

Chapter 3

The PRISM Environments

3.1 The PRISM Standards

The PRISM project aims at the establishment of a climate research network in Europe. An important step towards this goal is the development of an infrastructure including a flexible, easy-to-use, and portable software 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 PRISM software 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 achieve these goals, standards have been defined for all aspects of the PRISM software, starting with the source code storage of component models and ending with the processing of the model diagnostic output.

The PRISM software is highly modularised and gives the user a common look&feel for all activities in the system. Automatic processing and graphical user interfaces (GUIs) are provided.

In detail, conventions have been defined

1. for archiving and versioning of the PRISM software,
2. for the coding of models,
3. for the coding of scripts, and
4. for meta data and file formats of model I/O.

A coupling software system has been developed for

5. the time control of data exchange between models and for interpolation between grids, as well as
6. a model interface library for communication with the coupler process and for communication between component models, and
7. a library for parallel I/O of the models, compliant with the standard PRISM file format and meta data.

The requirements of low maintenance costs, high flexibility, and portability have been met by the development of a base of Unix shell script codes that enables the user

8. to generate compile scripts with a common look&feel specifically for all models and platforms of the system, and
9. to configure them for different coupled constellations by simple keyword specifications.

Similarly, a system of Unix script code has been developed that enables the user

10. to generate execution scripts with a common look&feel for all coupled constellations,
11. to activate optional tasks for data archiving, postprocessing, and visualisation of the output, and

12. to configure the execution scripts for specific model constellations and platforms by few specifications.

This system of Unix script code is at the base of the scripts used with the GUIs as well. Therefore, the user has

13. the same functionality and flexibility for the configuration of models and experiments with the configuration GUI as with the Unix shell scripts.

In addition, the same tasks are used for

14. submission with the supervising SMS GUI as with the Unix scripts.

The flexible setup and automatic processing requires

15. a common way of source code organisation for all component models as well as
16. a common structure for the storage of input and output data for all experiments.

These standards are grouped into several classes which define the PRISM standard environments. The environments are described in detail in various PRISM reports, handbooks, or user guides.

In Section ?? of this chapter, a short summary of the coupling environment, made up of the coupling software and its functionalities is given. It covers items 5 to 7 from the above list.

Section ?? shortly describes the standard compile environment (SCE) which comprises the items related to the generation of executable PRISM models (items 1, 2, 3, 8, 9, and 15).

The last section ?? is about the standard running environment, it deals with items 10, 11, 12, and 16, relating to the configuration, setup, and execution of experiments.

The installation and usage of the GUIs (items 13 and 14), and the file format and meta data definitions (item 4) are not included in this chapter but in Chapter 5 and 6 respectively below.

3.2 The Coupling Software

3.3 The PRISM Standard Compile Environment (SCE)

Because of the large number of component models which have already entered the PRISM system or will be included in the future, main issues are flexibility, a common look&feel, and low maintenance costs. It is obvious that this can be obtained only for a modular system with a high degree of automating and a low level of redundancy.

Automating requires that all objects are structured in a common way. For the compiling process this is achieved by the storage of the source code in a well defined Unix directory tree for all libraries and for all component models, as well as for all compiler output and script code. The structure of the model source code tree is kept as simple as possible. There is, however, no limit to the number of source code directories for a model. This allows to group source code according to a functionality (e.g. cloud physics). The only restriction is that all source code directories of a model are on the same hierarchical directory level. The storage of a model source code in this structure allows to use the tools developed for the generation of makefiles and compile scripts. These scripts gives a common look&feel with every model adapted to the PRISM source code structure.

Scripts for model compilation and execution are specific for the model, the model constellation and the platform. They are assembled from a base of small files, called header files, containing script code fragments. These fragments are specific for a model or a platform or both, or they can be used for all models on all platforms. The maintenance is small, since even for a large number of component models or platforms, there is little redundant code. To include a new platform to the system, only the site dependent header files have to be provided. To adapt a new model only the header files containing model dependent specifications have to be written.

The PRISM system supports the flexible replacement of component models. Although a component model can run within several constellations, only one version of source code is supported. The same version of code has to be used on all platforms and within all coupled model constellations. The configuration of the model code can be handled in different ways. However, it is recommended to use the preprocessor functionality. Standard names for cpp flags are proposed and used for this purpose.

Software archiving in the central PRISM repository is done with the Concurrent Versions System (CVS). For the ease of the user and to prohibit the download of mismatching versions, CVS-modules have been defined for the download of consistent code for all components of the PRISM system needed for an experiment. This includes the SCE, SRE, libraries, coupling software, and input data. Modules exist for each coupled PRISM constellation.

3.3.1 Source Code Management

Because of the large number of component models which have already entered the PRISM system or will be included in the future, main issues are flexibility, a common look&feel, and low maintenance costs. It is obvious that this can be obtained only for a modular system with a high degree of automating and a low level of redundancy.

Automating requires that all objects are structured in a common way. For the compiling process this is achieved by the storage of the source code in a well defined Unix directory tree for all libraries and for all component models, as well as for all compiler output and script code. The structure of the model source code tree is kept as simple as possible. There is, however, no limit to the number of source code directories for a model. This allows to group source code according to a functionality (e.g. cloud physics). The only restriction is that all source code directories of a model are on the same hierarchical directory level. The storage of a model's source code in this structure allows to use the tools developed for the generation of Makefiles and compile scripts. These scripts gives a common look&feel with every model adapted to the PRISM source code structure.

Scripts for model compilation and execution are specific for the model, the model constellation and the platform. They are assembled from a base of script code fragments specific for a model or a platform or both, or they can be used for all models on all platforms. The maintenance is small, since even for a large number of component models or platforms, there is little redundant code. To include a new platform to the system, only the site dependent header files have to be provided. To adapt a new model only the header files containing model dependent specifications have to be written.

The PRISM system supports the flexible replacement of component models. Although a component model can run within several constellations, only one version of source code is supported. The identical version of code has to be used on all platforms and within all coupled model constellations. The configuring of the model code can be handled in different ways, however it is recommended to use the preprocessor functionality. Standard names for cpp flags are proposed and used for this purpose.

Software archiving in the central PRISM repository is done with the Concurrent Versions System (CVS). For the ease of the user and to prohibit the download of mismatching versions CVS-modules have been defined. Modules exist for each coupled constellation and include all that is needed for an experiment. This comprises the model source code and libraries, the toolkit to set up the compilation and execution scripts, as well as the input data.

3.3.2 Compiling

The PRISM SCE provides a flexible set of tools for model compilation. The tools can be used for all models on all platforms provided that the source code is managed in the way described above. PRISM compilation is based on the GNU "make" software. This allowed a design that requires only a minimum of recompilation. Only the modified code and the code depending on it is recompiled after a change. This is particularly helpful in the developing phase.

Makefiles are required for each directory containing compiler input source code. They can be generated with the help of a PRISM tool ("Append_dependencies") which detects the directories that have to be searched for prerequisites. It also generates the prerequisite list for all compilation objects. The dependency checking searches through all source code directories of a model. The order in which compiler output binaries are compiled are controlled only by these prerequisite lists. The order in which the makefiles are called has no impact and there is no need to keep FORTRAN modules in an extra directory.

Compile scripts are generated by a PRISM tool called "Create_COMP_cpl_mod.frm". The only mandatory input parameter for this tool is the coupled model name. It creates a set of compile scripts for all components of the coupled model, specifically configured for the coupled constellation and for the platform. The compilation process loops over all source code directories and runs the make command for the makefiles therein. The makefiles are fully portable. all non-portable parameters are set in the compile script and are exported to the makefiles.

The libraries have one common compile script which is, however, specifically created for each platform. Each time a model is compiled the model's compile script launches the library compile script with the list of all libraries used by the model. This includes model specific as well as general PRISM software libraries. The library compile script checks whether or not the listed libraries are up-to-date. This guarantees the consistency of all software components.

3.3.3 Submodels

The coupling of component models can be done either through information exchange between the model executables or, if the components are assembled into the same executable, by information exchange through the parameter lists of subroutine calls. Any other access of the partner model's source code, besides the calls of submodel routines, is outside the PRISM standards (see ? about the package rule).

For each component model, the compilation process first creates a library containing its loadable binaries. For models that create their own executable (main models), i.e. models using an entry point in one of their own routines, the executables are generated in a last step by compiling the main program and linking the model library to it. A submodel is coupled to such a main model by linking its library as well. Model libraries are configured for the specific constellation. The exchange of submodels is achieved by replacing configured component model libraries.

The PRISM project has defined 6 classes of component models, i.e. atmosphere, chemistry, land-surface, ocean, bio-geo-chemistry and ice models. However one can think of additional classes such as hydrological routing and discharge schemes or cloud schemes. They are defined as parameterisations in the present system. Future developments of the PRISM model software system envisage to realise the exchange of parameterisations in a model in a similar way as the submodel exchange is realised in the present system. The only prerequisite is that the parameterisations meet the package rule.

3.3.4 Adapting a component model to the SCE

The adaptation of a component model to the SCE is achieved in several steps that are described below. For detailed information please consult the PRISM SCE handbook (?).

There is only one sine-qua-non criterion for a component model to enter the PRISM system, and that is the adaptation to the PRISM source code structure. It defines the way model and library source code is stored. If this minimum requirement is not met, the model can not be imported into the PRISM repository nor can the SCE tools be used.

An adaptation activity should start with the retrieval of the latest version of the PRISM TOYCLIM CVS module from the CVS repository which includes OASIS3, the toy models, the PRISM libraries, and the script codes. It is recommended to create the OASIS and toy model compile scripts and makefiles, and compile them to get a feeling how to use the SCE.

The next step of adaptation is the extension of the script code base of header files from which the compile scripts are created to accommodate the new model. To see which header files are needed chose the name of any other model already adapted to the SCE and make a list of header files containing the model's name in their file names. These files can be used as examples. If the platform, the new model is going to run on, is already part of the PRISM system only one platform dependent header file has to be provided. This file contains the compiler options for the new model. Several compile modes may be defined for default compilation, debugging, or profiling. For the adaptation of a new platform, the OS and site specifications have to be provided. These files reside in an extra directory. They are supposed to be used with all models. Therefore they may not contain any model specific features.

The inter-executable field exchange between component models is managed by OASIS and the PSMILE library. The calls of the PSMILE routines for initialisation and termination of the communication, and for the field exchange have to be implemented in the component models. The data a submodel exchanges with a model other than the model calling it should be sent directly by the submodel and not by the main model. This allows to strictly assign model data and tasks to the individual component models. The exact coupling algorithm is defined in a file called "namcouple" that is read by the coupler at runtime. It needs to be provided specifically for the coupled model constellation and the coupling algorithm (compare OASIS user guide (?))

If one of the partner models of the new component was already adapted to the PRISM system the latest version of that model has to be checked out from the CVS repository. It may be necessary to modify the physical interface of the model. The compile-time configuration for the coupled constellation should be done following the SCE design. The models already in the PRISM system must still be able to run in the old constellations. This can easily be achieved with conditional source code activated by the compiler preprocessor. All newly written code should follow the coding conventions as specified in ?.

When all executables are successfully created the coupled model can be tested and adapted to the SRE as described below in Section ?? or in the SRE handbook (?).

If desired, the scripting system can be interfaced to the GUI system. This allows to configure the compile and run scripts and to monitoring their execution through the GUI. For details please read the PRISM Graphical User Interface (GUI) and Web Services Guide (?) or see Chapter 5.

3.4 The PRISM Standard Running Environment (SRE)

The PRISM standard running environment (SRE) is intended to facilitate running a coupled model. A common look&feel for all PRISM coupled models minimises the effort to setup and run coupled model experiments. The standards also help designing and running new coupled models and facilitate porting activities to new platforms.

Within the PRISM project a standard directory tree was defined for the storage of the model source code and binaries (compare Section ??) as well as for the input and output data of an experiment and for the tools for the tasks generation. Details on the directory structure needed for model execution are given in Section ??.

As the SCE, the SRE provides a comprehensive set of utilities to generate standardised 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 dependent. The tasks are assembled from these files using the m4 preprocessor (compare Section ??). This method allows for easy adaptation to new coupled models or new platforms as model and site dependent sections are clearly identified. Extending the SRE for a new model or a new platform is illustrated in Section ??. For further information on the SRE please read the PRISM SRE handbook (?).

3.4.1 Directories of the SRE

The PRISM standard directory tree comprises all that is needed to setup and run a PRISM experiment beginning from the model source code and ending with the output data. This section is focusing on the experiment branch of the directory tree. It is not available from the CVS repository as it is created for each experiment set up by a user.

The experiment directory has subdirectories for the input data of component models, for the executables, for the output data produced at runtime, for the log files, for plots (if visualisation is desired) and for the tasks. Besides, there is a working directory. Each of the experiment directories is independent from the others. Thus it is possible to run several experiments at the same time.

Depending on the site, the experiment directories can be spread over several file systems on several hosts. Six different file systems are defined:

1. **home**: file system hosting the tasks (scripts) on the computing host
2. **data**: file system containing the in- and output data for a run (time horizon of little more than a run) on the computing host
3. **archive**: file system to archive the model output on an archiving host.
4. **archive_in**: file system on the archiving host to archive input data that can be used for several experiments. It might be accessed by several users.
5. **visdir**: file system in which the visualisation software is installed on the visualisation host.
6. **work**: temporary working directory on the computing host.

3.4.2 The Tasks

An experiment is defined by a series of families of tasks. Each family corresponds to the execution of a subperiod of the experiment done by one call of the executables. A family consists of different tasks, with or without dependencies between them. The tasks currently supported within the SRE are integration of the coupled model, postprocessing, visualisation, and archiving of the output data. At the beginning of an experiment, the run-script is submitted. When the model integration of the first run is completed, the run-script submits itself again for the integration of the next run. It may also submit the next task of the family, e.g. a postprocessing task. At the end of the postprocessing the visualisation task is submitted which again submits the archiving script. In case of errors, the archiving script is run again until all files are saved successfully.

* The run-script manages the integration of a coupled model. Beforehand the executables of the models and the input and restart data are transferred to the working directory. After the integration, output data and log files are saved. Finally the run-script resubmits itself for the next run and eventually submits a follow-on task.

* Postprocessing is highly model dependant. So far postprocessing is supported for ECHAM5 only. The postprocessing tool used here is the so called afterburner developed at the Max-Planck-Institute.

* 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 run-script the model output is transferred from the working directory to the "data" file system. This file system should have fast access to the working directory but is not necessarily permanent. The archiving task transfers the output files 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.

* The core of the visualisation task is the script "LE_parameter.py", which makes use of cdat. More information is given in the PRISM Data Processing and Visualisation System handbook (?). Visualisation can be carried out on the compute server or on an extra visualisation host where the visualisation software

is installed. 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.

3.4.3 Generation of the Tasks

Usually the tasks are 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 models. To increase portability and minimise the effort of maintenance, the tasks are assembled from several include files depending on either the coupled model constellation, the component model, the site or both. Others can be used for all models on all sites. The identical include files are used when an experiment is run at the scripting level and when it is run through a GUI.

At the scripting level the assemblage of include files is performed by a script called "Create_TASKS.frm". The script needs at least two input parameters: the name of the coupled model 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" is run. To generate the tasks for another machine, the node name can be given as a third (optional) parameter.

The first call of "Create_TASKS.frm" for a specific model and experiment ID leads to the generation of a setup file. This file contains 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. This editing is not needed with SMS where the user makes the specifications through a GUI.

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, if variables are missing or unknown, the creation of the experiment tasks is refused. In this case the user needs to correct the setup file and run "Create_TASKS.frm" again. When the setup check is passed successfully, the tasks are created and transferred to the scripts directory of the experiment on the computing host.

3.4.4 Extending the SRE

The ksh-script "Create_TASKS.frm" manages the assemblage of tasks from header files that are specific for the coupled model and the site (section 3.3.3). To add a new model to the SRE one has to provide all model specific header files for the new model. This includes header files for each new component model and for the new coupled model constellation. Analogously, to adapt the system to a new site, the site dependant header files need to be provided.

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. The include files specific for a coupled model can be identified from the extension "_cplmod", where cplmod represents the coupled model name. Analogously include files specific for a component model can be identified from the suffix "_model" with "model" being the component model name and files specific for the site have the suffix "_site".

Providing the Input Data

Initial data for a PRISM experiment is distributed in tar files. These tar files are available from the central prism CVS repository. Expanding the tar file leads to the creation of the directory "input" with subdirectories for each component 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 component

models. 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 which is 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 Section ??, the input data is stored additionally in an input data archive. This input data archive can be located on a remote archiving machine and might be used by several users for several experiments with various models. When the input data tar file of a coupled model is expanded at runtime the input data archive is filled automatically.

Providing Adjunct Files

Some component models 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 are therefore stored in an extra directory that profits from CVS version control.

Regular namelists are generated as here-documents at run time. 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. At the time the tasks are generated and transferred to the compute server by "Create_TASKS.frm", the adjunct files are transferred to the input directories of the specific experiment as well.

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 (?). Each of the coupled models uses a special namcouple version. The file 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. Namcouple variables are identified from a leading # followed by a capital letter.

Bibliography

- Guilyardi, E., R. Budich, G. Brasseur, G. Komen, 2002: PRISM System Specification Handbook V.1. PRISM Report Series No 1. , 239 pp.
(http://prism.enes.org/Results/Documents/PRISM_Reports/Report1_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., nn 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. , nn pp.,
(http://prism.enes.org/Results/Documents/PRISM_Reports/Report3_pdf).
- Legutke, S. and V.Gayler, 2004: The PRISM Standard Compilation Environment, PRISM Report Series No 4. , nn pp.
(http://prism.enes.org/Results/Documents/PRISM_Reports/Report4_pdf).
- Gayler, V. and S. Legutke, 2004: The PRISM Standard Running Environment, PRISM Report Series, No 5. , nn pp.
(http://prism.enes.org/Results/Documents/PRISM_Reports/Report5_pdf).
- Constanza, P., C. Larsson, C. Linstead, X. Le Pasteur, and N. Wedi, 2004: The PRISM Graphical User Interface (GUI) and Web Services Guide, PRISM Report Series, No 6. , nn pp.
(http://prism.enes.org/Results/Documents/PRISM_Reports/Report6_pdf).
- Valcke, S., 2004: The TOYCLIM PRISM Coupled Model Adaptation Guide, PRISM Report Series, No 7. , 70 pp.
(http://prism.enes.org/Results/Documents/PRISM_Reports/Report7_pdf).
- Legutke, S. and V.Gayler, 2004: The MPI-M PRISM Earth System Model Adaptation Guide, PRISM Report Series No 8, nn pp.
(http://prism.enes.org/Results/Documents/PRISM_Reports/Report8_pdf).
- Demory, M.-E., 2004: THE IPSL_CM4 coupled model adaptation guide, PRISM Report Series No 9. , nn pp.
(http://prism.enes.org/Results/Documents/PRISM_Reports/Report9_pdf).
- Planton, S., xxxx, 2004: Meteo France PRISM Model Adaptation, PRISM Report Series No 10. , nn pp.
(http://prism.enes.org/Results/Documents/PRISM_Reports/Report10_pdf).
- Kastowski, M., xxxx 2004: MPI Jena PRISM Model Adaptation, PRISM Report Series No 11. , nn pp.
(http://prism.enes.org/Results/Documents/PRISM_Reports/Report11_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).
- Carril, A., R. Budich, S. Valcke, P. van Velthoven, S. Legutke, T. Schoenemeyer, 2004: The wp5 TOYCLIM demo run report, PRISM Report Series, No 13. , nn pp.
(http://prism.enes.org/Results/Documents/PRISM_Reports/Report13_pdf).
- Carril, A., xxxx 2004: The PRISM Demonstration Runs, PRISM Report Series, No 14. , nn pp.
(http://prism.enes.org/Results/Documents/PRISM_Reports/Report14_pdf).

- Carter, M., xxx 2004: The PRISM Data Processing and Visualisation System, PRISM Report Series No 15. , nn pp.
(http://prism.enes.org/Results/Documents/PRISM_Reports/Report15.pdf).
- Mangili, A., M. Ballabio, M. Djordje, L. Kornblueh, R. Vogelsang, and P. Bourcier, 2003: PRISM Software Engineering Process, Coding Rules, and Quality Standard, PRISM Report Series No 16. , 31 pp.
(http://prism.enes.org/Results/Documents/PRISM_Reports/Report16.pdf).
- Larsson, C. and N., Wedi, 2003: PRISM Scripts. Script conventions and architecture.
(http://prism.dkrz.de/Workpackages/WP3i/Documentations/PRISM_scripts_conventions.v1.7.html)