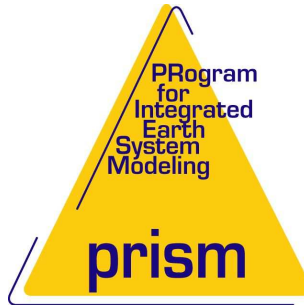


PRISM
Project for Integrated Earth System Modelling
An Infrastructure Project for Climate Research in Europe
funded by the European Commission
under Contract EVR1-CT2001-40012



PRISM SRE
Standard Running Environment
Handbook

Edited by:
Veronika Gayler and Stephanie Legutke

PRISM-Report Series-05

0. Edition
(last change: September 27, 2004)

Copyright Notice

© Copyright 2003 by PRISM

All rights reserved.

No parts of this document should be either reproduced or commercially used without prior agreement by PRISM representatives.

How to get assistance?

The individual work packages of the PRISM project can be contacted as listed below.

PRISM publications can be download from the WWW server of the PRISM project under the

URL: <<http://prism.enes.org/Results/Documents/>>

Phone Numbers and Electronic Mail Adresses

Electronic mail addresses of the individual work packages are composed as follows :

prism_ *work package* @prism.enes.org

Name	Phone	<i>PRISM Work Package</i>
Veronika Gayler	+49-40-41173-138	<i>wp3i</i>
Stephanie Legutke	+49-40-41173-104	<i>wp3i</i>

Contents

1	Introduction	1
1.1	Components of the SRE	1
1.2	Design of a PRISM Experiment	1
1.3	Notes to the Usage of this Handbook	2
2	The Standard Directory Structure	3
2.1	The <code>util</code> directory	4
2.2	The <code>data</code> directory	4
2.3	The <code>experiments</code> directory	4
3	The Tasks	7
3.1	The Suite of Tasks	7
3.1.1	Running	7
3.1.2	Postprocessing	8
3.1.3	Archiving	8
3.1.4	Visualisation	9
3.2	Generation of the Tasks	9
3.3	The Setup	10
3.4	Setup and <code>Complete_setup</code>	14
3.5	The Functions	14
3.6	The Calendar	17
4	Enlarging the System	19
4.1	Adding a new Coupled Model	19
4.1.1	Adaptation of <code>Create_TASKS.frm</code>	19
4.1.2	Include Files Depending on the Coupled Model Combination	20
4.1.3	Include Files Depending on the Component Model	21
4.1.4	Providing the input data	22
4.1.5	Providing adjunct files	23
4.2	Adaptation of the SRE to a new Site	24
4.2.1	Node Dependant Include Files	24
5	Running a coupled model Step-by-Step	27
6	Using the GUI	29
6.1	Configuring an Experiment with PrepIFS	29
6.2	Monitoring an Experiment with SMS	29

List of Figures

1.1	Flowchart of a PRISM experiment	2
2.1	The PRISM standard directory tree	3
3.1	The suite of tasks of an experiment	7
3.2	The structure of a PRISM runscript	8
3.3	Schematic view on the tasks' generation using <code>Create_TASKS.frm</code>	10
3.4	Structure of the setup	10

List of Tables

1.1	Definition of the variables used in the handbook	2
3.1	List of the Unix commands defined in the setup	13
3.2	Variables of setup and complete_setup	15
3.3	List of in- and output parameters of the calendar tool.	17
4.1	Variable names of the model components	20
4.2	Variables used in the namcouple base file	23

Chapter 1

Introduction

One of the key aspects of the PRISM project is the definition of common standards. The PRISM standard compile environment (SCE) and the standard running environment (SRE) play a central role as they bring together most of work that had been done within the PRISM project starting from the models' source code organization until the visualisation of output data. The standard environments are intended to facilitate running a coupled model combination. A common look&feel for all PRISM coupled model combinations minimizes the effort to setup and run coupled model experiments. The standards also help designing and running new coupled model combinations and facilitate portability to new platforms.

1.1 Components of the SRE

The running environment comprises a standard directory structure which is described in detail in chapter 2. It is closely connected to the source code directory structure of the SCE (Legutke and Gayler (2004)). Besides, the SRE includes a comprehensive set of utilities to generate standardized tasks (i.e. scripts for model integration, data pre- or postprocessing, visualisation and archiving of output data). The tasks are composed of several short include files, some of them platform or model dependant. These files are assembled to the tasks using the m4 preprocessor. This method allows for easy adaption to new coupled models or new platforms as model and site dependant sections are clearly identified. The tasks and their generation is topic of chapter 3.

1.2 Design of a PRISM Experiment

A PRISM experiment starts with the checkout of model source code and utilities from the central PRISM cvs repository and ends with the visualisation and archiving of output data. Depending of the user's preference it is possible to setup and monitor an experiment using a graphical user interface (GUI). Figure 1.1 demonstrates the close relationship between the GUI and the scripting level. With both methods at the beginning of the experiment the source code of the component models and the coupler is retrieved from the cvs repository. It is up to the user to adapt the code for the experiment he is going to run. The tasks for compilation, the Makefiles and the tasks for model execution are created from identical header files whether or not the GUI is used. Whereas at the scripting level the compile scripts are submitted manually they are launched by SMS using the GUI. Same is true for the runscript. The following chapters are focusing on the scripting level. Details on running a model using the GUI are given in chapter 6.

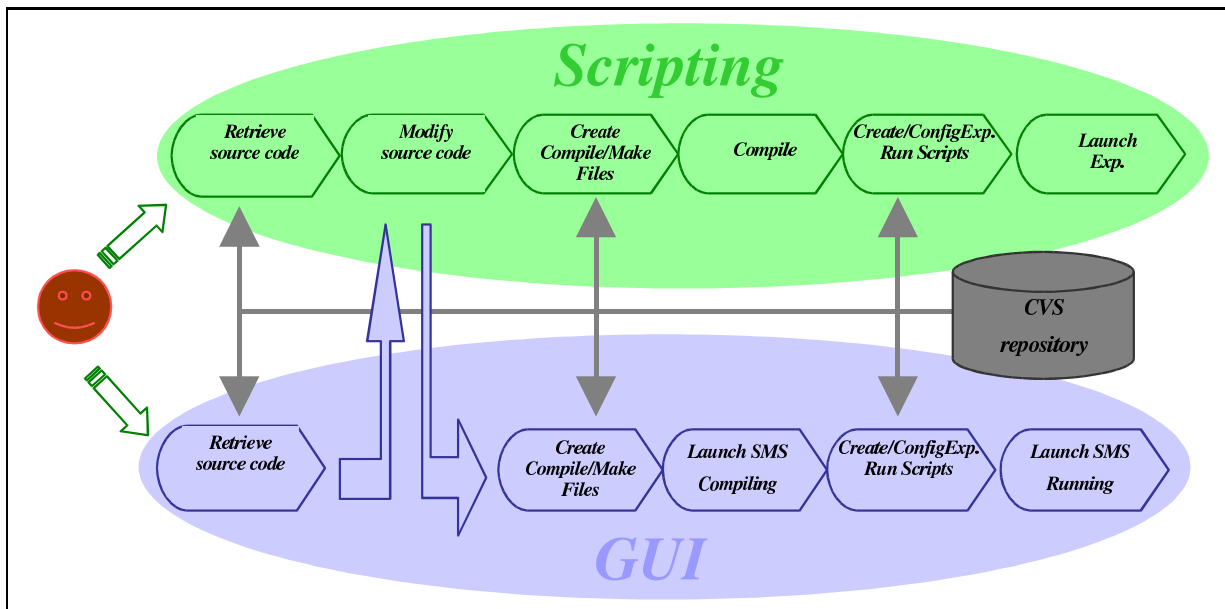


Figure 1.1: Flowchart of a PRISM experiment comparing scripting level and the usage of a GUI.

1.3 Notes to the Usage of this Handbook

The document at hand is describing the SRE at its state at the end of the project (version prism_2-4). Even though the SRE has a very flexible design there arise new requirements with new models and new platforms being integrated. That means that it has not reached a final form. For further information on the current standards please refer to the WP-3i web page <http://prism.dkrz.de/Workpackages/WP3i/> which is continuously updated.

To indicate file and directory names, Unix commands or standard input of programs the typewriter font is used. 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 1.1 gives a list of the variables used in the following chapters.

Variable	Description
<i>model</i>	component model name
<i>cp1mod</i>	name of the coupled model combination
<i>node</i>	node name of the computing host
<i>res</i>	horizontal resolution of a component model
<i>cmp</i>	abbreviation for the components: atm, che, srf, oce, ice, bgc

Table 1.1: Definition of the variables used in the handbook

Short instructions on running an existing PRISM coupled model are given in chapter 5. Instructions for the compilation are not included. They are available from the PRISM SCE handbook or from the PRISM web site mentioned above.

Detailed information on how to enlarge the system is provided in chapter 4. If you plan to adapt a new coupled model combination to the standard running environment or to port the system to a new site please do not hesitate to contact us. We are interested in your experiences and are happy to answer questions. Besides it is possible to integrate your changes into the central cvs system.

Chapter 2

The Standard Directory Structure

A central feature of the PRISM standard modeling environment is a standardized directory structure. It interfaces the standard compile environment (SCE) and the standard running environment (SRE). The directory tree comprises the source code and the libraries of the component models and the coupler, the utilities needed for compilation and provides room for binaries and executables (compare the PRISM SCE handbook (Legutke and Gayler (2004))). Besides, it includes execution utilities and the initial data needed to run a coupled model.

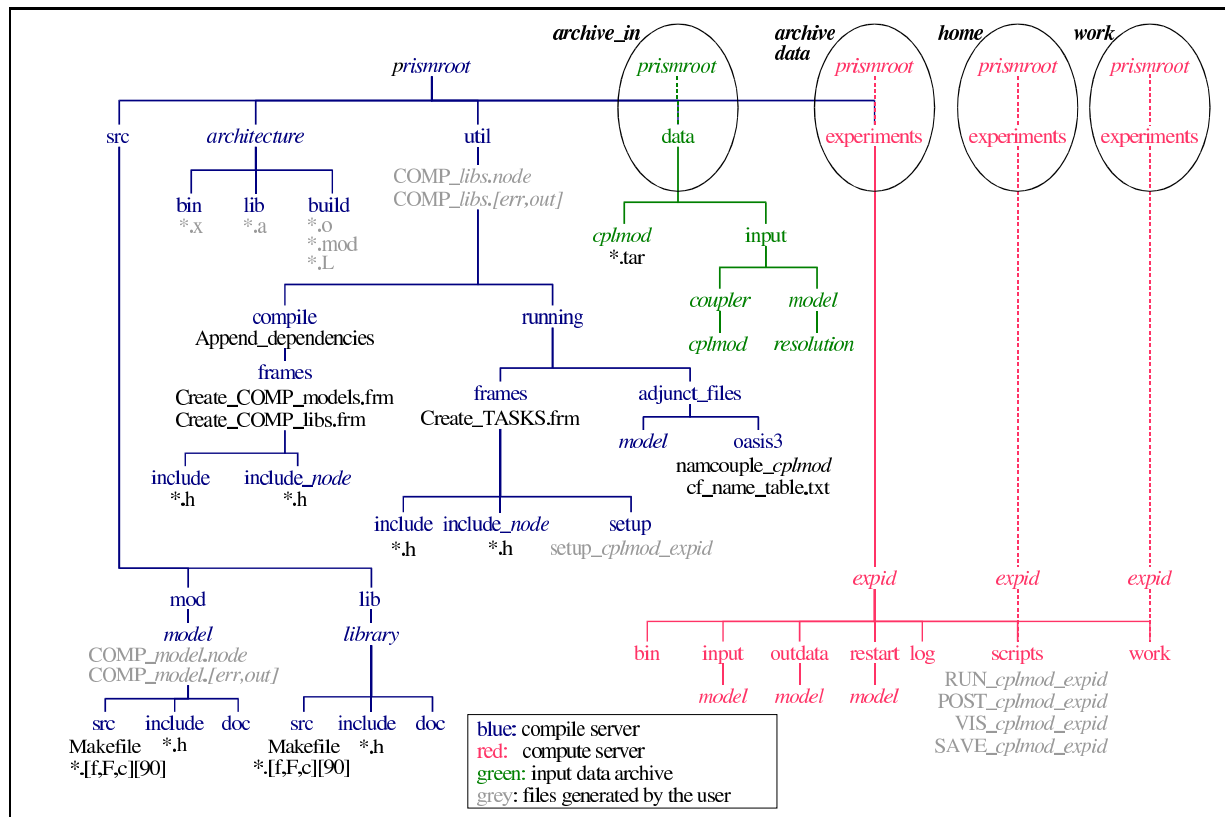


Figure 2.1: The PRISM standard directory tree

The standard directory structure is displayed in figure 2.1. The name of the root directory (*prismroot*) can be defined by the user. As compilation, execution, postprocessing, visualisation and archiving rarely takes place on a single machine, the directory structure might span several file systems on several machines. In that case several independent root directories are needed. The variables *archive*, *data*, *archive* and *home* refer to configurable variables of the experiment setup (section 3.3).

The checkout of a coupled model from the cvs repository provokes the inflation of the standard directory structure with a root directory called `prism`. It is possible to rename this directory. Renaming or moving any other directory of the tree may cause errors. The `prismroot` directory has six subdirectories. Two of them (`src` and `architecture`) are part of the SCE. They are described in the SCE handbook (Legutke and Gayler (2004)). The remaining three branches are described below.

2.1 The `util` directory

The `util` directory, also introduced in the SCE handbook (Legutke and Gayler (2004)), contains tools and scripts for compilation (`util/compile`) and for the execution of a coupled model (`util/running`). Directory `util/running/frames` contains a script to generate the tasks called `Create_TASKS.frm`. It uses several include files residing in the directories `util/running/include` and `util/running/include_node`. The variable `node` specifies the node name of the computing host. The setup files of different experiments defined by the user will be placed in the directory `util/running/setup`. A detailed description of the tasks and their generation is given in chapter 3.

Some models need namelist like input files. These ASCII text files are rather associated to the namelists (generally defined in the runscripts) than to initial data files. They are kept in directory `util/running/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 `util/running/adjunct_files/oasis3/cplmod/namcouple_cplmod`. For more information on the `namcouple` please read section 4.1.5.

2.2 The `data` directory

The directory `data` contains input data files for the component models and the coupler (`data/input/model`). These input data files can be used for different experiments with different coupled model combinations. Tar files with input data for a specific coupled model combination are stored in `data/cplmod`. In contrast to the input files of a single model the tar files are available from the PRISM central cvs repository.

2.3 The `experiments` directory

In contrast to the directories described above the `experiments` directory is not available from the cvs repository. It is installed when the user defines a new PRISM experiment. All his experiments are stored in the `experiments` directory, each of them in a special subdirectory labeled with the experiment ID (`expid`). The directory `experiments/expid` contains all that is needed to run the specific experiment. The executables of the component models and the coupler are placed in `experiment/expid/bin`. These executables are copied from the directory `architecture/bin` at the beginning of the experiment. They stay unchanged on the computing host until the end of the experiment, even if a model is recompiled.

The input data needed for experiment `expid` is stored in `experiments/expid/input`. Each of the component models and the coupler have separate input data directories (`experiment/expid/input/model`). Analogously there exist separate directories for the output data (`experiment/expid/outputdata/model`) and the restart files (`experiment/expid/restart/model`) of the components. The adjunct files of the component models and of the coupler are copied from the `util/running/adjunct_files` directory to the models' input directories when the tasks are created.

The directory `experiment/expid/log` contains the log files of the model components and the standard output files of the runs.

The scripts (tasks) used to execute the experiment *expid* can be found in `experiment/expid/scripts`. Depending on the setup of the experiment there might be scripts for running, postprocessing, archiving, visualisation and others. At run time two additional files will be created and placed into this directory: The `expid.log` and the `expid.date` file. Both files are important for the course of the experiment and must not be removed.

The directory `experiment/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 `experiment/expid/work`. When the run is over the output is transferred from there to the experiment's output data directories (`experiment/expid/output/`*model*).

Chapter 3

The Tasks

The execution of a model experiment consists of different tasks, one following the other. The tasks currently supported within the SRE are execution of the coupled model, postprocessing, visualisation and archiving of the output data. The features of the different tasks are described in detail in section 3.1.

Usually the tasks are model and site dependant, 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 of maintenance PRISM distributions do not contain runscripts or other ready-to-use tasks but a tool to generate the tasks from a collection of short include files. Details about the tasks' generation are given in section 3.2.

3.1 The Suite of Tasks

An experiment is composed of one or several consecutive runs. Each run consist of one or more tasks. At the beginning of an experiment the runscript is submitted. When the model integration of the first run is completed, the runscript submits itself again for the integration of the next run. Besides it submits the next task. In the case displayed in figure 3.1 it is the postprocessing task. At the end of the postprocessing script the visualisation task is submitted which again submits the archiving task. 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 postprocessing is supported only for ECHAM5 output. Visualisation is optional. Whether or not archiving is expedient is site dependant (compare section 3.3).

It is specific to the scripting level that one task is submitting the other. In the case of experiments controlled through the GUI all tasks are submitted by SMS.

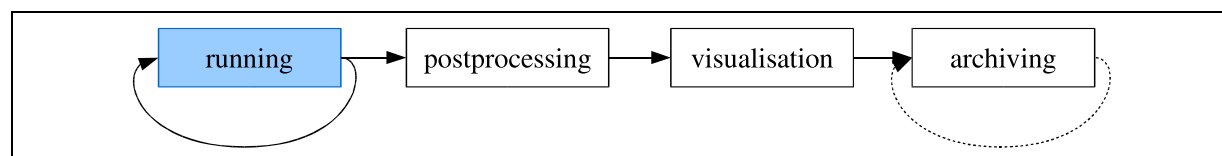


Figure 3.1: The suite of tasks of an experiment. Currently supported are tasks for running, postprocessing, visualisation and archiving.

3.1.1 Running

The runscript manages the integration of a coupled model. Beforehand the executables of the model components and the input and restart data are transferred to the working directory. After the integration output data and log files are saved. Finally the runscripts resubmits itself for the next run and eventually submits a follow-on task. Figure 3.2 shows the structure of the runscript. As the other tasks it is composed of several include files depending on either the coupled model, the site or the component.

Definitions	comments_ <i>cplmod</i> .h queue_commands_ <i>node</i> .h setup_ <i>cplmod</i> _expid.h complete_setup_ <i>cplmod</i> .h functions.h calendar.h
Pre-processing	create_directories.h get_executables_ <i>cplmod</i> .h get_input_data_ <i>model</i> .h namelist_ <i>model</i> .h
Execution	launching_ <i>cplmod</i> _ <i>node</i> .h
Post-processing	save_output_ <i>model</i> .h
Job submission	submit_next_jobs.h

Figure 3.2: The structure of a PRISM runscript. It consists of several include files, some of them depending on the coupled model (*cplmod*), the site (*node*) or the component (*model*). Include files depending on the component are listed only once in the figure. However there is one file of the same kind for each component model and for the coupler.

The runscript 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 sections 3.3 and 3.4. The runscript makes use of several ksh-functions. For details please read section 3.5. A calendar tool is used for time control (compare section 3.6).

The preprocessing phase starts with the creation of directories. All directories needed for the experiment are generated at run time. This includes directories on a remote archiving host. The executables of the component models and of the coupler are transferred to the working directory. Same is true for the input data (compare section 3.5). Namelists 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 postprocessing phase model output is saved. This is explained closely in section 3.5. Finally the runscript submits the next tasks.

3.1.2 Postprocessing

Postprocessing is highly model dependant. Sofar postprocessing is supported for ECHAM5 only. The postprocessing tool used here is the so called afterburner developed at the Max-Planck-Institute (<http://www.mpimet.mpg.de/afterburner/>). Most include files that build the postprocessing script are used for the runscript as well. This is true for the setup, complete-setup some functions and for the calendar tool.

3.1.3 Archiving

The archiving task 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 postprocessing phase of the runscript the model output is transferred from the working directory to the `data` file system (compare section file systems of section 3.3). 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.

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.

All the include files used to generate the archiving script are part of the runscript as well. Function

`save_file` which performs the archiving is working differently when it is called within a runscript then within an archiving script.

3.1.4 Visualisation

The output of an ongoing experiment can be visualised. The core of the visualisation task is the script `LE_parameter.py`. The python script makes use of `cdat`. More information is given in the WP4a handbook.

Visualisation can be carried out on the compute server or on an extra visualisation host where the visualisation software is installed. In the latter case all output data needed for the visualisation is copied to the visualisation host. The images produced are in gif format. They are moved to a web server and can be used to monitor the experiment through the web.

3.2 Generation of the Tasks

As mentioned above the tasks are assembled from several include files depending on the coupled model combination, the component model or the site. Others can be used for all models on all sites. Identical include files are used when an experiment is run through the GUI and when it is run at the scripting level. But the tools to assemble the tasks and the tasks themselves differ.

At the scripting level the assemblage of include files is performed by a script called `Create_TASKS.frm`. It makes use of the m4 macro-processor. The script resides in the directory `util/running/frames`. `Create_TASKS.frm` needs at least two input parameters: the name of the coupled model combination the tasks are created for and an experiment ID of the user's choice. By default, the scripts are generated for the machine where `Create_TASKS.frm` runs on. Exceptions are cross (node name `ds`, DKRZ) and rhodes (IPSL) where the tasks are created by default for hurrikan and uqbar respectively. To generate the tasks for another machine, the node name can be given as a third (optional) parameter.

```
Usage:
  Create_TASKS.frm 'cplmod' 'expid' ['node']

  cplmod: name of the coupled model
  expid:  experiment ID
  node:   node name of the computing host - if other than default.
          default computing host is: hurrikan
```

Usage of `Create_TASKS.frm`. The text was produced by typing "`Create_TASKS.frm --`" on `cross.dkrz.de`

The first call of `Create_TASKS.frm` for a specific model combination and experiment ID leads to the generation of a setup file (`util/running/frames/setup/setup_cplmod_expid`). This file contains a list of all configurable variables for the selected coupled model combination. It needs to be edited according to the experimental design. Comments give a short description of the variables and an overview of the choices. This editing is not needed with SMS where the user makes his selections through a GUI. More information on the setup file is given in the section below (3.3).

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 appropriate setup file. The tasks are transferred to the computing host in the directory `experiment/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 3.3 gives a graphical overview on the usage of `Create_TASKS.frm`. A note for ECHO users: In contrast to the former script `PREP_TASKS` `Create_TASKS.frm` just produces

one task of a kind, that will be used for all runs of the experiment.

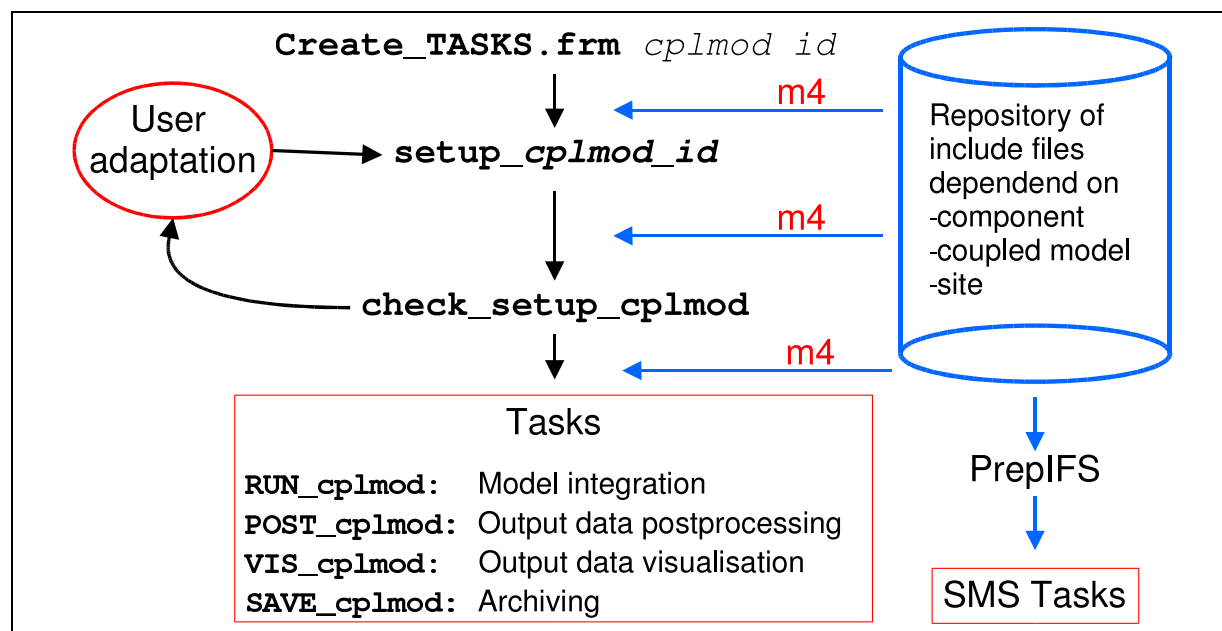


Figure 3.3: Schematic view on the tasks' generation using `Create_TASKS.frm`

3.3 The Setup

The setup of an experiment contains the users choices of all configurable variables of an coupled model configuration. It gives a precise description of an experiment which is valuable for the documentation of an 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 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 combination, the component model or the site (compare figure 3.4).

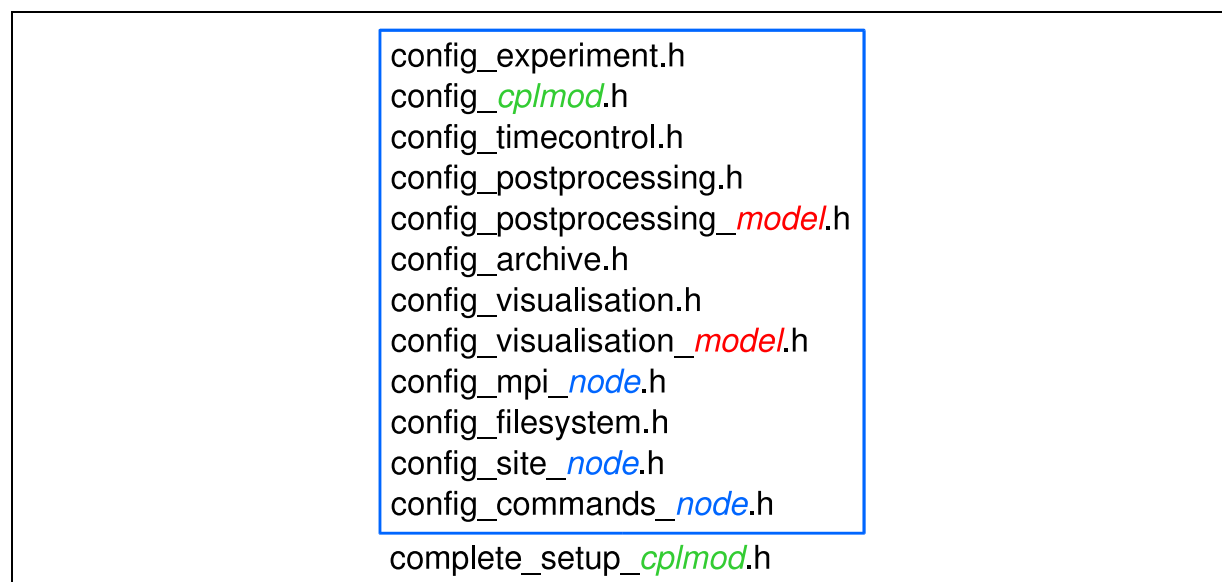


Figure 3.4: Structure of the setup. The file is composed of several header files some of them depending on the coupled model combination (`cplmod`) the component model (`model`) or the site (`node`).

Component models

The include file `config_cp1mod.h` lists all variables defining the component models of the selected coupled combination. Only variables that give a choice are mentioned. For ECHO (ECHAM5 + MPI-OM) 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 as `gridswr` stating whether or not new grid description files should be generated at run time depend on the coupled models flexibility.

Time control

The section time control is identical for all coupled model combinations and all sites. The variables needed for time control are

Calendar Type: The calendar type used by the component models and by the coupler needs to be the same. Possible options are the Gregorian calendar with leap years, without leap years (365 days per year) or a calendar with equal month of any length. This corresponds to the choices given by the OASIS3 coupler.

Initial Date of the Experiment: An experiment usually spans a long (simulated) time period and is divided into several runs (or chunks) of equal length. The initial date of the experiment is specified using three variables: `iniyear` for the year in which the experiment starts, `inimonth` for the month in which the experiment starts and `iniday` for the starting day in that month. With `iniyear=2000`, `inimonth=01` and `iniday=01` the experiment starts on January 1st 2000.

Final Date of the Experiment: Analogous to the initial date, the final date of the experiment is given by the three variables `finalyear`, `finalmonth` and `finalday`.

Duration of a run (chunk): The duration of a run is specified in the same manner: `nyear` gives the number of years within a run, `nmonth` the number of months and `nday` the number of days. Depending on the calendar type not all combinations of these variables are allowed. If for example you chose a calendar with realistic length of months it is not allowed to select a length of a run with `nmonth` and `nday` different from zero (e.g. `nmonth=1` and `nday=10`). To allow for runs of say 40 days rather specify `nday=40`.

Usage of Restart Files: All the components of the selected coupled model combination 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 model components starts the experiment 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_restart_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.

Postprocessing

The setup only contains a postprocessing section if postprocessing is needed for output data of one of the experiment's component models. The variable `postprocessing` opens this section. It is the switch to define whether or not postprocessing is wanted. The following variables are depending on the postprocessing tool.

Archiving

Initial data for a model run is distributed in a tar-file containing the initial data of all the component models. At runtime this tar-file can either be checked out from the PRISM central cvs repository (`archive_tar=cvs`), or it can be transferred from the archive (`archive_tar=tar_local`). If the initial data already is available no input data tar-file needs to be used (`archive_tar=none`). The initial data tar-files might be specific to a version of the PRISM system. The `tag` allows to specify the version of the initial data tar-file. Default tag is the latest PRISM cvs revision.

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 postprocessing or visualisation take place.

Visualisation

The PRISM standard data format is netCDF. The PRISM visualisation tools only support this format. The parameter `visualisation` determines whether or not a task for low end visualisation will be generated. If no visualisation is wanted (`visualisation=no`) the remaining parameters of the visualisation section are irrelevant. The visualisation options strongly depend on the content of the output files of the component models. This section allows for specifying a list of variables to be displayed as well as level and time information. Besides it is possible to choose between predefined plot types. For a complete list of options please refer to the handbook on visualisation (?).

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. (2004)). For automatic visualisation of these files set `vis_out=yes`.

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 `message_passing_method`, whether or not to use buffered MPI send (`bSEND`) and the number of processors used for the `mpiexec` (`nprocmpi`). On most of the machines this number is zero, but for example fujitsu needs a processor exclusively for the `mpi`.

File Systems

The variables defined in the file system section are the roots of the PRISM standard directory tree (2.1). 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 ghost. 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 is 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 the in- 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 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 there 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 needs not 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 important for the first run of an experiment, only. 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 version of the executables are used.

Platform dependant Specifications

This section gives room for system specific variables. They might not be relevant on other platforms. Generally the header file comes with reasonable defaults.

Commands

In this section some Unix commands are defined. 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.

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>f90</code>	f90 compiler on the compute server. This compiler is used for compilation of the calendar tool. It is not used for model compilation.
<code>ftp</code>	file transfer to a remote machine, possibly without asking questions (-i)
<code>get_compile</code>	receive files from a remote compile server (if different from ftp)
<code>put_compile</code>	put files on a remote compile server (if different from ftp)
<code>get_archive</code>	receive files from a remote archiving host (if different from ftp)
<code>put_archive</code>	put files on a remote archiving host (if different from ftp)
<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 ("").

Table 3.1: List of the Unix commands defined in the setup. The site specific default values of these variables are defined in `config_commands_node.h`.

3.4 Setup and Complete_setup

The setup file contains all configurable parameters. But 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 the `complete_setup` (`util/running/frames/include/complete_setup_cplmod.h`). This file also contains derived variables. The `setup` and `complete_setup` together contain the complete list of variables needed for the `cplmod` tasks. It is shown in table 3.2

3.5 The Functions

To keep the tasks as clear as possible several ksh-functions are used within `Create_TASKS.frm`. The functions are part of the tasks for all models on all platforms. The functionality of the functions is demonstrated below.

Function `get_model_resolution`

This function is called to get the acronym of a grid a model is currently 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 defines the variable `res` which is the grid acronym for a component model and the name of the coupled model combination 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. The tar-files are available from the prism central cvs server. Depending on the user's specifications in the `setup` at the beginning of an experiment the initial data can be exported from the cvs repository `archive_tar=cvs` or it can be transferred from the archiving or compiling host `archive_tar=tar_local`. The initial data will be un-zipped and un-tarred in the input data directories of the specific experiment. If the data already is available in the good directories the user can suppress the data transfer by specifying `archive_tar=none`. 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 namelists 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`) which can be located either on the computing host or 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.

There exist optional input files (e.g. remapping matrices for script interpolation). These files 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.

Predefined Variables	
<code>expid</code>	experiment ID (input parameter of <code>Create_TASKS.frm</code>)
<code>cplmod</code>	coupled model combination (input parameter of <code>Create_TASKS.frm</code>)
<code>node</code>	node name of the computing host
<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>iniyear</code>	initial year of the experiment
<code>inimonth</code>	initial month of the experiment
<code>iniday</code>	initial day of the experiment
<code>finalyear</code>	final year of the experiment
<code>finalmonth</code>	final month of the experiment
<code>finalday</code>	final day 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)
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 the communication with the coupler
<code>bsend</code>	flag stating whether or not to use buffered send with MPI
<code>nprocmpl</code>	number of processors reserved for mpi

Table 3.2: 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`

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 runscript 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 an archiving file system (*archive*) is defined 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. 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 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.

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

Function `ecfil`

The `ecfil` function is used for data transfer from one machine to another via ftp. The function replaces the formerly used shell script of the same name. The function is called from `get_tarfile`, `get_file` and `save_file`. It is used only, if the commands `get_archive`, `put_archive`, `get_compile` and `put_compile` defined in the `setup` do not provoke the usage of an alternative command.

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`. In addition to the source and target file names the function knows a third input parameter. It specifies whether or not the source and target files are located on the same machine (`ftp`) and whether or not the source file shall be removed when archiving was successful (`remove`).

Function `submit`

The function executes the submission of a successional job. It depends on the queuing system and whether or not the job is run interactively.

3.6 The Calendar

Time control of the tasks is given by a calendar tool written in fortran. The runscript contains the code as a here-document (compare `calendar.h`). The calendar is compiled at runtime. The executable (`calendar.x`) is placed in the same directory as the scripts on the computing machine (`experiments/expid/scripts`). The calendar reads input data from standard-in and writes the output parameters to standard-out. The in- and output parameters are listed in table 3.3.

Parameter	Intent	Description
<code>inidate</code>	IN	initial date of the experiment
<code>date</code>	IN	initial date of the current run
<code>nyear</code>	IN	number of years per run
<code>nmonth</code>	IN	number of months per run
<code>nday</code>	IN	number of days per run
<code>caltype</code>	IN	calendar type
<code>prevdate</code>	OUT	final date of the previous run
<code>enddate</code>	OUT	final date of this run
<code>nextdate</code>	OUT	initial date of the next run
<code>previnidate</code>	OUT	initial date of the previous run
<code>days_in_run</code>	OUT	number of days in the current run
<code>days_since_start</code>	OUT	number of days since beginning of the experiment
<code>date_in_days</code>	OUT	number of days since beginning of year 0.

Table 3.3: List of in- and output parameters of the calendar tool.

Except for the initial date of the current run all input parameters are available from the user defined setup (compare section 3.3). 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.

Chapter 4

Enlarging the System

One of the aims of the PRISM project was to facilitate the assemblage of new coupled models and to minimize 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 3.2.

The PRISM SCE handbook (Legutke and Gayler (2004)) describes how to adapt model source code to the SCE. This chapter demonstrates the adaption of a coupled model to the SCE (section 4.1) and to run it on a new site (section 4.2).

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>.

4.1 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 3.2). 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 recognized by a common syntax: They all begin with an underscore (`_`) and one capital character followed by lower case characters.

4.1.1 Adaptation of `Create_TASKS.frm`

One of the input parameters of `Create_TASKS.frm` is the coupled model configuration (`cp1mod`). 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 couple model name is realized 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 4.1. Please use lower case characters for the component model names.

Variable	Description
atmmod	atmosphere model
chemod	atmospheric chemistry model
srmod	surface model
ocemod	ocean model
icemod	ice model
bgcmmod	bio-geo-chemistry model
coupler	coupler

Table 4.1: 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.

4.1.2 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 dependant include files are located in directory `util/running/frames/include_node`, the files relevant on all platforms can be found in directory `util/running/frames/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 3.3. 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 3.3.

complete_setup_cplmod.h: Setup variables that are not configurable for the coupled model are listed here (compare section 3.3).

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 dependant include file. Commands to generate a profiling protocol can be added.

4.1.3 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_visualisation_model.h: As the file above `config_visualisation_model.h` is used to generate the setup (compare section 3.3). Variables needed for visualisation of the component models output files are listed. Not included are variables needed or the visualisation 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 `visualisation_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 3.3 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 dependant 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 here. The check guarantees that if a user wants to start from a restart file the file name is defined.

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 3.5). 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 runscript 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 3.5). 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 runscript and part of the archiving script.

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 dependant 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.

visualisation_cplmod.h: The include file is part of the visualisation task. It calls a python script for low end visualisation called `LE_parameter.py`. The script and some environment variables need

to be set on the visualisation host. For information on low end visualisation please consult the relevant PRISM report (?).

4.1.4 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.

```
> 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_ocv_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
```

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.

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 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 2 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.

4.1.5 Providing adjunct files

Some model components have namelist like input files in ASCII format. These adjunct files are not distributed within the input data tarfiles 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's Guide (Valcke et al. (2004)). 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 runsript can be found in the include file `namelist_oasis3_cplmod.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 4.2).

Variable	Description
<code>#Nmseq</code>	Maximum number of fields exchanged sequentially
<code>#Channel</code>	Message passing method
<code>#Mod1procs,</code> <code>#Mod2procs,...</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, #Dto2a, ...</code>	exchange time step in seconds
<code>#Stat_field01,</code> <code>#Stat_field02, ...</code>	Status of the exchange fields
<code>#Cnfileow,</code> <code>#Cnfileaw</code>	Restart filenames
<code>#Lono, #Lato, #Lona,</code> <code>#Lata</code>	Field dimensions
<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, #Lago2a, ...</code>	Time lag of field exchange
<code>#Extrapwr</code>	Enforce rewriting of the extrapolation matrix

Table 4.2: 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 3.3 and in the OASIS3 User's Guide Valcke et al. (2004).

4.2 Adaptation of the SRE to a new Site

Running `Create_TASKS.frm` on a new machine or specifying a new node name as third input parameter leads to the error message:

```

unknown machine: xxx
  No include files exist for a machine with node name xxx

List of machines to which the running is sofar adapted:
node-name      machine
barolo         NEC SX-6
cs             NEC SX-6
dcm13          SGI IRIX64
dcm23          SGI IRIX64
dcm25          Altix 3000
elnino         SGI IRIX64
hpca           IBM Power4
mercure        NEC SX-6
p1             SGI Origin 3800
total1        Linux Opteron
uqbar         NEC SX-5
xbar          fujitsu VPP

```

Standard output of `Create_TASKS.frm` if an unknown node name was given as third input parameter (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 dependant 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 dependant 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.

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 dependant 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.

4.2.1 Node Dependant Include Files

Here a list of the site dependant 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 3.3. 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 dependant 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 queuing 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 dependant.

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.

Chapter 5

Running a coupled model Step-by-Step

This chapter gives a brief to-do list for running a coupled model within the SRE. It is assumed that the executables of all model components and of the coupler are available in the standard directory (*architecture/bin*) on the compile server having the standard names (*model[_submodel]_cmpvers.-MPI[1,2].x*). For details on model compilation please check the PRISM SCE handbook (Legutke and Gayler (2004)). Besides it is assumed that all input data is available either as a tar file in *data/cplmod* (on the compile server or in *archive_in*) or as separate files distributed over the component models' input directories (compare section 4.1.4).

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

1. Go to the directory *util/running/frames*
2. Create the include files for the coupled model combination if they do not exist in directory *util/running/frames/include* (section 4.1).
3. Create the include files for your platform if they do not exist in directory *util/running/frames/include_node* (section 4.2).
4. Run the script *Create_TASKS.frm* to generate a setup file for your experiment. Type *Create_TASKS.frm --help* for help. More information is given in section 3.2.
5. The setup file (*util/running/frames/setup/setup_cplmod_expid*) 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 of your experiment (section 3.3).
6. Run *Create_TASKS.frm* a second time 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.
7. Correct the setup if necessary and run *Create_TASKS.frm* again until the setup check is passed successfully and the tasks (i.e. *runscript*, ..) are created. They are available in directory *experiments/expid/scripts* on the compute server.
8. Login on the compute server (if different from the compile server).
9. Change to the directory *home/expid/scripts*. (The file system *home* was defined in the setup.)
10. Submit the runscript *RUN_cplmod_expid*.

The current state of the experiment can be observed in the working directory *work/expid/work*. At the end of each run the output files are moved to the *data* or *archive* file system depending on the setup. The runscript resubmits itself for the execution of the follow-up run until the end of the experiment is reached. If applicable, scripts for postprocessing, visualisation and archiving are launched automatically at the end of each run.

Chapter 6

Using the GUI

6.1 Configuring an Experiment with PrepIFS

6.2 Monitoring an Experiment with SMS

Bibliography

- Gayler, V. and S. Legutke, 2004: The PRISM Standard Running Environment Guide', PRISM Report Series, no. 4, nn pp. (http://prism.enes.org/Results/Documents/PRISM_Reports/Report4.pdf).
- Legutke, S. and V. Gayler, 2004: The PRISM Standard Compilation Environment Guide', PRISM Report Series, no. 3, nn pp. (http://prism.enes.org/Results/Documents/PRISM_Reports/Report5.pdf).
- Mangili, A., M. Ballabio, M. Djordje, L. Kornbluh, 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)
- Valcke, S., A. Caubel, D. Declat, and L. Terray, 2003: OASIS3 User's Guide. PRISM Project Report 2, 57 pp., CERFACS TR/CMGC/03/69 (http://prism.enes.org/Results/Documents/PRISM_Reports/Report2_oasis3_UserGuide.pdf).
- Valcke, S., A. Caubel, R. Vogelsang, and D. Declat, 2004: OASIS3 prism.2-2 User's Guide. PRISM Project Report 3, 75 pp., CERFACS TR/CMGC/04/48 (http://prism.enes.org/Results/Documents/PRISM_Reports/Report3_oasis3_UserGuide.pdf).

