIBIN=/sw/sles9-x64/ofed/openmpi-1.2.5-pgi7/bin
;;
  nag)
MPIBIN=/sw/sles9-x64/ofed/openmpi-1.2.5-nag/bin
;;
  sun)
  #MPIBIN=/opt/ofed/openmpi-1.2.5-sun12/bin          # GKSS
  #MPIBIN=/opt/openmpi-1.2.8/sun/bin                # DWD
MPIBIN=/sw/sles9-x64/ofed/openmpi-1.2.5-sun12/bin  # ZMAW
;;
  intel)
  MPIBIN=/sw/sles9-x64/ofed/openmpi-1.2.5-intel/bin
;;
  path)
  #MPIBIN=/opt/scali/bin                            # GKSS
  MPIBIN=/sw/sles9-x64/ofed/openmpi-1.2.5-path/bin
;;
*)
echo Compiler $compiler not supported.
exit 1
esac
#
#-- number of processors used for the mpiexec
#
nprocmpi=0
#-----
# 1.6 FILE SYSTEMS
#-----
#
#-- home: Permanent file system for the SCRIPTS on the COMPUTING HOST
# (only needs to be specified if the tasks are NOT generated on the
# computing host)
export home=/scratch/work/bm0021/k204019/cosmos-dev/experiments
#
#-- archive_in: Root directory of the LONG TERM INPUT data archive. It needs
# to reside on the same machine as the output archive. This
# archive is intended for input data that is needed with
# several experiments, e.g. initial , forcing or restart files.
# - The parent-directory of ${archive_in} needs to exist before submission of the job-
export archive_in=/pool/data/COSMOS/cosmos2
#
#-- data: Root directory of the SHORT TERM data server.
# Model INPUT and OUTPUT will be read from/written to
# this file system of the computing host
# - The parent-directory of ${data} needs to exist before submission of the job
export data=/scratch/wrkshr/k204019/cosmos-dev/experiments
#
#-- archive: Root directory of the LONG TERM OUTPUT data archive.
# - Either a filesystem of the computing host or of a remote archiving host.
# If ${archive} differs from ${data} model output will be saved in
# ${archive} and removed from ${data}.
# - The parent-directory of ${archive} needs to exist -
export archive=${PRJHOME}/arch/k204019/cosmos-dev/experiments
#
#-- work: Root directory for the temporary working directory
# (for production runs use $TMPDIR on NEC)
#
work=/scratch/wrkshr/k204019/cosmos-dev/experiments
#
#-- Compilation
#
# compile_server: Node name of the compile-server
# compile_path: Directory where the executables are stored
# on the compile-server
compile_server=linux-x64
compile_path=/scratch/work/bm0021/k204019/cosmos-dev/x86_64-intel/bin
#
#-- Path to the IMDI function directory
#
export fpath=/scratch/work/bm0021/k204019/cosmos-dev/util/running/functions
export PATH=${fpath}:$PATH
#-----
# 1.7 RESTART CONTROL
#-----
#
#-- 'component'_restart: start from restart or initial files (climatology)
# 1 : start experiment from restart files for 'component'
# 0 : start experiment from initial conditions for 'component'
# If the experiment starts from restart files you need to specify:
#
# 'component'_age: the age of the restart file used in years
# 'component'_restart_file: filename of the restart file (including path)
#
atm_restart=1
atm_age=0
atm_restart_file=${data}/${expid}/restart/echam5/rerun_echam.nc

```

```

atm_restart_tracer=${data}/${expid}/restart/echam5/rerun_tracer.nc
atm_restart_co2=${data}/${expid}/restart/echam5/rerun_co2.nc
atm_restart_accu=${data}/${expid}/restart/echam5/rerun_accuflx.nc
hd_restart_file=${data}/${expid}/restart/echam5/hdrestart.nc
srf_restart=1
srf_age=0
srf_restart_jsbach=${data}/${expid}/restart/jsbach/rerun_jsbach.nc
srf_restart_surf=${data}/${expid}/restart/jsbach/rerun_surf.nc
srf_restart_veg=${data}/${expid}/restart/jsbach/rerun_veg.nc
oce_restart=1
oce_age=0
oce_restart_file=${data}/${expid}/restart/mpiom/rerun_mpiom.ext
bgc_restart=1
bgc_age=0
bgc_restart_file=${data}/${expid}/restart/hamocc/rerun_hamocc.nc
cpl_restart=1
cpl_restart_file_sstocan=${data}/${expid}/restart/oasis3/sstocan.nc
cpl_restart_file_flxatmos=${data}/${expid}/restart/oasis3/flxatmos.nc
#-----
# 1.8 PLATFORM DEPENDEND SPECIFICATIONS
#-----
#-- set system stuff required:
#
if [ "$(sitename)" = "dkrz.de" ]; then
  if [ "${MODULESHOME:-}" != "" ]; then
    . ${MODULESHOME}/init/ksh
    module load SunGridEngine/6.0u10 globus/4.0.3
  else
    echo "ERROR: module command not available."
    exit 1
  fi
fi
#
#-- batch queueing system ( PBS | SGE | LL | NQS2 | LSF )
#
#queueing_system=PBS ; n_nodes=24 # total number of nodes
queueing_system=SGE
#
#-- email address (for queueing system)
#
email=your@email
#
#-- account (for queueing system)
#
account=default # DKRZ account number ("default" for default account)
#
#-- queue (for queueing system)
#
queue=default # queue name ("default" for default queue)
#-----
# 1.9 UNIX COMMANDS
#-----
export mkdir="mkdir -p" # create a new directory
export cp="cp -p" # copy without changing the time stamp
export ln="ln -sf" # soft link (if no links are wanted: same as cp)
export rm="rm" # remove
export rtp=gridftp # remote transfer protocol
export rtp_post=$rtp # transfer protocol to remote processing host
export put_archive="" # special command to put files to band archive (e.g. dsmc)
export get_archive="" # special command to get files from band archive (e.g. dsmc)
export gunzip="gzip -d" # unzip a file that was zipped using gzip
export job_account="" # command to receive job account at the end of the run
export cdo=/client/bin/cdo # climate data operator
export python=/client/bin/python # python
#-----
# 1.10 REMOTE DATA PROCESSING
#-----
#
#-- Perform data processing on remote host: yes/no
#
postprocessing_rem=no # yes/no
monitoring_rem=no # yes/no
dbfill_rem=no # yes/no
archiving_rem=no # yes/no
#
#-- data_rem: Location, where data processing (postprocessing etc.) should be performed.
# It is specified as [processing_host:][processing_path], e.g.
#
# data_rem=$data # default, i.e. processing in the directory
# ($data) on the compute (frontend) host
# data_rem=mil00.zmaw.de:/mil00 # TPP: processing on host mil00.zmaw.de
# in the working directory /mil00
data_rem=$data
[[ $(print ${data_rem} | grep :) = "" ]] && host_rem="" || host_rem=${data_rem%%:*}
path_rem=${data_rem#*:}
#
#-- qsub_rem: Submit command for the processing jobs on remote host
#
# qsub_rem=$qsub # default, i.e. processing jobs are submitted
# with the same command as the run job ($qsub)
# qsub_rem=sge_qsub # TPP
# qsub_rem='ssh [-l user] nohup' # processing on remote host without
# queueing system (interactive)

```

```

qsub_rem=$qsub
qsub_rem_sync=$qsub_rem' -sync y'          # wait for the job to complete before exiting
#
#-- Path to the IMDI function directory on remote processing host
#
fpath_rem=${path_rem}/${expid}/functions
[[ "${fpath}" = "${fpath_rem}" ]] || export PATH=$PATH:${fpath_rem}
#
#-- Submit host (i.e. compute or frontend node)
#
submit_host=cross.dkrz.de
#
#-- chron_proc: Data processing in chronological order: yes/no
#
chron_proc=yes
#####
#           END OF THE USER INTERFACE
#####
set -e
#-- Number of openMP threads
#
export MPIOM_THREADS=${nthredoce}
export ECHAM5_THREADS=${nthreadatm}
export MPIEXPORT="MPIOM_THREADS ECHAM5_THREADS"
#-----
# Complete setup of COSMOS (parameters wich cannot be changed)
#-----
# declarations
#
integer ntproc nprocatm nprococo nprocmpi ncplprocatm ncplprococo
#
# treatment of coupling fields
#
timtransa2o=INSTANT
#
# Exchange time steps
#
dta2o=86400
dto2a=${dto2a}
# Parameters for ECHAM5
# -----
#-- Time step
#
if [ ${vres_atm} = 19 ]; then
  if [ ${res_atm} = T21 ]; then
    nadt=2400
  elif [ ${res_atm} = T31 ]; then
    nadt=2400
  elif [ ${res_atm} = T42 ]; then
    nadt=1800
  elif [ ${res_atm} = T63 ]; then
    nadt=1200
  elif [ ${res_atm} = T85 ]; then
    nadt=900
  elif [ ${res_atm} = T106 ]; then
    nadt=720
  fi
elif [ ${vres_atm} = 31 ]; then
  if [ ${res_atm} = T31 ]; then
    nadt=1800
  elif [ ${res_atm} = T42 ]; then
    nadt=1200
  elif [ ${res_atm} = T63 ]; then
    nadt=720
  elif [ ${res_atm} = T85 ]; then
    nadt=480
  elif [ ${res_atm} = T106 ]; then
    nadt=360
  elif [ ${res_atm} = T159 ]; then
    nadt=240
  fi
fi
# Parameters for JSBACH
# -----
res_srf=${res_atm}
# Parameters for MPIOM
# -----
#-- Time step and grid dimensions
#
if [ ${res_oco} = GR30 ]; then
  nx_oco=122
  ny_oco=101
  nodt=8640
elif [ ${res_oco} = GR15 ]; then
  nx_oco=256
  ny_oco=220
  nodt=4320
elif [ ${res_oco} = T43 ]; then
  nx_oco=130
  ny_oco=211
  nodt=4800
echo " TIMESTEP NEEDS TO BE CHECKED!"

```

```

fi
# Parameters for HAMOCC
# -----
res_bgc=${res_oce}
nbdtd=${nodt}
#
# Number of MPI-processors/openMP-threads
# -----
ncplprococce=1 # number of ocean MPI processes communicating with oasis
ncplprocatm=1 # number of atmosphere MPI processes communicating with oasis
nprocatm=nproca_atm+nprocb_atm # total number of atm. MPI processes
nprococce=nproca_oce+nprocb_oce # total number of ocean MPI processes
# total number of MPI processes
ntproc=nprocatm+nprococce+nprocmpl+1
# -----
#
# Directory and name of this script
# -----
if [ "$queueing_system" ]; then
  qsub=$(submit_command -q ${queueing_system})
fi
if [ "${qsub}" ]; then
  jobdir=${home}/${expid}/scripts # script directory
  print " - submitted by \t${queueing_system}"
else # interactive
  jobdir=$(dirname $0)
  cd ${jobdir}
  jobdir=$(pwd) # script directory
  print " - submitted \tinteractively"
fi
job=$(job_name -q $queueing_system -s $(basename $0)) # script name
jobid=$(job_identifier -q ${queueing_system} -e ${expid}) # job-id
print " - job name \t${job}"
print " - job id \t${jobid}\n - job directory \t${jobdir}\n"
# -----
#
# 3. CALENDAR
# -----
#-- calculate length of the run in seconds for the case that (optionally)
# the length of run is given in number of model steps of any of the models.
#
if [ "${nstep_atm}" -ne 0 ] && [ "${nstep_atm}" -ne "" ]; then
  (( nsecond = nstep_atm * nadtd ))
elif [ "${nstep_oce}" -ne 0 ] && [ "${nstep_oce}" -ne "" ]; then
  (( nsecond = nstep_oce * nodtd ))
elif [ "${nstep_che}" -ne 0 ] && [ "${nstep_che}" -ne "" ]; then
  (( nsecond = nstep_che * ncdtd ))
elif [ "${nstep_srf}" -ne 0 ] && [ "${nstep_srf}" -ne "" ]; then
  (( nsecond = nstep_srf * nsdtd ))
fi
#-- find out smallest time unit in inidate and job length
#
inidate=`format_date -- ${initial_date}` # transform to format (YearMMDD_hhmmss)
findate=`format_date -- ${final_date}`
nwords=$(format_date -f4 -- ${inidate} | wc -w)
if [ ${nwords} -eq 6 ] || [ ${nsecond} -ne 0 ]; then
  inidate=$(format_date -s -- ${inidate})
  findate=$(format_date -s -- ${final_date})
elif [ ${nwords} -eq 5 ] || [ ${nminute} -ne 0 ]; then
  inidate=$(format_date -m -- ${inidate})
  findate=$(format_date -m -- ${final_date})
elif [ ${nwords} -eq 4 ] || [ ${nhour} -ne 0 ]; then
  inidate=$(format_date -h -- ${inidate})
  findate=$(format_date -h -- ${final_date})
fi
#-- date of this run
#
cd ${jobdir}
space_error="no"
datefmt='%a %b %d %H:%M:%S %Z %Y' # date format for expid.log file
if [ ! -f ${expid}.date ]; then
  startdate=${inidate}
  jobnum=1
  if [ -f ${expid}.log ]; then
    rm ${expid}.log
  fi
  print "$(date +"${datefmt}") : Beginning of Experiment ${expid}" > ${expid}.log.new || {
    space_error="yes"; print "Could not create ${expid}.log";
  }
else
  read startdate jobnum < ${expid}.date
  cp ${expid}.log ${expid}.log.new || {
    space_error="yes"; print "Could not save ${expid}.log";
  }
fi
print "$(date +"${datefmt}") : ${jobnum} ${startdate} ${jobid} - start" >> ${expid}.log.new || {
  space_error="yes"; print "Could not append to ${expid}.log";
}
if [ "${space_error}" = "no" ]; then

```

```

mv ${expid}.log.new ${expid}.log
else
  print " |- ERROR: No disk space left or quota exceeded?"
  exit 1
fi
integer scrcap
line=$(df -k $data | tail -1)
scrfs=${line##* }
line=${line%%\%*} ;scrcap=${line##* }
if (( scrcap > 99 )); then
  print " |- ERROR: Less than 5% disc space left on filesystem $scrfs, where your"
  print " | workshare data=$data is mounted. Please clean up before you continue !"
  exit 1
fi
nextdate=$(calc_date plus -c${caltype} -Y${nyear} -M${nmonth} -D${nday} -h${nhour} -m${nminute} -s${nsecond} -)
print " |+ Time integration and run periode"
print " |- Initial date of the experiment:\t${inidate}"
print " |- Final date of the experiment:\t${findate}"
print " |- Beginning of this run : \t${startdate}"
print " |- Beginning of the next run:\t${nextdate}\n"
#-----
#
# Definition of the functions
#-----
. function_check_size
. function_check_codes
. function_dbfill
. function_generate_tarfile
. function_get_file
. function_get_model_resolution
. function_get_tarfile
. function_make_directories
. function_make_ppdirectories
. function_prepare_saving
. function_put_file
. function_plot_file
. function_pperror
#-----
#
# 4. PRE PROCESSING
#-----
#
# 4.1 DIRECTORY DEFINITION
#-----
if [ "${hostname}" = "${host_rem%.*}" ] ; then
  exphome=${path_rem}/${expid} # Root directory of the experiment (data)
else
  exphome=${data}/${expid} # Root directory of the experiment (data)
fi
export bindir=${exphome}/bin # Directory of the executables
export inpdire=${exphome}/input # Directory of the input files
export restdir=${exphome}/restart # Directory of the restart files
export outdir=${exphome}/outdata # Directory of the output data files
export logdir=${exphome}/log # Directory of the log data files
export postdir=${exphome}/post # Directory for postprocessed data
if [ ${jobnum} = 1 ];then
  if [ "${task}" = "RUN" ]; then
    make_directories
  elif [ "${task}" = "REM" ]; then
    make_ppdirectories
  fi
fi
#-----
#
# save log file of the previous run
#-----
#
# find out the id of the last run
#
if [ ${jobnum} != 1 ]; then
  loginfo=$(get_logpid -d ${startdate} -f ${jobdir}/${expid}.log)
  previd=${loginfo%[ ]*}
  if [ "${previd}" = "${expid}" ]; then
    echo "No logfile of the previous run available (interactive run)"
  else
    date=${loginfo#[ ]}
    logfile=${job}.o${previd}
    if [ ! -f ${jobdir}/${logfile} ] && [ ! -f ${logdir}/${job}_${date}.o${previd} ]; then
      echo "waiting for the log file of the previous run: ${logfile} ..."
      sleep 300
    fi
  fi
  if [ -f ${jobdir}/${logfile} ]; then
    mv ${jobdir}/${logfile} ${logdir}/${job}_${date}.o${previd}
    if [ -f ${jobdir}/${job}.po${previd} ]; then # remove SGE stdout
      rm ${jobdir}/${job}.po${previd}
    fi
  fi
  elif [ ! -f ${logdir}/${job}_${date}.o${previd} ]; then
    echo "No logfile of the previous run - exit"
    exit 1
  fi
fi
fi

```

```

-----
# 4.3 PRE - PROCESSING : Get the input data tar-file
-----
if [ ${jobnum} = 1 ]; then
  get_tarfile ${use_initial_tarfile}
fi
# 4.4 PRE - PROCESSING : Computing environment
-----
# create and go to the temporary working directory
if [ "${work}" = "" ]; then
  print "\n |- ERROR: Can't create the temporary working directory"
  print " | Variable 'work' is empty"
  exit
fi
cd ${work}
if [ ! -d ${expid} ]; then
  ${mkdir} ${expid}
fi
if [ ! -d ${expid}/work ]; then
  ${mkdir} ${expid}/work
fi
cd ${expid}/work
rm -rf *
print "\n |- Temporary working directory is:\t$(pwd)"
print " |- Data workshare on compute node is:\t\data/${expid}"
-----
# 4.5 PRE - PROCESSING : Provide the executables
-----
print " |+ Get executables"
oceexec=${ocemod}_${bgcmmod}_${ocevers}.${message_passing}.x
atmexec=echam5j_${atmvers}.${message_passing}.x
cplexec=${coupler}.${message_passing}.x
#
# remove old executables to get them anew from the compiling host
#
if [ ${jobnum} = 1 ]; then
  [ ! -f ${bindir}/${oceexec} ] || rm ${bindir}/${oceexec}
  [ ! -f ${bindir}/${atmexec} ] || rm ${bindir}/${atmexec}
  [ ! -f ${bindir}/${cplexec} ] || rm ${bindir}/${cplexec}
fi
get_file ${ocemod} bin ${oceexec} mpi-om
get_file ${atmmod} bin ${atmexec} echam5
get_file ${coupler} bin ${cplexec} oasis.x
-----
# 4.6 PRE - PROCESSING : Provide the input data
-----
echo " |- Get input and restart data\n"
# Input files for the coupler (OASIS3)
#
# List of variable names according to cf conventions
#
get_file ${coupler} input cf_name_table.txt
# Grid and analysis auxiliary data files
#
FV_cpl=frac
if [ $gridswr = 0 ] || [ $jobnum != 1 ]; then
  get_file -opt ${coupler} input grids_${res_atm}_${res_oce}${FV_cpl}.nc \
  grids.nc
  get_file -opt ${coupler} input areas_${res_atm}_${res_oce}${FV_cpl}.nc \
  areas.nc
  get_file -opt ${coupler} input masks_${res_atm}_${res_oce}${FV_cpl}.nc \
  masks.nc
fi
if [ $scripwr = 0 ] || [ $jobnum != 1 ]; then
  get_file -opt ${coupler} input \
  rmp_oces_to_atmo_CONSERV_FRACAREA_${res_atm}_${res_oce}.nc \
  rmp_oces_to_atmo_CONSERV_FRACAREA.nc
  get_file -opt ${coupler} input \
  rmp_atmo_to_oces_CONSERV_FRACAREA_${res_atm}_${res_oce}.nc \
  rmp_atmo_to_oces_CONSERV_FRACAREA.nc
  get_file -opt ${coupler} input \
  rmp_atmo_to_oceu_CONSERV_FRACAREA_${res_atm}_${res_oce}.nc \
  rmp_atmo_to_oceu_CONSERV_FRACAREA.nc
  get_file -opt ${coupler} input \
  rmp_atmo_to_ocv_CONSERV_FRACAREA_${res_atm}_${res_oce}.nc \
  rmp_atmo_to_ocv_CONSERV_FRACAREA.nc
  get_file -opt ${coupler} input \
  rmp_atmo_to_oces_BILINEA_${res_atm}_${res_oce}.nc \
  rmp_atmo_to_oces_BILINEA.nc
  get_file -opt ${coupler} input \
  rmp_atml_to_oces_CONSERV_FRACAREA_${res_atm}_${res_oce}.nc \
  rmp_atml_to_oces_CONSERV_FRACAREA.nc
fi
if [ $extrapwr = 0 ] || [ $jobnum != 1 ]; then
  get_file ${coupler} input nweights_${res_atm}_${res_oce}${FV_cpl} nweights
fi
# Restart files
#
-----
prevdate=`calc_date minus -c${caltype} -D1 -- ${startdate}`
if [ ${jobnum} = 1 ]; then
  if [ ${cpl_restart} = 1 ]; then

```

```

    if [ ${run_mode} = concurrent ]; then
        \cp ${cpl_restart_file_flxatmos} flxatmos
    fi
    \cp ${cpl_restart_file_sstocean} sstocean
fi
else
    if [ ${run_mode} = concurrent ]; then
        get_file ${coupler} restart flxatmos_${expid}_${prevdate}.nc flxatmos
    fi
    get_file ${coupler} restart sstocean_${expid}_${prevdate}.nc sstocean
fi
#-----
# Input files for the atmosphere (ECHAM5)
#-----
#
#-- Initialization and forcing
#-----
#
# 3d initial file of atmosphere (spectral, no dependence on lsmask)
get_file ${atmmod} input ${res_atm}L${vres_atm}_jan_spec.nc unit.23
#
# surface boundary conditions (land/sea mask, albedo etc.)
# (the original file has been modified near Antarctica (M.Esch);
# annual mean data); file depends on lsmask
get_file ${atmmod} input ${res_atm}${res_oce}_jan_surf.nc unit.24
#
# AMIP sst and sea ice concentration; files depend on lsmask
#
if [ "${ocemod}" = "" ] && [ ${forcing} = amip ]; then
    typeset -Z4 yr ypl yml
    yr='format_date -f4 -- ${startdate} | cut -f1 -d" "'
    (( ypl = yr + 1 ))
    (( yml = yr - 1 ))
    for yrs in ${yr} ${ypl} ${yml}; do
        get_file ${atmmod} input ${res_atm}_amip2sst_${yrs}.nc sst${yrs}
        get_file ${atmmod} input ${res_atm}_amip2sic_${yrs}.nc ice${yrs}
    done
else
    get_file ${atmmod} input ${res_atm}_amip2sst_clim.nc unit.20
    get_file ${atmmod} input ${res_atm}_amip2sic_clim.nc unit.96
fi
#
# ozone climatology (m.m., zonal) (no dependence on lsmask)
get_file ${atmmod} input ${res_atm}_O3clim2.nc unit.21
#
# leaf area index climatology (monthly)
#
get_file ${atmmod} input ${res_atm}${res_oce}_VLTCLIM.nc unit.90
#
# 3d vegetation climatology (a.m.) (monthly)
#
get_file ${atmmod} input ${res_atm}${res_oce}_VGRATCLIM.nc unit.91
#
# land surface temperature climatology (monthly)
#
get_file ${atmmod} input ${res_atm}_TSLCLIM2.nc unit.92
#
# input data for radiation scheme
#
get_file ${atmmod} input surrta_data rrtadata
#
# input data for the hydrological discharge model
#
if [ $lhd = yes ]; then
    get_file ${atmmod} input hdpara.nc hdpara.nc
    get_file ${atmmod} input hdstart.nc hdstart.nc
fi
#
# input data for volcanic forcing
#
if [ ${volc_forc} = "true" ]; then
    get_file ${atmmod} input volc_data volc_data
    get_file ${atmmod} input rad_table rad_table
    cp ${jobdir}/volc_data volc_data
    cp ${jobdir}/rad_table rad_table
fi
#
#-- Restart files
#-----
if [ ${jobnum} != 1 ]; then
    prevdate='calc_date minus -c${caltype} -D1 -- ${startdate}'
    get_file ${atmmod} restart rerun_${expid}_echam_${prevdate} \
        rerun_${expid}_echam
    if [ $lhd = yes ]; then
        get_file ${atmmod} restart hdrestart_${expid}_${prevdate}.nc \
            hdrestart.nc
    fi
    if [ "${save_dblrad}" = "true" ]; then
        get_file ${atmmod} restart rerun_${expid}_accuflx_${prevdate} \
            rerun_${expid}_accuflx
    fi
    if [ "${srfmod}" = "jsbach" ]; then
        get_file ${atmmod} restart rerun_${expid}_co2_${prevdate} \
            rerun_${expid}_co2
    fi
fi

```

```

fi
if [ "${co2_transport}" = "true" ]; then
  get_file ${atmmod} restart rerun_${expid}_tracer_${prevdate} \
    rerun_${expid}_tracer
fi
elif [ ${atm_restart} = 1 ] ; then
  \cp ${atm_restart_file} rerun_${expid}_echam
  if [ "${srfmod}" = "jsbach" ]; then
    \cp ${atm_restart_co2} rerun_${expid}_co2
  fi
  if [ $lhd = yes ]; then
    \cp ${hd_restart_file} hdrestart.nc
  fi
  if [ "${co2_transport}" = "true" ]; then
    \cp ${atm_restart_tracer} rerun_${expid}_tracer
  fi
  if [ "${save_dblrad}" = "true" ]; then
    \cp ${atm_restart_accu} rerun_${expid}_accuflx
  fi
fi
#-----
# Input files for the land surface model (JSBACH)
#-----
#
# input for JSBACH
#
if [[ "${ocemod}" = "" && ${alone_as_coupled} = false ]]; then
  grid=${res_srf}_${ntiles}tiles
else
  grid=${res_srf}_${res_oce}_${ntiles}tiles
fi
[[ ${lctlibvers} != "" ]] || lctlibvers=albedo_snow
get_file ${srfmod} input lctlib_${lctlibvers}.def lctlib.def
get_file ${srfmod} input jsbach_${grid}_${refyear}${jsbinivers}.nc jsbach_${res_srf}.nc
if [[ ${read_cpools} = true ]]; then
  get_file ${srfmod} input Cpools_${grid}_${refyear}.nc Cpools.nc
fi
if [[ ${land_use} = true ]]; then
  get_file ${srfmod} input ${inpdirex}/jsbach/land_use/cover_fract_`echo ${startdate} | cut -c1-4`_${res_srf}_${ntiles}tiles.
  cover_fract.`echo ${startdate} | cut -c1-4`.nc
fi
if [ "${jsb_standalone}" = "true" ]; then
  get_file ${srfmod} input Climate.${res_srf}.nc climate.${res_srf}.nc
  get_file ${srfmod} input mpot.${res_srf}.nc mpot.${res_srf}.nc
fi
#-- Restart files
#
prevdate=`calc_date minus -c${caltype} -D1 -- ${startdate}`
if [ ${jobnum} != 1 ]; then
  get_file ${srfmod} restart rerun_${expid}_jsbach_${prevdate} \
    rerun_${expid}_jsbach
  get_file ${srfmod} restart rerun_${expid}_veg_${prevdate} \
    rerun_${expid}_veg
  if [ ${jsb_standalone} = true ]; then
    get_file ${srfmod} restart rerun_${expid}_forcing_${prevdate} \
      rerun_${expid}_forcing
    get_file ${srfmod} restart rerun_${expid}_driving_${prevdate} \
      rerun_${expid}_driving
  else
    get_file ${srfmod} restart rerun_${expid}_surf_${prevdate} \
      rerun_${expid}_surf
  fi
elif [ ${srf_restart} = 1 ] ; then
  \cp ${srf_restart_jsbach} rerun_${expid}_jsbach
  \cp ${srf_restart_veg} rerun_${expid}_veg
  if [ ${jsb_standalone} = true ]; then
    \cp ${srf_restart_forcing} rerun_${expid}_forcing
    \cp ${srf_restart_driving} rerun_${expid}_driving
  else
    \cp ${srf_restart_surf} rerun_${expid}_surf
  fi
fi
#-----
# Input files for the ocean (MPIOM)
#-----
#-- Initialization and forcing
#-----
#
# File containing basin masks (formatted)
#
get_file ${ocemod} input ${res_oce}_BEK BEK
chmod ${dir_permits} BEK
#
# Ocean grid and bathymetry file
#
get_file ${ocemod} input ${res_oce}_anta anta
get_file ${ocemod} input ${res_oce}_arcgri arcgri
get_file ${ocemod} input ${res_oce}_topo_jj topo
#
# Surface salinity file (a.m. Levitus atlas).
#
get_file ${ocemod} input ${res_oce}L${vres_oce}_SURSAL_PHC SURSAL
#
# Initial (3D) ocean temperature/salinity file (Levitus a.m.).

```

```

#
get_file ${ocemod} input ${res_oce}L${vres_oce}_INITEM_PHC INITEM
get_file ${ocemod} input ${res_oce}L${vres_oce}_INISAL_PHC INISAL
#
#-- Forcing (for stand alone ocean runs)
#
if [ "${message_passing}" = "NONE" ]; then
  get_file ${ocemod} input ${res_oce}_GICLOUD_OMIP365 GICLOUD
  get_file ${ocemod} input ${res_oce}_GIPREC_OMIP365 GIPREC
  get_file ${ocemod} input ${res_oce}_GISWRAD_OMIP365 GISWRAD
  get_file ${ocemod} input ${res_oce}_GITDEW_OMIP365 GITDEW
  get_file ${ocemod} input ${res_oce}_GITEM_OMIP365 GITEM
  get_file ${ocemod} input ${res_oce}_GIU10_OMIP365 GIU10
  get_file ${ocemod} input ${res_oce}_GIWIX_OMIP365 GIWIX
  get_file ${ocemod} input ${res_oce}_GIWIY_OMIP365 GIWIY
  get_file ${ocemod} input ${res_oce}_GIRIV_OMIP365 GIRIV
fi
#
#-- Restart files
#
if [ ${jobnum} = 1 ]; then
  if [ ${oce_restart} = 1 ]; then
    \cp ${oce_restart_file} Z38000
    \cp ${oce_restart_file} Z37000
  fi
else
  prevdate=`calc_date minus -c${caltype} -D1 -- ${startdate}`
  get_file ${ocemod} restart RESTART_${expid}_${prevdate} Z38000
  get_file ${ocemod} restart RESTART_${expid}_${prevdate} Z37000
fi
#-----
# Input files for the bio-geo-chemistry model HAMOCC
#-----
#-- Get dust fields (Iron for BGC)
#
get_file ${bgcm} input ${res_bgc}_INPDUST.nc INPDUST.nc
#
#-- Restart files
#
if [ ${jobnum} = 1 ]; then
  if [ ${bgc_restart} = 1 ]; then
    \cp ${bgc_restart_file} restarttr_bgc.nc
  fi
else
  get_file ${bgcm} restart restart_bgc_${expid}_${prevdate}.nc restarttr_bgc.nc
fi
#-----
# 4.7 PRE - PROCESSING :
# Provide and update configuration files (namelists, XML, etc.)
#-----
echo "\n |- Provide configuration files (namelists, XML, etc.)"
#-----
#-- Namelist OASIS3 (namcouple)
#
#
# runtime: duration of the experiment (seconds)
#
runtime=`time_between -c${caltype} -- ${startdate} ${nextdate} seconds`
#
# startdate in format YYYYMMDD
#
typeset -Z8 yyyymmdd
yyymmdd=`echo ${startdate#-} | cut -c1-8`
#
# sequential mode and time lag
#
# both models run sequentially when the experiment is started
if [ ${jobnum} = 1 ] && [ ${cpl_restart} = 0 ]; then
  run_mode=sequential
fi
if [ ${run_mode} = sequential ]; then
  nmseq=2 # models run sequentially
  lags2a=$nodt
  lags2o=`expr $nodt - $dta2o`
  iseq=2
else
  nmseq=1 # models run concurrently
  iseq=1
  lags2a=$nodt
  lags2o=$nodt
fi
#
# buffered/simple MPI send
#
if [ ${bsend} = no ]; then
  nobsend="NOBSEND"
else
  nobsend=""
fi
#
# restart filenames for the atmosphere/ocean
#
cnfileaw=flxatmos
cnfileow=sstocean
#
# loctrans
#

```

```

loctrans=LOCTRANS
#
# adaption of the namcouple template
#
get_file -nolink ${coupler} input namcouple_{$cplmod}{$ncplvers} namcouple
[[ $monitoring = yes ]] && monexp="EXPOUT" || monexp="$export"
ed -s namcouple <<EOF
g/#Nmseq/s/#Nmseq/$_{nmseq}/
g/#Chan/s/#Chan/$_{message_passing} $_{nobsend}/
g/#Modlprocs/s/#Modlprocs/ $_{nprocatm} $_{ncplprocatm} $arg1 /
g/#Mod2procs/s/#Mod2procs/ $_{nprococ} $_{ncplprococ} $arg2 /
g/#Cplexptid/s/#Cplexptid/$_{jobname}/
g/#Atmmodnam/s/#Atmmodnam/$_{atmmod}/
g/#Ocemodnam/s/#Ocemodnam/mpi-om/
g/#Runtime/s/#Runtime/$_{runtime}/
g/#Yyyymmdd/s/#Yyyymmdd/$_{yyyymmdd}/
g/#Nlogprt/s/#Nlogprt/$_{nlogprt}/
g/#Dta2o/s/#Dta2o/$_{dta2o}/
g/#Dto2a/s/#Dto2a/$_{dto2a}/
g/#Iseq/s/#Iseq/$_{iseq}/
g/#Laga2o/s/#Laga2o/$_{laga2o}/
g/#Lago2a/s/#Lago2a/$_{lago2a}/
g/#TimTransa2o/s/#TimTransa2o/$_{timtransa2o}/
g/#TimTranso2a/s/#TimTranso2a/$_{timtranso2a}/
g/#Exp/s/#Exp/$_{export}/
g/#Mon/s/#Mon/$_{monexp}/
g/#LocTrans/s/#LocTrans/$_{loctrans}/
g/#Extrapwr/s/#Extrapwr/$_{extrapwr}/
g/#Cnfileaw/s/#Cnfileaw/$_{cnfileaw}/
g/#Cnfileow/s/#Cnfileow/$_{cnfileow}/
w
q
EOF
echo "*" -----
echo "*" Namelist of OASIS3: namcouple"
echo "*" -----
cat namcouple
echo "*" -----
echo "*" end of namcouple"
echo "*" -----
echo "*"
#-----
#-- Namelist ECHAM5
#
#
# resumed or initial run?
#
if [ $_{jobnum} = 1 ] && [ $_{atm_restart} = 0 ]; then
  rerun=.FALSE.
else
  rerun=.TRUE.
fi
#
# coupled or stand-alone run?
#
if [ "${ocemod}" = "" ]; then
# stand alone echam5
  lcouple=".FALSE."
  if [ $_{forcing} = amip ]; then
    lamip=".TRUE."
  else
    lamip=".FALSE."
  fi
else
# echam5 coupled to ocean
  lcouple=".TRUE."
  lamip=".FALSE."
  getoff=0
  putoff=-$_nadt
  na2ocsteps=`expr $dta2o / $_nadt `
  no2acsteps=`expr $dto2a / $_nadt `
  getocean="$_no2acsteps,'steps','exact',$_getoff"
  putocean="$_na2ocsteps,'steps','exact',$_putoff"
fi
#
# initial date of the experiment (with coupled runs: one timestep before
# midnight)
# Initialisation of the echam time manager can take a long time if the current
# date is far from the initial date of the experiment. To improve this, we set
# the year in dt_start just one year before the the current date.
# (Setting dt_start to the current date would lead to echam re-initialization!)
# Note that events that do not occur on a yearly basis will not be treated
# correctly!
if [ $_{lcouple} = ".FALSE." ]; then
  date=`calc_date minus -c$_{caltype} -Y$_{atm_age} -- $_{inidate}`
  year=`format_date -f4 -- $_{inidate} | cut -f1 -d" "`
else
  date=`calc_date minus -c$_{caltype} -Y$_{atm_age} -s$_{nadt} -- $_{inidate}`
  year=`format_date -f4 -- $_{startdate} | cut -f1 -d" "`
  if [ $_{rerun} = .TRUE. ]; then
    (( year = year - 2 ))
  fi
  if [ $_{rerun} = .FALSE. ]; then
    (( year = year - 1 ))
  fi
fi
fi

```

```

month_day_time='format_date -f4 -s -- ${date} | cut -f2- -d" "'
date="${year} ${month_day_time}"
dt_start='echo ${date} | tr " " ,\'
#
# end date of the run
#
dt_stop='format_date -f4 -s -- ${nextdate} | tr " " ,\'
#
# rerun interval
#
if [ ${nmonth} -ne 0 ]; then
  (( nm = 12 * nyear + nmonth ))
  put_rerun="${nm}, 'months', 'last', 0"
elif [ ${nyear} -ne 0 ]; then
  put_rerun="${nyear}, 'years', 'last', 0"
elif [ ${nday} -ne 0 ]; then
  put_rerun="${nday}, 'days', 'last', 0"
fi
#
# usage of the hydrology model (HD)
#
if [ ${lhd} = yes ]; then
  hd=.TRUE.
else
  hd=.FALSE.
fi
#
# output data format
#
if [ ${out_filetype} = GRIB ]; then
  format=1
else
  format=2
fi
#
# usage of 1/0 sea land mask (not 0.5 criteria of fractional mask)
#
if [[ ${res_atm} = T31 && ( ${alone_as_coupled} = true \
  || ${cplmod} = cosmos-aso || ${cplmod} = cosmos-asob ) ]]; then
  lslm=.true.
else
  lslm=.false.
fi
#
# CO2
#
if [ "${co2_transport}" = "true" ] && [ "${srfmod}" = "jsbach" ]; then
  ico2=1 # prognostic CO2 mass mixing ratio
else
  ico2=2 # uniform CO2 volume mixing ratio
fi
#
# Middle atmosphere
#
if [[ ${vres_atm} = 47 ]]; then
  middle_atmosphere=true
fi
echo "&RUNCTL" >> namelist.echam
echo " LRESUME = ${rerun}" >> namelist.echam
echo " out_datapath = './'" >> namelist.echam
echo " out_expname = '${expid}'" >> namelist.echam
echo " out_filetype = ${format}" >> namelist.echam
echo " DT_START = ${dt_start}" >> namelist.echam
echo " DT_STOP = ${dt_stop}" >> namelist.echam
if [[ ${nadt} != default ]]; then
  echo " DELTA_TIME= ${nadt}." >> namelist.echam
fi
echo " PUTDATA = ${dt_write_atm}, 'hours', 'first', 0" >> namelist.echam
echo " PUTRERUN = ${put_rerun}" >> namelist.echam
echo " TRIGFILES = 1, 'months', 'first', 0" >> namelist.echam
echo " LAMIP = ${lamip}" >> namelist.echam
echo " LABORT = .FALSE." >> namelist.echam
echo " NPROCA = ${nproca_atm}" >> namelist.echam
echo " NPROCB = ${nprocb_atm}" >> namelist.echam
echo " NPROMA = ${nproma_atm}" >> namelist.echam
echo " LDEBUGEV = .FALSE." >> namelist.echam
echo " LCOUPLE = ${lcouple}" >> namelist.echam
echo " NO_CYCLES = 1" >> namelist.echam
echo " LSO4 = .FALSE." >> namelist.echam
echo " LHD = ${hd}" >> namelist.echam
if [ "${srfmod}" = "jsbach" ]; then
  echo " LSLM = ${lslm}" >> namelist.echam
fi
if [ "${co2_transport}" = "true" ] && [ "${srfmod}" = "jsbach" ]; then
  echo " LCO2 = .TRUE." >> namelist.echam
fi
if [ ${volc_forc} = "true" ]; then
  echo " L_VOLC = .TRUE." >> namelist.echam
fi
if [ ${lhd} = yes ]; then
  echo " NHD_DIAG = 1" >> namelist.echam
fi
if [ ${lcouple} = ".TRUE." ]; then
  echo " GETOCEAN = ${getocean}" >> namelist.echam
  echo " PUTOCEAN = ${putocean}" >> namelist.echam
fi
if [[ ${alone_as_coupled} = true ]]; then
  echo " LIPCC = .TRUE." >> namelist.echam

```

```

fi
if [[ ${middle_atmosphere} = true ]]; then
  echo "  LMIDATM = .TRUE." >> namelist.echam
fi
echo "/" >> namelist.echam
echo "&RADCTL" >> namelist.echam
if [ ${lcouple} = .TRUE. ] && [ ${res_atm} = T31 ]; then
  echo "  TRIGRAD = 120,'minutes','first',2400" >> namelist.echam
fi
echo "  ICO2 = ${ico2}" >> namelist.echam
if [ "${millennium_ctrl}" = "true" ]; then
  echo "  ICH4=2" >> namelist.echam
  echo "  IO3=3" >> namelist.echam
  echo "  IN20=2" >> namelist.echam
  echo "  ICFC=0" >> namelist.echam
  echo "  IAERO=2" >> namelist.echam
  echo "  CO2VMR=278.E-06" >> namelist.echam
  echo "  CH4VMR=650.E-09" >> namelist.echam
  echo "  N2OVMR=270.E-09" >> namelist.echam
  echo "  YR_PERP=800" >> namelist.echam
fi
if [ ${save_dblrad} = "true" ]; then
  echo "  LDBLRAD = .TRUE." >> namelist.echam
fi
if [[ ${middle_atmosphere} = true ]]; then
  echo "  ICH4=3" >> namelist.echam
  echo "  IN20=3" >> namelist.echam
fi
echo "/" >> namelist.echam
if [[ ${middle_atmosphere} = true ]]; then
  echo "&DYNCTL" >> namelist.echam
  echo "  VCHECK=235." >> namelist.echam
  echo "  SPDRAG=0.926E-4" >> namelist.echam
  echo "/" >> namelist.echam
fi
if [[ ${lcouple} = .TRUE. || ${alone_as_coupled} = true ]]; then
  echo "&PHYSCTL" >> namelist.echam
  echo "  LCOVER = .FALSE." >> namelist.echam
  echo "/" >> namelist.echam
fi
#cat > namelist.echam << EOR
#&RUNCTL
# LRESUME = ${rerun}
# out_datapath = ""
# out_expname = "${expid}"
# out_filetype = ${format}
# DT_STOP = ${dt_stop}
# DELTA_TIME= ${nadt}.
# PUTDATA = ${dt_write_atm},'hours','first',0
# PUTRERUN = 1,'months','last',0
# TRIGFILES = 1,'months','first',0
# LAMIP = ${lamip}
# LABORT = .FALSE.
# NPROCA = ${nproca_atm}
# NPROCB = ${nprocb_atm}
# NPROMA = ${nproma_atm}
# LDEBUGEV = .FALSE.
# LCOUPLE = ${lcouple}
# NO_CYCLES = ${no_cycles}
# LSO4 = .FALSE.
# LHD = ${hd}
# NHD_DIAG = 1
# GETOCEAN = ${getocean}
# PUTOCEAN = ${putocean}
#
#&PHYSCTL
# LCOVER = .FALSE.
#
#EOR
#--- Set output stream properties:
#
echo "&SET_STREAM_ELEMENT name = 'az01' lpost = 0 /" >> namelist.echam
echo "&SET_STREAM_ELEMENT name = 'slm' lpost = 0 /" >> namelist.echam
echo "&SET_STREAM_ELEMENT name = 'glac' lpost = 0 /" >> namelist.echam
echo "&SET_STREAM_ELEMENT name = 'runtoc' lpost = 1 /" >> namelist.echam
echo "&SET_STREAM_ELEMENT name = 'ao3' lpost = 0 /" >> namelist.echam
echo "*" -----
echo "*" Namelist of ECHAM5: namelist.echam"
echo "*" -----
cat namelist.echam
echo "*" -----
echo "*" end of namelist.echam"
echo "*" -----
echo "*"
#-----
#-- Namelist JSBACH
#
if [ "${jsb_standalone}" = "true" ]; then
  lpost_echam=true
else
  lpost_echam=false # Variables of the echam output will not be
                    # printed twice
fi
if [ ${jsb_standalone} = true ]; then
  #
  #-- Start/end date of the experiment
  #
  dt_start='format_date -f4 -s -- ${inidate}'
  date='calc_date minus -s${nsdt} -- ${nextdate}'
  dt_stop='format_date -f4 -s -- ${date}'

```

```

#
# resumed or initial run?
#
if [ ${jobnum} = 1 ] && [ ${srf_restart} = 0 ]; then
    restart=.FALSE.
else
    restart=.TRUE.
fi
fi
#
# Read C-pools from file only at the beginning of an initialized experiment
# To force reading of the initial C-pools during a running experiment (jobnum>1),
# or for a restarted model ( i.e. to overwrite the values from the restart file)
# you need to either
# - set read_cpools=true above and comment out the next three lines, or
# - set read_cpools=true here, irrespective of the first setting above
#
if [ ${jobnum} != 1 ] || [ ${srf_restart} = 1 ] ; then
    read_cpools=false
fi
if [ ${land_use} = true ]; then
    lc_change=EXTERNAL
else
    lc_change=NONE
fi
cat > run.def <<EOF
# ===== Runtime options file for JSBACH =====
# --- General options ---
EXP_NAME=${expid}
DEBUG=false
TEST_STREAM=false
FILE_TYPE=${out_filetype}
OUT_STATE=true
LPOST_ECHAM=${lpost_echam}
# MISS_VAL=1.D20
STANDALONE=${jsb_standalone}
# NPEDIM=1
# USE_NPECHUNKS = {false|true} controls how the JSBACH-Interface is called in SINGLE PROCESSOR runs:
# false: The interface is called for all landpoints, i.e. a single call of the interface per time step
# true : The interface is called several times with at most NPEDIM landpoints. Make sure that the
# driver is designed to handle several calls (otherwise the program may crash!)
USE_NPECHUNKS=false
# --- Number of tiles
N_TILES=${ntiles}
#--- Choice of landsurface scheme ("VIC" or "ECHAM" or "BETHY")
LSS=ECHAM
# --- Handling of landcover change (keyword: LC_CHANGE_HANDLING):
# NONE: run JSBACH with fixed landcover map
# EXTERNAL: run JSBACH using a sequence of landcover maps,
# i.e. a new map is read in at each restart.
# LPJ: Use the LPJ vegetation dynamics
LC_CHANGE_HANDLING=${lc_change}
# --- Operation of Submodel BETHY -----
USE_PHENOLOGY=true
USE_ALBEDO=true
USE_ALBEDOCANOPY=false
USE_BETHY=true
BETHY_NCANOPY=3
BETHYLIB_FILE=lctlib.def
READ_CPOOLS=${read_cpools}
CPOOL_FILE=Cpools.nc
# --- Location of startup files ---
GRID_FILE=jsbach_${res_srf}.nc
SURF_FILE=jsbach_${res_srf}.nc
SOIL_FILE=jsbach_${res_srf}.nc
VEG_FILE=jsbach_${res_srf}.nc
LCTLIB_FILE=lctlib.def
# Soil parameters
SKIN_RES_MAX=0.0002 # Maximum content of skin reservoir for bare soil [m]
INTERCEPTION_MAX=0.0002 # Maximum content of canopy interception [m]
EOF
if [ ${jsb_standalone} = true ]; then
cat >> run.def <<EOF
# --- Parallelization options ---
NCPUS=${nproc_srf}
NPROCA=${nproca_srf}
NPROCB=${nprocb_srf}
# --- Temporal options -----
DT_START=${dt_start}
DT_STOP=${dt_stop}
DELTA_TIME=${nsdt}
# --- Restart options -----
RESTART=${restart}
# --- temperature forcing -----
FORCING_TEMP_FILE=climate.${res_srf}.nc
FORCING_TEMP_FREQU=DAILY # Forcing frequency ("MONTHLY" or "DAILY")
# --- precipitation forcing -----
FORCING_PRECIP_FILE=climate.${res_srf}.nc
FORCING_PRECIP_FREQU=DAILY # Forcing frequency ("MONTHLY" or "DAILY")
# --- longwave radiation forcing -----
#The keyword "FORCING_LW_RADIAT_TYPE" controls what data are used to generate radiation forcing. Possible va
# "CLOUD" : fraction of sky covered by clouds (in percent)
# "MEAN_RAD" : daily means of incident longwave radiation [W/m^2]

```

```

FORCING_LW_RADIAT_TYPE=MEAN_RAD
FORCING_LW_RADIAT_FILE=climate.${res_srf}.nc
FORCING_LW_RADIAT_FREQU=DAILY
# --- shortwave radiation forcing -----
#The keyword "FORCING_SW_RADIAT_TYPE" controls what data are used to generate radiation forcing. Possible va
# "CLOUD"      : fraction of sky covered by clouds (in percent)
# "MEAN_RAD"   : daily or monthly means of incident shortwave radiation [W/m^2]
#              (in this case in addition a table for potential radiation are needed: keywords:
#              "FORCING_TABLE_SW_POT_FILE" and "FORCING_SW_RADIAT_FREQU")
FORCING_SW_RADIAT_TYPE=MEAN_RAD
FORCING_TABLE_SW_POT_FREQU=DAILY
FORCING_TABLE_SW_POT_FILE=mpot.${res_srf}.nc
FORCING_SW_RADIAT_FREQU=DAILY
FORCING_SW_RADIAT_FILE=climate.${res_srf}.nc
#The keyword "FORCING_RADIAT_SCHEME" controls what algorithm is used for obtaining shortwave comonents from
#Possible values are:
# "ORIG"       : The algorithm described in Wolfgangs dissertation is used
# "CCDAS"     : The algorithm currently implemented in CCDAS is used
# "NEW"       : The algorithm newly proposed by Wolfgang is used (only for radiation input "MEAN_RA
# "NEW_ORIG_CF": As "NEW", but with the original conversion factor.
FORCING_SW_RADIAT_SCHEME=ORIG
# --- CO2 forcing -----
FORCING_CO2_FREQU=CONST
FORCING_CO2_CONST_CO2=3.67e-4 # Constant value for CO2-concentration [mol CO2/mol air]
# --- wind speed forcing -----
FORCING_WIND_FREQU=DAILY
FORCING_WIND_FILE=climate.${res_srf}.nc
# FORCING_WIND_CONST_WSPEED=2.
# --- ???? -----
LON_WEST=0.
LON_EAST=360.
LAT_SOUTH=-90.
LAT_NORTH=90.
EOF
fi
echo "*" -----
echo "*" Namelist of JSBACH: run.def"
echo "*" -----
cat run.def
echo "*" -----
echo "*" end of run.def"
echo "*" -----
echo "*"
#-----
#-- Namelist MPIOM
#
if [ ${vres_oce} = 20 ]; then
  dzw="20.,20.,20.,30.,40.,50.,70.,90.,120.,150.
,180.,210.,250.,300.,400.,500.,600.,700.,900.,1400."
elif [ ${vres_oce} = 23 ]; then
  dzw="20.,20.,25.,25.,25.,25.,25.,30.,45.,60.
,90.,120.,150.,180.,210.,250.,300.,400.,500.,600.
,700.,900.,1400."
elif [ ${vres_oce} = 40 ]; then
  dzw="12.,10.,10.,10.,10.,10.,13.,15.,20.,25.
,30.,35.,40.,45.,50.,55.,60.,70.,80.,90.
,100.,110.,120.,130.,140.,150.,170.,180.,190.,200.
,220.,250.,270.,300.,350.,400.,450.,500.,500.,600."
elif [ ${vres_oce} = "" ]; then
  echo 'ERROR: '
  echo '  Vertical resolution of the ocean model not specified!'
  echo ''
else
  echo 'ERROR: '
  echo '  No layerdepth known for vres_oce = ' ${vres_oce}
  echo ''
fi
#
# to allow test runs with just a few days
#
if [ ${nmonth} = 0 ] && [ ${nyear} = 0 ]; then
  nmonts=1
else
  (( nmonts = ${nyear} * 12 + ${nmonth} ))
fi
#
# mean output
#
imean=2
#
# relaxation time for salinity
#
if [ "$coupler" = "" ]; then
  crelsal=3.8E-8
  crelsal=3.E-7
else
  crelsal=0.0
fi
#
# restarted run or start from leviuis
#
if [ ${jobnum} != 1 ] || [ ${oce_restart} = 1 ]; then
  istart=3
else
  istart=2
fi
cat -> OCECTL << EOF
&OCEDIM
IE_G=${nx_oce}
JE_G=${ny_oce}

```

```

/ KE=${vres_oce}
/
&NPROCS
NPROCX=${nproca_oce}
NPROCY=${nprocb_oce}
/
&OCECTL
exptid = "$expid"
DT = $nodt.
CAULAPTS= 0.0000
CAULAPUV= 0.0060
AUS = 0.
CAH00 = 1000.
IBOLK = 500
DVO = 0.2E-2
AVO = 0.2E-2
CWT = 0.5E-3
CWA = 0.75E-3
CSTABEPS= 0.03
DBACK = 1.05E-5
ABACK = 5.E-5
CRELSAL = ${crelsal}
CDVOCON = 0.1
NMONTS = ${nmonths}
IMEAN = ${imean}
ISTART = ${istart}
I3DREST = 0
IOASISFLUX = 0
IMOCDIAG = 1
LDIFFDIAG = .true.
LFORCEDIAG = .false.
LCONVDIAG = .false.
LGRIDINFO = .false.
LHFLDIAG = .false.
LGMDIAG = .true.
ITSDIAG = 2
LTSTRANSPOSE = .true.
/
&OCEDZW
CDZW = ${dzw}
/
EOF
echo "*" -----
echo "*" Namelist of MPIOM: OCECTL"
echo "*" -----
cat OCECTL
echo "*" -----
echo "*" end of OCECTL"
echo "*" -----
echo "*"
#-----
#-- Namelist HAMOCC
#
# For time series:
#           nfreqts1 : frequency of time series sampling
#           nts1 : no. of time series (+1 for global inventory, max=10)
#           rlonts1/rlatts1 : positions of samples in time series1
#
(( steps_per_day = 86400 / nbdt ))
cat > NAMELIST_BGC << EOF
&BGCCTL
deltacalc = 493.57,
deltaorg = 2.457,
deltasil = 0.22,
io_stdo_bgc = 8,
kchck = 0,
isac = 1,
mean_2D_freq = 2,
mean_3D_freq = 2,
nfreqts1 = ${steps_per_day},
rlonts1 = -20.0,60.0,65.0,-64.0,-175.0,-145.0,-25.0,-140.0
rlatts1 = 47.0,17.0,10.0,32.0,-53.0,50.0,64.0,5.0
rdeplts1 = 80.0, 80.0,80.0,80.0,80.0,80.0,80.0,80.0
rdep2ts1 = 1000.0,1000.0,1000.0,200.0,1000.0,1000.0,1000.0,1000.0
rdep3ts1 = 3000.0,3000.0,3000.0,300.0,2000.0,2000.0,2000.0,2000.0
/
&END
EOF
echo "*" -----
echo "*" Namelist of HAMOCC: NAMELIST_BGC"
echo "*" -----
cat NAMELIST_BGC
echo "*" -----
echo "*" end of NAMELIST_BGC"
echo "*" -----
echo "*"
#-----
#
#           5. LAUNCHING THE MODEL
#
#-----
ls -al
date
# Create a test file. The date of output files will be compared to the date
# of this reference file to assure, that the output files are newer.
# (script save_file)
echo "The date of the output files is compared to the date of this file" \
> reference_file
if [ ${compiler} = nag ] || [ ${compiler} = sun ]; then
$cd0 copy anta lb_anta
rm -f anta
mv lb_anta anta
$cd0 copy arcgri lb_arcgri

```

```

rm -f arcgri
mv lb_arcgri arcgri
$cdo copy SURSAL lb_SURSAL
rm -f SURSAL
mv lb_SURSAL SURSAL
$cdo copy INITEM lb_INITEM
rm -f INITEM
mv lb_INITEM INITEM
$cdo copy INISAL lb_INISAL
rm -f INISAL
mv lb_INISAL INISAL
if [ ${jobnum} = 1 ]; then
  $cdo copy Z37000 Z38000
  cp Z38000 Z37000
fi
fi
if [ ${message_passing} = MPI2 ]; then
  echo MPI dynamic process generation not tested.
  exit 1
elif [ ${message_passing} = MPI1 ]; then
cat > ${expid}_asob.$jobid << EOF
#!/bin/ksh
-----
#\ $ -S /bin/ksh
#\ $ -o ${jobdir}/\${JOB_NAME.o}\${JOB_ID}
#\ $ -j y
###HEW-D #\ $ -cwd
#\ $ -pe orte ${ntproc}
-----
#
echo Run in ${work}/${expid}/work
#
cd ${work}/${expid}/work
#
$MPIBIN/mpiexec -n 1 oasis.x : -n ${nprocatm} echam5 : -n ${nprococe} mpi-om || {
  ls -lta
  exit 1
}
#
rm -f ${expid}_asob.$jobid
#
#-----
exit 0
#
#-----
EOF
cat ${expid}_asob.$jobid
# could not use function submit with options so have to to use command
# directly:
qsub -sync y ${expid}_asob.$jobid
else
  echo Invalid message passing method specified !
  exit 1
fi
#
#-- Generate profiling protocol
#
ls -lat
if [ -f *.${ocemod} ]; then
  echo 'Profiling '${ocexec}' ...'
  ${cp} *.${ocemod} ${logdir}/${ocemod}.mon.out
  ${cp} *.${ocemod} mon.out
  prof ${ocemod}
fi
if [ -f *.${atmmod} ]; then
  echo 'Profiling '${atmexec}' ...'
  ${cp} *.${atmmod} ${logdir}/${atmmod}.mon.out
  ${cp} *.${atmmod} mon.out
  prof ${atmmod}
fi
if [ -f *.oasis.x ]; then
  echo 'Profiling '${cplexec}' ...'
  ${cp} *.oasis.x ${logdir}/oasis.mon.out
  ${cp} *.oasis.x mon.out
  prof oasis.x
fi
#-----
#
#      6. POST - PROCESSING: Saving the output data
#
#-----
prepare_saving
saving_error=no
#-----
# Definition of some time variables
#-----
# enddate:      last day of this run
# prevdate:    last day of the previous run
# startyear:   year at the beginning of this run
# prevyear:    year at the last day of the previous run
# startdecade: decade at the beginning of this run
# prevdecade:  decade at the last day of the previous run
# previd:      job-id of the previous run (from expid.log)
# prevstart:   beginning of the previous run (from expid.log)
enddate=$(calc_date minus -c${caltype} -D1 -- ${nextdate})

```

```

prevdate=$(calc_date minus -c${calttype} -D1 -- ${startdate})
startyear=$(format_date -f4 -- ${startdate} | cut -f1 -d" ")
prevyear=$(format_date -f4 -- ${prevdate} | cut -f1 -d" ")
startdecade=${startyear%?}
prevdecade=${prevyear%?}
loginfo=$(get_logpid -d ${startdate} -f ${jobdir}/${expid}.log)
previd=${loginfo%[ ]*}
prevstart=${loginfo#[ ]}
[ ${task} = "RUN" ] && cd ${work}/${expid}/work
#-----
# Save files of the coupler (OASIS3)
#-----
# Output data (EXPOUT)
print "      |- Save output files of the coupler $coupler"
if [ ${export} = "EXPOUT" ] || [ ${monitoring} = "yes" ]; then
  expoutdate=$(format_date -f2 -s -- ${startdate})
  expoutfiles=$(ls *_out.${expoutdate}.nc)
  for file in ${expoutfiles}; do
    save_file ${coupler} output ${file}
  done
fi
# Log files
save_file ${coupler} log cplout cplout_${startdate}_${enddate}
save_file ${coupler} log Oasis.prt Oasis_${startdate}_${enddate}.prt
# Restart files
save_file ${coupler} restart flxatmos flxatmos_${expid}_${enddate}.nc
save_file ${coupler} restart sstocean sstocean_${expid}_${enddate}.nc
if [ ${jobnum} = 1 ]; then
  print "      |- Save input files of the coupler $coupler"
  FV_cpl=frac
  # Grid description files (if just created)
  if [ ${gridswr} = 1 ]; then
    save_file ${coupler} input grids.nc grids_${res_atm}_${res_oce}${FV_cpl}.nc
    save_file ${coupler} input areas.nc areas_${res_atm}_${res_oce}${FV_cpl}.nc
    save_file ${coupler} input masks.nc masks_${res_atm}_${res_oce}${FV_cpl}.nc
  fi
  # SCRIP remapping matrices (if just created)
  if [ ${scripwr} = 1 ]; then
    save_file ${coupler} input \
      rmp_oces_to_atmo_CONSERV_FRACAREA.nc \
      rmp_oces_to_atmo_CONSERV_FRACAREA_${res_atm}_${res_oce}.nc
    ocetypes="oces oceu ocev"
    for type in $ocetypes;
    do
      save_file ${coupler} input \
        rmp_atmo_to_${type}_CONSERV_FRACAREA.nc \
        rmp_atmo_to_${type}_CONSERV_FRACAREA_${res_atm}_${res_oce}.nc
    done
    save_file ${coupler} input \
      rmp_atmo_to_oces_BILINEA.nc \
      rmp_atmo_to_oces_BILINEA_${res_atm}_${res_oce}.nc
    save_file ${coupler} input \
      rmp_atml_to_oces_CONSERV_FRACAREA.nc \
      rmp_atml_to_oces_CONSERV_FRACAREA_${res_atm}_${res_oce}.nc
  fi
  # extrapolation matrix (if just created)
  if [ $extrapwr = 1 ]; then
    save_file ${coupler} input nweights nweights_${res_atm}_${res_oce}${FV_cpl}
  fi
fi
#####
# Save raw output of ECHAM
#
#####
print "\n+++++ Save ECHAM5 output from $work to $data"
# check output format and save code files for first run
if [ "${out_filetype}" = "NETCDF" ]; then
  suf0=.nc
  suff=.nc
  method=tar
elif [ "${out_filetype}" = "GRIB" ];then
  suf0=""
  suff=.grb
  method=cat
else
  printf "      |- ERROR: Unsupported output file format ${out_filetype}\n"
  exit 0
fi
expmod=${expid}_${atmmmod}
datastreams="${expmod}"
[[ "${srfmod}" = "jsbach" ]] && datastreams="${datastreams} ${expmod}_co2"
[[ "${co2_transport}" = "true" ]] && datastreams="${datastreams} ${expmod}_tracer"
[[ "${save_dbldrad}" = "true" ]] && datastreams="${datastreams} ${expmod}_accuflx"
# Save ECHAM raw output
print "      |- Save ECHAM5 monthly raw (and restart) files of\n          datastreams $datastreams"
for datastream in ${datastreams};do
  if [[ "${datastream}" = "${expmod}" ]];then
    substream=""
    resubstr="_echam"
  else
    substream="_${datastream##*_}"
    resubstr="$substream"
  fi
fi

```

```

date=${startdate}
while [[ $(later_date -- ${date} ${enddate}) = ${enddate} ]]; do
  year=$(format_date -f4 -- ${date} | cut -f1 -d" ")
  month=$(format_date -f4 -- ${date} | cut -f2 -d" ")
  ym=${year}${month}
  save_file ${atmmmod} output ${expid}_${ym}.01${substream}${suf0} ${datastream}_${ym}${suff} &
  date=$(calc_date plus -c${caltype} -M1 -- ${date})
done # months
# Save ECHAM code files (for the first run)
if [ ${jobnum} = 1 ]; then
  save_file ${atmmmod} log ${expid}_${startyear}01.01${substream}.codes ${expid}_${atmmmod}${substream}.codes
fi
# Save restart files
save_file ${atmmmod} restart rerun_${expid}${resubstr} rerun_${expid}${resubstr}_${enddate} &
done # datastreams
if [ "${lhd}" = "yes" ]; then
  save_file ${atmmmod} output ${expid}_${year}01.01_hd_higres.nc ${expmod}_hd_higres_${startdate}.nc &
fi
# Save log files
if [ ${message_passing} != NONE ]; then
  save_file ${atmmmod} log atmtout atmtout_${startdate}_${enddate}
  save_file ${atmmmod} log ${atmmmod}.prt0 ${atmmmod}_${startdate}_${enddate}.prt0
fi
# Save hydrology restart files
if [ $lhd = yes ]; then
  print " | - Save ECHAM5 hydrology restart files from $work in &data"
  save_file ${atmmmod} restart hdrestart.nc hdrestart_${expid}_${enddate}.nc hdrestart_${expid}_${prevdate}.nc
fi
#####
# Save output files of JSBACH
#####
print "\n+++++ Save JSBACH raw output\n"
if [ ${out_filetype} = NETCDF ]; then
  suf0=.nc
  suff=.nc
  method=tar
elif [ ${out_filetype} = GRIB ]; then
  suf0=""
  suff=.grb
  method=cat
  if [ ${jobnum} = 1 ]; then
    save_file ${srfmod} log ${expid}_${startyear}01.01_land.codes \
      ${expid}_${srfmod}_land.codes
    save_file ${srfmod} log ${expid}_${startyear}01.01_jsbach.codes \
      ${expid}_${srfmod}.codes
    save_file ${srfmod} log ${expid}_${startyear}01.01_veg.codes \
      ${expid}_${srfmod}_veg.codes
    if [ ${jsb_standalone} = true ]; then
      save_file ${srfmod} log ${expid}_${startyear}01.01_driving.codes \
        ${expid}_${srfmod}_driving.codes
      save_file ${srfmod} log ${expid}_${startyear}01.01_forcing.codes \
        ${expid}_${srfmod}_forcing.codes
    else
      save_file ${srfmod} log ${expid}_${startyear}01.01_surf.codes \
        ${expid}_${srfmod}_surf.codes
    fi
  fi
else
  echo "ERROR: Unsupported output file format ${out_filetype} for ${srfmod}"
  exit 0
fi
# save raw JSBACH output
outmod=${exphome}/outdata/${srfmod}
expmod=${expid}_${srfmod}
datastreams=${expmod} ${expmod}_land ${expmod}_veg
if [ ${jsb_standalone} = true ]; then
  datastreams=${datastreams} ${expmod}_driving ${expmod}_forcing
else
  datastreams=${datastreams} ${expmod}_surf
fi
tar_suff=${suff}
for datastream in ${datastreams}; do
  if [[ "${datastream}" = "${expmod}" ]]; then
    substream=""
    origsubstream="jsbach"
  else
    substream="${datastream##*_}"
    origsubstream="$substream"
  fi
  date=${startdate}
  while [[ $(later_date -- ${date} ${enddate}) = ${enddate} ]]; do
    year=$(format_date -f4 -- ${date} | cut -f1 -d" ")
    month=$(format_date -f4 -- ${date} | cut -f2 -d" ")
    ym=${year}${month}
    tar_suff=${suff}
    save_file ${srfmod} output ${expid}_${ym}.01_${origsubstream}${suf0} ${datastream}_${ym}${suff} &
    date=$(calc_date plus -c${caltype} -M1 -- ${date})
  done # months
  # Restart files
  if [[ "${origsubstream}" != "land" ]]; then # no rerun file for substream land
    save_file ${srfmod} restart rerun_${expid}_${origsubstream} \
      rerun_${expid}_${origsubstream}_${enddate} &
  fi
fi

```

```

done # datastreams
#####
#
# Save output files of MPIOM
#
#####
print "\n++++ MPIOM output saving"
runper=${startdate}_${enddate}
### Rename and tar raw output fortran files
if [ -f fort.270 ]; then
  mv fort.270 ${expid}_oasisflux # fluxes
fi
[ "$imean" != "0" ] && mv fort.71 ${expid}_mpiom # main output
if [ -f fort.tar ]; then
  rm fort.tar
fi
# tarfile of all remaining fort.* and TIMESER.*
[ $(ls fort.* | wc -l) -ge 1 ] && tar cvf fort.tar fort.*
if [ -f TIMESER.asc ]; then
  cp TIMESER.asc TIMESER.${runper}.asc
  if [ -f fort.tar ]; then
    tar rvf fort.tar TIMESER.${runper}.asc
  else
    tar cvf fort.tar TIMESER.${runper}.asc
  fi
fi
if [ -f TIMESER.ext ]; then
  cp TIMESER.ext TIMESER.${runper}.ext
  if [ -f fort.tar ]; then
    tar rvf fort.tar TIMESER.${runper}.ext
  else
    tar cvf fort.tar TIMESER.${runper}.ext
  fi
fi
### save raw output from compute node on workshare
outmod=${exphome}/outdata/${ocemod}
outfile=${expid}_${ocemod}
outfile_tp="${expid}_${ocemod}_${runper}"
if [ "$imean" != "0" ]; then
  save_file ${ocemod} output ${outfile} ${outfile_tp}.ext &
fi
### save oasisflux from compute node on workshare
outfile=${expid}_oasisflux
outfile_tp="${expid}_oasisflux_${runper}"
if [ -f ${outfile} ]; then
  save_file ${ocemod} output ${outfile} ${outfile_tp}.ext
fi
if [ -f fort.tar ] || [ -f ${outmod}/fort_${runper}.tar ]; then
  save_file ${ocemod} output fort.tar fort_${runper}.tar
fi
# save TIMESER* to "data" directory but not individually to archive since they are already included in fort.
print " | - Save timeseries"
if [ -f TIMESER.asc ] || [ -f ${outmod}/TIMESER.${runper}.asc ]; then
  save_file ${ocemod} output TIMESER.asc TIMESER.${runper}.asc
fi
if [ -f TIMESER.ext ] || [ -f ${outmod}/TIMESER.${runper}.ext ]; then
  save_file ${ocemod} output TIMESER.ext TIMESER.${runper}.ext
fi
# Restart files
# find out, which of the restart files is the newest
file=$(ls -rt Z37000 Z38000 | tail -1)
# Log files
save_file ${ocemod} log oceout oceout_${runper}
if [ "${coupler}" = "oasis3" ]; then
  save_file ${ocemod} log mpi-om.prt0 ${ocemod}_${runper}.prt0
fi
save_file ${ocemod} restart ${file} RESTART_${expid}_${enddate} &
#####
# Save output files of HAMOCC
#
#####
printf "\n++++ HAMOCC raw output saving\n"
### save raw output from compute node on workshare
outmod=${exphome}/outdata/${bgcmmod}
expmod=${expid}_${bgcmmod}
timeperiode=${startdate}_${enddate}
substreams="2d 3d bioz sed"
for substream in ${substreams}; do
  outfile_tp=${expmod}_${substream}_${timeperiode}
  save_file ${bgcmmod} output bgcmean_${substream}.nc \
    ${expid}_${bgcmmod}_${substream}_${timeperiode}.nc &
done
save_file ${bgcmmod} output timeser_bgc.nc ${expid}_${bgcmmod}_timeser_${startdate}_${enddate}.nc
# Restart files
save_file ${bgcmmod} restart restartw_bgc.nc restart_bgc_${expid}_${enddate}.nc &
# Log files
save_file ${bgcmmod} log bgcout bgcout_${startdate}_${enddate}
#-----
# Check whether everything was saved successfully
#-----
wait
if [ ${saving_error} = no ]; then

```

```

print "      |+ Everything saved successfully"
if [ -s rm.lst.$$ ] ; then
  printf "      |+ Removing files listed in file rm.lst.$$\\n"
  for file in $(cat rm.lst.$$); do
    printf "      |- Removing file : $file\\n"
    ${rm} $file
  done
  rm rm.lst.$$
fi
if [ "${rm_list}" != "" ]; then
  print "      |- Removing files ${rm_list}\\n"
  ${rm} ${rm_list}
fi
else
  printf "      |- ERROR occured: saving_error = ${saving_error}\\n"
fi
#-----
#
#      8. SUBMISSION OF THE NEXT JOB
#-----
cd ${jobdir}
#
# Number of the next job
#
(( nextjob = ${jobnum} + 1 ))
#
# edit .date and .log file
#
space_error="no"
echo "${nextdate} ${nextjob}" > ${expid}.date.new || {
  space_error="yes"; echo "Could not create ${expid}.date";
}
cp ${expid}.log ${expid}.log.new || {
  space_error="yes"; echo "Could not save ${expid}.log";
}
echo "$(date +"${datefmt}") : ${jobnum} ${nextdate} ${jobid} - done" >> ${expid}.log.new || {
  space_error="yes"; echo "Could not append to ${expid}.log";
}
if [ "${space_error}" = "no" ]; then
  mv ${expid}.date.new ${expid}.date
  mv ${expid}.log.new ${expid}.log
else
  echo "No disk space left or quota exceeded?"
  echo " - Show quota"
  quota
  exit
fi
#
# Check whether final date is reached
#
if [[ `later_date -- ${nextdate} ${findate}` = ${nextdate} ]]; then
  echo "Experiment over"
  echo "$(date +"${datefmt}") : Experiment over" >> ${expid}.log
else
  qs=${queueing_system}
  submit -q ${qs} ${job}
fi
#####
#
#      MONITORING
#
#####
#-- edit the monitoring script
#
cd ${jobdir}
if [ "${monitoring}" = "yes" ]; then
  cp ${expid}.mon ${expid}.mon.${nextdate}
  ed -s ${expid}.mon.${nextdate} <<EOF
1,100s/Jobnum/${jobnum}/
1,100s/Startdate/${startdate}/
1,100s/Nextdate/${nextdate}/
1,100s/Findate/${findate}/
w
q
EOF
  qs=${queueing_system_pp:-$queueing_system}
  submit -q ${qs} ${expid}.mon.${nextdate}
fi
#####
#
#      POST PROCESSING
#
#####
#-- edit the postprocessing script
#
cd ${jobdir}
if [ ${postprocessing} = yes ]; then
  cp ${expid}.post ${expid}.post.${nextdate}
  ed -s ${expid}.post.${nextdate} <<EOF
1,100s/Jobnum/${jobnum}/
1,100s/Startdate/${startdate}/
1,100s/Nextdate/${nextdate}/
1,100s/Findate/${findate}/
w

```

```
q
EOF
qs=${queueing_system_pp:-$queueing_system}
submit -q ${qs} ${expid}.post.${nextdate}
fi
#-----
#
#      9. EPILOGUE
#-----
#
date
${job_account}
wait
exit
```

Appendix C

Examples

In the following we give three concrete examples of experiments with given *expid*'s ID1 to ID3, how to perform an experiment with a given *expid* and each for special coupled model *cplmod* and IT-environment.

Let for all examples *userid* the UNIX user id, under which control the experiments are performed. This account must of course have login access to all involved hosts.

C.1 COSMOS-ao on SX-6 and the cross platform at DKRZ

We describe here a COSMOS experiment, where the atmosphere model "ECHAM5" and the physical ocean model "MPIOM" are coupled by the coupler "OASIS3". Model execution is performed by interactive usage of the HPC computer "hurrikan", a NEC SX-6 super computer at the DKRZ. I.e. it is launched by submitting the run scripts by the NQSII queueing system ¹ on the Multi Node HPC computer "Hurrikan", which is installed at the DKRZ ².

1. Login on the submit and compute system, which is here the "cross" system 'cross.dkrz.de' :

```
ssh cross.dkrz.de
```
2. Check out the IMDI and model sources from repository the MaD repository into your working directory, if not already done, e.g. in your home directory:

```
cd  
svn co http://svn-mad.zmaw.de/svn/mad/Model/IMDI/trunk imdiroot
```
3. Let in the following *root* the root directory, where you checked out the IMDI version from SVN. Change in the tools directory.

```
root= userid/imdiroot  
cd root/util/running/tools
```
4. Generate a setup file for
 - an experiment with *expid* ID1,
 - the coupled COSMOS model *cosmos-ao*,
 - the 'symbolic node name' *ds* (which is the default setting and corresponds to the cross system at DKRZ) and
 - with model executables, which are resulted from by the MPI Fortran compiler for SX-6 platforms *sxf90* compiled model sources (default setting as well) and which have to reside in *root/SX-6/bin*.

¹For more information about the NQSII queueing system, see http://www.dkrz.de/ec.var/manpages/NQSII/NQSII_usercommands.html

²For more information about the NEC-SX6 at DKRZ check <http://www.dkrz.de/dkrz/services/docs/manuals>

by entering

```
Create_TASKS.frm --id ID1 [ -n ds -c sxf95 ] cosmos-ao
```

5. We change only the integration periode in the generated setup file `./setup/setup_cosmos-ao_ID1` from twenty to four years:

```
final_date=0803-12-31 # final date of the experiment
```

6. Rerun `Create_TASKS.frm`, to generate the task scripts.

```
Create_TASKS.frm --id I1 cosmos-ao
```

7. Change to script directory and ashure that all task scripts and executables are available.

```
cd /root/experiments/I1/scripts
```

```
ls
```

```
ls /root/SX-6/bin/
```

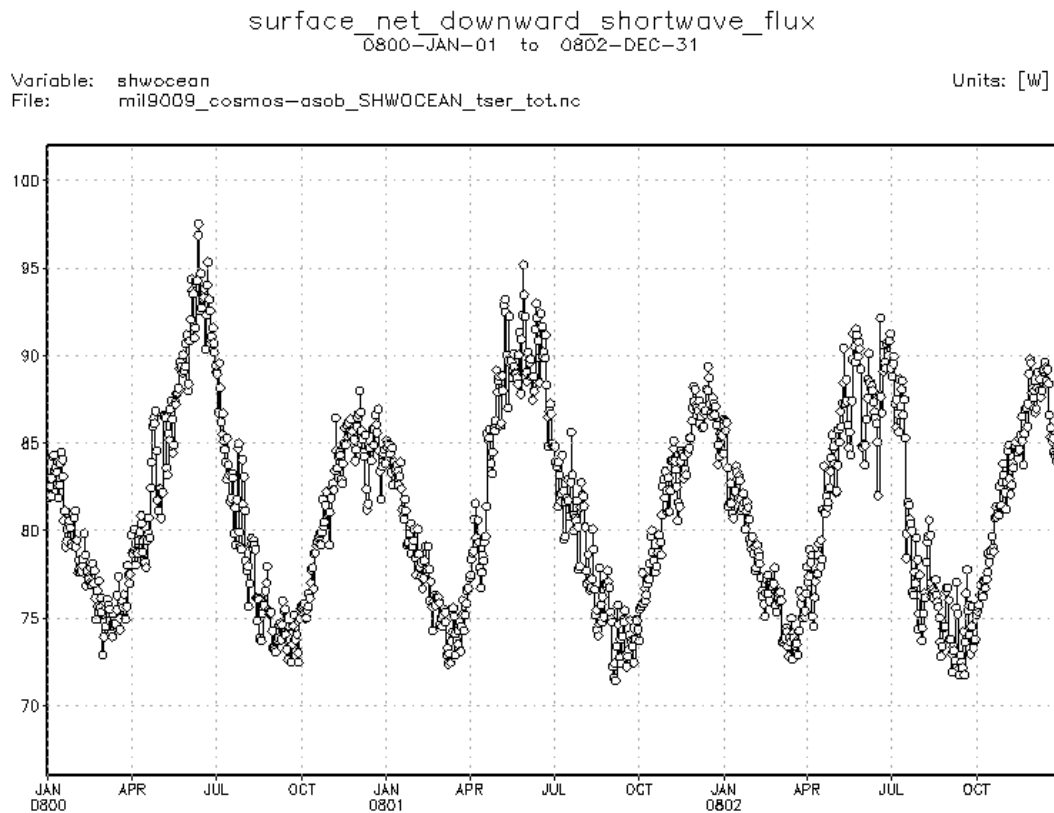
```
echam5.I1.MPI1.x mpiom.I1.MPI1.x oasis3.MPI1.x
```

8. Launch the experiment by

```
qsub I1.run
```

You can check and control the status of your experiment with several tools and logfiles :

- `qstat` returns the status of your running jobs, e.g. :
- `ID1.o<jobno>` : standard output of the run script
- In `ID1.log` ist der Status und in
- In `ID1.date` the Job number and model date of your next job



Heiner Widmann

02/20/08

Figure C.1: Example for a total timeseries plot as shown on a html page if monitoring is active.

monitoring=yes

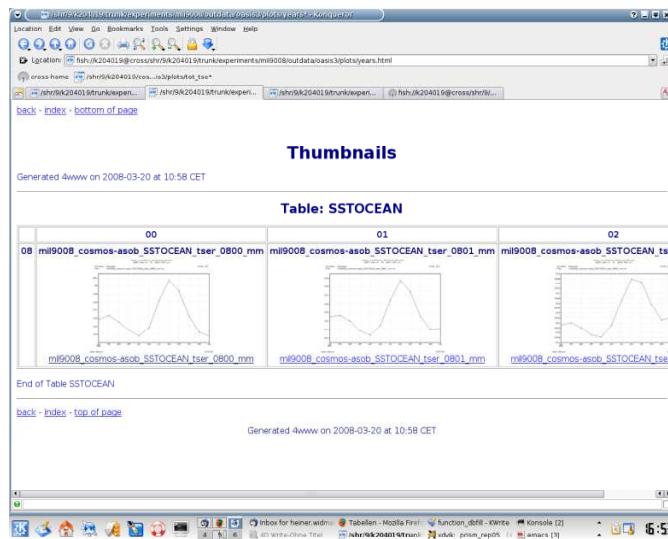


Figure C.2: Example for the three generated figures of yearly time series over 12 monthly means

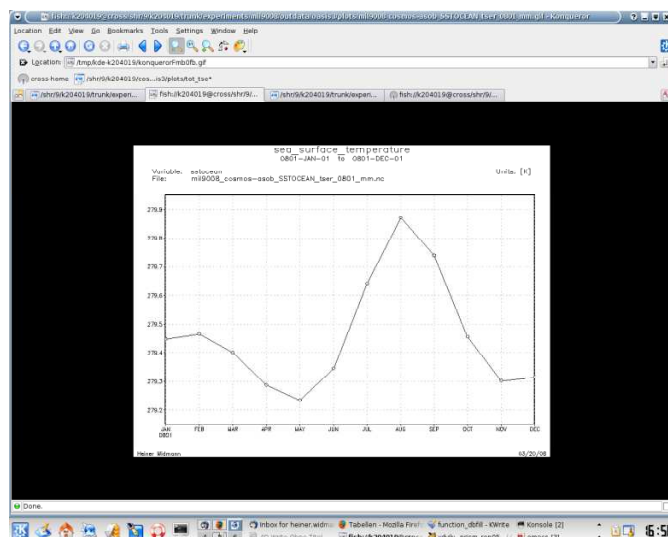


Figure C.3: By clicking on one figure, the corresponding year is shown

C.2 COSMOS-asob on the Linux cluster tornado at ZMAW

We describe here a full coupled COSMOS experiment, i.e. with all for model components (atmosphere, surface, ocean and biogeochemical ocean). It is launched by submitting the run scripts by the SGE³ on the Linux cluster "Tornado", which is installed at the ZMAW⁴.

1. Login on the submit and compute node, which is here one and the same computer 'tornado1.dkrz.de':
:


```
ssh tornado1.dkrz.de
```
2. Check out the IMDI and model sources from repository the MaD repository into your working directory, if not already done, e.g.:


```
cd /scratch/wrkshr/userid
svn co http://svn-mad.zmaw.de/svn/mad/Model/IMDI/trunk imdirroot
```
3. Let in the following *root* the root directory, where you checked out the IMDI version from SVN. Change in the tools directory.


```
root=/scratch/wrkshr/userid/imdirroot
cd root/util/running/tools
```
4. Generate a setup file for
 - an experiment with *expid* ID2,
 - the coupled COSMOS model *cosmos-asob*,
 - the 'symbolic node name' *linux-x64* (which corresponds to the 64 Bit Linux architecture of the Tornado cluster) and
 - with model executables, which resulted from compilation of model sources with the Intel Fortran compiler *ifort* and which have to reside in *root/x86_64-intel/bin*.

by entering

```
Create_TASKS.frm --id ID2 -n linux-x64 -c intel cosmos-asob
and you get the standard output as shown in C.1
```

```
Creating setup file setup\_cosmos-asob\_ID2
```

```
-----
The setup file:
```

```
  /scratch/wrkshr/k204019/imdirroot/util/running/setup/setup\_cosmos-asob\_ID2
needs to be edited according to the experimental setup.
```

```
When this is done run Create\_TASKS.frm again:
```

```
  Create\_TASKS.frm cosmos-asob ID2
-----
```

Table C.1: Output of the first call of `Create_TASKS.frm`

5. We make two changes in the generated setup file `../setup/setup_cosmos-asob_ID2`:


```
final_date=0803-12-31 # final date of the experiment
nproca_atm=6 (which leads to total 8 processors in
I.e., we run the experiment just for four years and with six processors for the atmosphere component,
which leads to an appropriate processor configuration for Tornado.
```

³For more information about the Sun Grid Engine (SGE), the queuing system on "Tornado", see <http://www.sun.com/software/gridware>

⁴For more information about the Tornado Linux Cluster check <https://tornado-wiki.dkrz.de/farm/FrontPage>

6. Rerun `Create_TASKS.frm`, to generate the task scripts.

```
Create_TASKS.frm --id I2 -n linux-x64 -c intel cosmos-asob
```

7. Change to script directory and assure that all task scripts and executables are available.

```
$ cd root/experiments/I2/scripts
```

```
$ ls
```

```
ID2.arch ID2.post ID2.run
```

```
$ ls root/x86_64-intel/bin/
```

```
echam5j_ID2.MPI1.x mpiom_hamocc_ID2.MPI1.x oasis3.MPI1.x
```

8. Launch the experiment by

```
$ qsub ID2.run Your job 302659 ("ID2.run") has been submitted
```

You can check and control the status of your experiment with several tools and logfiles :

- `qstat` returns the status of your running jobs, e.g. :

```
k204019@tornado1:~...ID2/scripts> qstat | grep ID2
 303610 0.53979 ID2_asob.3 k204019      r      10/28/2008 12:47:32 cluster@tc138
 303609 0.53854 ID2.run      k204019      r      10/28/2008 12:47:07 serial@tornado
k204019@tornado1:~...ID2/scripts>
k204019@tornado1:~...ID2/scripts>
k204019@tornado1:~...ID2/scripts> qstat -t | grep ID2
 303610 0.53979 ID2_asob.3 k204019      r      10/28/2008 12:47:32 cluster@tc138
 303610 0.53979 ID2_asob.3 k204019      r      10/28/2008 12:47:32 cluster@tc210
 303609 0.53854 ID2.run      k204019      r      10/28/2008 12:47:07 serial@tornado
```

Table C.2: Check status of an experiemnt ID2 as output of `qstat`

- `ID2_asob.<jobno>.<jobno2>` : standard output of the model execution job
- In `ID2.log` ist der Status und in
- In `ID2.date` the Job number and model date of your next job

Bibliography

- Mangili, A., M. Ballabio, M. Djordje, L. Kornblueh, R. Vogelsang, and P. Bourcier, 2003: PRISM Software Engineering, Coding Rule, Quality Standards, 31 pp.
(http://prism.dkrz.de/Workpackages/WP3i/WP2b_coding_rules.ps)
- Legutke, S., I. Fast, and V. Gayler, 2007: The PRISM Standard Compilation Environment, PRISM Report Series No 4, nn pp. (http://prism.enes.org/Results/Documents/PRISM_Reports/Report4.pdf)
- Valcke, S., A. Caubel, R. Vogelsang, and D. Declat, 2004: OASIS3 User Guide (oasis3_prism_2-4). PRISM Report Series, No 2, 5th Ed., 60 pp.
(http://prism.enes.org/Results/Documents/PRISM_Reports/Report2.pdf).
- Valcke, S., R. Redler, R. Vogelsang, D. Declat, H. Ritzdorf, T. Schoenemeyer, 2004: OASIS4 User Guide. PRISM Report Series No 3, 72 pp.
(http://prism.enes.org/Results/Documents/PRISM_Reports/Report3.pdf).
- Meier-Fleischer, K., xxxx 2004: Low end vizualisation of PRISM model output, PRISM Report Series No 12, nn pp. (http://prism.enes.org/Results/Documents/PRISM_Reports/Report12.pdf).

Index

- CDO, 19
- Coupled models, 27
- Create_TASKS.frm, 9

- Experiment, 7
 - Examples, 35
 - experiment ID, 7
- Experiment:Configuration of, 11

- Graphics
 - Madplot, 17

- Model components, 27
- Monitoring, 17

- Setup, 11
 - Generation of, 9
- Standard Compiling Environment (SCE), 2
- Standard Directory Structure, 5
- Standard Running Environment (SRE), 2
- SVN, 2

- Tasks, 9, 15
 - Archiving, 20
 - Data base filling, 21
 - Generation of, 9
 - Monitoring, 17
 - Post processing, 19
 - Pre processing, 17
 - Running, 17
 - runscript, 17

