

Table of Contents

CAMxRunner.....	1
See also.....	1
Conventions used in this document.....	2
System Requirements.....	3
Concepts.....	3
Installing the CAMxRunner.....	4
Basic help / Options.....	4
Run Installer -I.....	6
Create a new run -C.....	6
Dryrun -d.....	6
Force deletion of output -F.....	6
Log even if in dryrun -l.....	6
Verbose screen messages -v.....	6
Verbose log messages -V.....	6
Cleanup state database -c.....	7
Allow to run more than one instance in the same run -m.....	7
Set error threshold -t [n].....	7
Stop another Runner from running -s.....	7
Run a specific day only -DYYYY-MM-DD.....	7
Leave temporary files in CXR_TMP_DIR -L.....	8
Change number of parallel workers -P[n].....	8
Do NOT run Model -N.....	8
Run only parts of the system.....	8
Execute Model (default) -x.....	8
Input Preparation -i<module>.....	9
Output preparation -o<module>.....	9
Run pre-run step -p<module>.....	9
Run finish step -f<module>.....	9
Using the interactive features.....	9
Creating a new model run.....	9
Starting a new run.....	10
Things to check before running.....	10
Ready to go?.....	10
Adding new config file to repository	11
Naming conventions for run names.....	11
Best practice for using CAMxRunner.....	11
Manage different model versions.....	11
The Configuration.....	12
The chemparam file.....	12
Expansion of the Base Configuration.....	13
Variable names.....	13
File Rules (core concept).....	13
The Task architecture (parallel execution).....	15
The directory structure.....	16
Pre & Postprocessing (Input and output preparation).....	19
Conditional execution of per- and postprocessing modules.....	19
Currently available modules.....	20
Installers.....	20
Common functions.....	21
One-Time preprocessing modules (run once at the beginning of a run).....	21
Daily preprocessing modules (run once each simulation day, before the model).....	21
Daily postprocessing modules (run once each simulation day, after the model).....	21
One-Time postprocessing modules (run once at the end of a run).....	21

CAMxRunner

Filed in CAMx, Modelling, tagged CAMxRunner

See also

- Installation Guide for CAMxRunner
 - CAMxRunnerDevelopment
 - CAMxRunner FAQ
 - CAMxRunnerPresentation
 - Create a dummy Run for CAMxRunner

 - CAMxUserGuide
 - CAMx User guide for SA tools

 - CAMxRunnerWebInterface

 - CAMxSampleJobFile
 - CAMx Runs

 - API of current version
-

This document describes the CAMxRunner, a environment that facilitates the use of CAMx and other air quality models, such as PMCAMx.

CAMxRunner is a framework to execute modules. The order in which these modules are executed is determined by a two-digit prefix in their filename.

If the documentation mentions `XX_array_functions.sh`, `XX` is a 2-digit number, because the user can rename it to affect the order modules are loaded. See Dynamic Extensions

The name of a module (exported in the variable `CXR_META_MODULE_NAME`) is derived from the filename: `10_array_functions.sh` is called `array_functions` internally. When referring to a module (e. g. calling only a specific module), the user needs not to worry about the number, he refers to the internal name as well.

(This text refers to the revision 1839+ of the `CAMxRunner.sh`), `CAMxRunner.sh -h` displays the revision number. If in doubt, *use the source*, it is hopefully an enjoyable read :-).

Note: Starting with revision 386, the CAMxRunner also supports other model systems (at the moment CAMx and PMCAMx). However, currently some modules for PMCAMx are still under development.

The CAMxRunner (`CAMxRunner.sh`) is a modular script written in bash containing logic to prepare, run and postprocess air quality model jobs. See Create a run for details on how to start running your jobs.

Here are the highlights of its features - CAMxRunner

- runs CAMx or other modelling systems, together with pre- and postprocessing
- supports the installation CAMxRunner and the compilation of models and support programs
- allows to manage different versions of models in parallel (CAMx 4.42, 4.51 and 5.01 are supported out of the box).
- supports so-called dry-runs, runs for which all checks are done but no processing takes place. This allows to test a simulation run before it is actually performed.

CAMxRunner

- can parallelize parts of a run, while respecting dependencies (this has nothing to do with MPI or OpenMP)
- stores each executed step in an internal state database (DB) so that steps are not repeated involuntarily - also each CAMx.in file generated is stored there for future reference
- checks the configuration settings as far as possible (see VariableNames), even checks limits of the model binary
- checks if input files are accessible (necessary). If they are not, CAMxRunner can check if the file was compressed before and can decompress it (transparent to the user)
- checks if output files are existing (which would be bad - but this behaviour can be enforced by `-F`)
- notifies the user via mail, SMS or Twitter after each simulation day if needed
- supports simulations spanning several simulation months or even run across a year boundary
- Can automatically continue previously started runs (once the cause of a crash are fixed)

An important restriction of the CAMxRunner

The CAMxRunner cannot easily support different naming conventions on a per-grid basis. In other words, it is not OK for the CAMxRunner to have a file called `emiss.stl.12kmsmall.20020603.a1.bin` for grid 1 and `emiss.stl.36km.20020603.a1.bin` for grid 2 because it expects them to contain the grid number instead of 36km and 12km.

The test case for version 4.51 by ENVIRON contains files that have such a naming convention. For these files, a converter script (`rename_inputs4CAMxRunner.sh`) is provided. Note that the original job file supplied with the test case works only with the original filenames.

When the behaviour of a run must be different, its configuration file must be changed and not the Runner itself.

A normal run consists of these steps:

- One-Time Preprocessing
- For each day
 - ◆ Daily Preprocessing
 - ◆ Model Run
 - ◆ Daily Postprocessing
- One-Time Postprocessing


So there is preprocessing which is done once before each run (**One-Time Preprocessing**), and preprocessing which is done repeatedly before each single model run (each day) (**Daily Preprocessing**). The terms for postprocessing are analogous.

A note on the CAMx test case

The CAMxRunner supports the CAMx test case for CAMx 4.51 by providing a run, called **CAMx-v4.51-ENVIRON_testcase**. However, the data files are not installed, they must be downloaded to the `testcase/4.51` directory (this can be done automatically using the CamxRunner installer).

See CAMxUserGuide for details.

Conventions used in this document

- Commands or options only supported in the environment of the PSI are marked with this icon: 
- The shell prompt is indicated like this:

System Requirements

Operating System	Tested on Scientific Linux. Expected to run on any Unix system
Bash	Version 3.x needed
Other requirements	The script is self-contained and able to download and install the model. Some models might require other software, such as IDL
Recommended support software	A good visual Diff tool, such as WinMerge

Concepts

This is a list of the most important concepts used in CAMxRunner in (hopefully) logical order.

Module

The whole CAMxRunner is modular, so that one can easily add new or disable existing functionality.

Module Type

The modules of CAMxRunner are grouped in Module types (each module belongs to exactly one Module Type. These types include: One-Time Preprocessing Modules, Daily Preprocessing Modules, Model Modules, Daily Postprocessing Modules, One-Time Postprocessing Modules, Installers etc. (See List of available modules for a complete list)

Step

A step is one specific call of a module, e. g. "Run Preprocessing Module X for simulation day Y"

Simulation day

The core model is called once for each simulation day (this does not have to be a real day, but normally it is)

Dry run

In a dry run, all checks are executed, but no real action is taken. This allows you to see if a whole run might be successful

Run

All steps of a given model simulation, including pre- and postprocessing. Has a unique name and its own configuration. Basically a collection of tasks.

Instance

Using the -m Option, more than one CAMxRunner instance can work on the same run. This allows to make full use of many computer at the same time, but a network file system is needed.

Rule

Rules are a mechanism to describe a file name pattern in terms of literals and variables that are automatically evaluated once they are needed.

Automatic decompression

If a rule for a filename results in a name that does not exist, CAMxRunner checks if there is another file with an added extension from the list CXR_COMPRESSED_EXT (e.g. .gzip) that exists. If this is the case, the file is decompressed.

Checks

CAMxRunner

CAMxRunner provides many check functions out of the box. For example, input files are checked for presence, as well as output files (so that nothing gets overwritten). Also, the system estimates the amount of space a run will consume and warn the user should it be necessary.

State database

A folder, in which CAMxRunner keeps track of steps already finished. This allows to resume a partially completed run. Can be manipulated using the `-c` option.

Task

A task is a call to a module on a given simulation day. Tasks can depend on each other. A task can only be started, if it has no dependencies or if all dependencies have finished successfully. At any given time, all tasks which dependencies are fulfilled can be run in parallel.

Installing the CAMxRunner

The installation is described in detail in the Installation Guide for CAMxRunner.

Basic help / Options

```
$ CAMxRunner.sh -h
```

```
-----  
Usage: CAMxRunner.sh [options]  
-----
```

```
CAMxRunner.sh - Modular runner for CAMx  
(Comprehensive Air quality Model with extensions)  
and other air quality models.
```

```
Find detailed information here:  
https://wiki.intranet.psi.ch/LAC/CAMxRunner
```

```
This is revision \Rev: 1146 $ of the CAMxRunner
```

```
-----  
Written by Daniel C. Oderbolz (dco)  
CAMxRunner@psi.ch
```

```
-----  
Options:
```

```
-h                shows this screen  
  
-I                Starts the installation of CAMxRunner (interactive)  
  
-d                causes a dry run  
  
-F                ignores existing output files (force)  
  
-l                writes a logfile for dry-runs as well  
  
-v                verbose: really talkative script  
                  (log will be at least an order of magnitude larger!)  
  
-c                cleanup: removes state information  
  
-m                allow multiple instances of the runner on the same run  
                  (experts only - not recommended)  
  
-t<n>            set the threshold for allowed errors (Default ${CXR_ERROR_THRESHOLD}).  
                  a threshold of ${CXR_NO_ERROR_THRESHOLD} ignores errors.
```

CAMxRunner

-s stop this run gracefully (stop all runners executing this run)

-DYYYY-MM-DD execute a specific simulation day given in the form YYYY-MM-DD

-L Leaves tempfiles where they are. Useful for partial runs on compressed in

-Pn activates parallel execution of pre/postprocessing with
max. n concurrent procs. n must be given!

-C Create a new run, you are guided through the process.

-N do NOT run CAMx, only input and output prep

In the following (limiting) options, a module name can be given.
if omitted, all modules of the respective type are run.
Currently, if these options are used, serial execution is implied (irrespective of -P).
These options can also be combined.

-x<module> only runs model modules

-p<module> only "One-Time" preprocessing modules

-i<module> only daily preprocessing step module

-o<module> only daily postprocessing step module

-f<module> only "One-Time" postprocessing modules

Examples:

To create a new run, start:

```
$ ./CAMxRunner.sh -C
```

The script walks you through the creation of the new run.

The script then creates link called "runname" (here, we assume the run is called CAMx-v4.51) to CAMxRunner.sh and a config template in config/CAMx-v4.51-test_run.conf containing exports of all variables needed.

Afterwards, the script is no longer called using "CAMxRunner.sh".
Instead, use the link ~~(CAMx-v4.51-test_run)~~ (CAMx-v4.51-test_run)

Afterwards, run a so called dryrun:

```
$ "runname" -d  
$ ./CAMx-v4.51-test_run -d
```

This will tell you if the run can be successful.

Edit the configuration file if needed and run another dryrun, this time you might want to produce a logfile:

```
$ "runname" -d -l  
$ ./CAMx-v4.51-test_run -d -l
```

If satisfied, run "properly" (automatically creates a logfile)

```
$ "runname"  
$ ./CAMx-v4.51-test_run
```

In some environments, you must use a special program to start a long-running job, here is an example for a AFS/Kerberos environment:

```
# Refresh token
```

CAMxRunner

```
$ klog  
  
# Run  
$ k5run -b ./CAMx-v4.51-test_run
```

Report bugs to CAMxRunner@psi.ch.

Find more info here:
<https://wiki.intranet.psi.ch/LAC/CAMxRunner>

Run Installer –I

Starts the interactive modular installer for the CAMxRunner, the supported models, testcases and support programs. The CAMxRunner will detect if the installation was already run, and if so, where it left off. You can always quit the installation - but running it will leave you with a complete modelling system.

Because the programs that need to be installed are not shipped with the CAMxRunner, an Internet connection and either `wget` or `curl` is needed to run the installation.

Create a new run –C

Interactively helps you to create a new run, makes sure that the new name adheres to the Naming Convention.

Dryrun –d

In this setup, the Runner just tries if everything could work (this does not imply it will work...) This should always be the first thing one runs, because it can prevent stupid issues with missing files etc.

Force deletion of output –F

Normally, the CAMxRunner does not allow to overwrite existing output files. This option overrides this behaviour. Note that a dryrun does not necessarily show which files would be overwritten.

Log even if in dryrun –l

A dryrun normally creates no log in `$CXR_RUN_DIR/log` to reduce clutter, but with this option it will anyway.

Verbose screen messages –v

This option reveals the mechanics of the script by showing lots of details. Can be repeated to show even more details. Normally only needed for debugging.

Verbose log messages –v

Same as above but for the Logfile.

Cleanup state database -c

When a run is started, a simple DB is created in `$CXR_RUN_DIR/state`. It takes note about started and stopped runs and each step executed. If a run crashes before completion is noted in the DB, the CAMxRunner will refuse to repeat the run. Starting the run using -c will clean up this DB and enable reruns.

CAMxRunner will ask you what part of the state database should be cleaned: `all`, `specific`, `tasks` or `none`. (Selecting none in any of the questions in the cleanup process will stop this process without doing any changes to the database.

Here is what these choices mean:

`all`

Delete all state information for this run. Only needed after a dry-run

`specific`

Delete only specific parts - either all steps of one simulation day or all entries of a certain module type or module

`tasks`

Delete only task management information (used for parallel execution)

See also Using the interactive features.

Allow to run more than one instance in the same run -m

Normally, only one CAMxRunner is allowed on a run. Theoretically, throughput can be increased (especially when only doing pre- and postprocessing) by using instances on more than 1 machine (on the same machine, this makes no sense at all). If -m is not given, CAMxRunner will exit if another instance is detected (by means of a `.CONTINUE` file in the `state` directory of the run. This option is especially useful when -P (parallel execution) is used. If -P is not used, the user needs to make sure that the different instances run different modules (by means of -p -i, -x, -o -f and combinations thereof)

Set error threshold -t [n]

If a run produces a lot of errors, it normally makes no sense to continue. There is a built-in threshold of 50 errors before stopping, which can be reset using this option. If you specify 0, the run continues irrespective of the number of errors.

Stop another Runner from running -s

This option deletes **all** `.CONTINUE` files in the `state` directory of the run, which causes all instances of CAMxRunner *acting on this run* to stop as soon as this is detected (can take some time).

Run a specific day only -DYYYY-MM-DD

Runs the specified simulation day only. The user is responsible for making shure that eventual dependencies (like the previous days run) are fulfilled.

Leave temporary files in CXR_TMP_DIR -L

This option turns off the automatic deletion of all temporary files after the run. Especially useful if only a partial run is done.

Change number of parallel workers -P [n]

CAMxRunner is designed to run processing in parallel (theoretically even on more than one server - this is however not well tested). Per default, CAMXRunner spawns as many workers as there are detectable cores on a machine. With the -P option, this number can be changed. Note that high number might incur a considerable performance loss. If P=1, no parallel execution occurs, less than 1 is obviously not possible.

Overrides setting of \$CXR_PARALLEL_PROCESSING and \$CXR_MAX_PARALLEL_PROCS.

Do NOT run Model -N

This option only runs Pre & Postprocessing and skips the model run. Of course, post-processing will fail if there is no model output. Normally, this option is not needed, because CAMxRunner stores what steps where already executed and automatically continues where it left off if started again.

Run only parts of the system

Normally, The runner executes these types of modules (not necessarily in this order):

- One-Time Preprocessing
- For each day
 - ◆ Daily day Preprocessing
 - ◆ Model Run
 - ◆ Daily day Postprocessing
- One-Time Postprocessing

The following options allow to enable specific parts of these tasks. If any of these options in used, all parts of the chain which are not mentioned will **not** run.

therefore

```
$ CAMx-v4.42-bafu3_winter07.run3 -p -x
```

Only runs the (complete) pre-start preprocessing -p and the Model -x

Execute Model (default) -x

Turns on the model (per default the model is on, but it is off if only parts of the system are run or it might have been turned off in the config)

Input Preparation **-i<module>**

Turns on daily preprocessing.

If a module name is given, just the specified step is executed.

Output preparation **-o<module>**

Turns on daily postprocessing.

If a module name is given, just the specified step is executed.

Run pre-run step **-p<module>**

Turns on pre-run preprocessing.

If a module name is given, just the specified step is executed.

Run finish step **-f<module>**

Runs only the specified step of the finish postprocessing.


If a module name is given, just the specified step is executed.

Using the interactive features

Generally, CAMxRunner is designed to be run non-interactively. Some parts, however, require user intervention. To increase usability, care has been taken to use the same usage paradigms in the whole system. Interactive parts include:

- The Installation
- Creation of new Runs
- Manipulation of the state database

Creating a new model run

A **run** is just a symbolic link from run-name  CAMxRunner.sh. Because the Runner determines critical information like the model name and its version from this name (see Naming Convention), it is recommended to let the script create the link for you as described here:

- Create a new run by executing `CAMxRunner.sh -C`, the script will walk you through (you could also start by creating a symbolic link to the runner which is called like your run (here *CAMx-v4.42-bafu3_winter07.run3*), but this is **not recommended**
- do a dry-run

```
CAMx-v4.42-bafu3_winter07.run3 -d
```

- Change the variables in the configuration file `conf/CAMx-v4.42-bafu3_winter07.run3.conf` according to your needs. Particularly, replace `$(uname -n)` with the machine **CAMx** (not any other stuff like

ektraction asf.) runs on.

- Then, rerun dry and create a log in order to inspect it for errors:

```
CAMx-v4.42-bafu3_winter07.run3 -d -l
```

- If you are happy with the dry log, run it "properly". k5run acquires a kerberos token for AFS, therefore it makes sense to refresh the token using klog

```
# Refresh token
klog

# Run
k5run -b CAMx-v4.42-bafu3_winter07.run3
```

Starting a new run

Things to check before running

- Compare the configuration file to a similar run, do the changes make sense (here WinMerge or similar tools are of great help)
- Check the list of output species (CXR_OUTPUT_SPECIES_NAMES), its embarrassing not to find a relevant species in the output
- Look at the Mechanisms, Geometry and solver options chosen
- Look at the disabled modules: is this OK?
- Is the Meteorology ready?
- Are all required Input files in place (especially large files like MOZART?)
- If you repeat a run:
 - ◆ are old output files deleted (otherwise use -F or -S)
 - ◆ when the new run is longer than the previous one: do you need to repeat pre-start preprocessing modules (e. g. convert_input)

Ready to go?

To start a new run, change into the directory containing CAMxRunner.sh and run

```
$ ./CAMxRunner.sh -C
```

The runner will then ask a number of questions:

- Model name (enter a number)
- Model Version (only supported ones are shown)
- A String that will be added to the run name for uniqueness
- If the configuration should be copied or created from scratch
- If yes, which file should be used as template (default is base.conf)
- If no, all relevant settings will be asked

If you copied the configuration from another file, you might need to change some settings such as:

- Modeling period
- Grid specification
- Scenarios for Emissions/Meteorology

- Modules that should be enabled or disabled

Adding new config file to repository

This is only needed if you want to make full use of the version control features of CAMxRunner. With this, you can prevent modules from running if the versions of the Runner and/or the configuration are too low.

```
$ cd conf
$ svn add
$ svn propset svn:keywords "Id"
$ svn commit -m"Added configuration for run"
```

Naming conventions for run names

It is very important that a run name has the correct form, because the CAMXRunner infers the model name and version to use from this name. Use the option `-C` which creates valid run names by asking for the relevant settings - it is not recommended to create runs manually.

The name of a run (and hence the name of the symbolic link to `CAMxRunner.sh`) has this form:

Modelname-vX.YZ-sometext (for example `CAMx-v4.42-bafu3_winter07.run1_11c6`).

Neither Model names nor versions are allowed to contain a hyphen "-", because this is the character separating this information in the run name.

Best practice for using CAMxRunner

- When you want to run a model using a slightly different setup, create a new run using the `-C` option, and use the original runs configuration file as a starting point. This way, you keep both runs separate.
- When repeating a run over a longer period than originally (suppose you ran the first 7 days and now you want to complete the month), remember that you might need to re-run pre-start preprocessing (if e.g. TUV and AHOMAP are run on a less-than-monthly basis)
- The option `-F` helps when repeating a run. Use `-F` when you want to re-create a lot of files.
- Let CAMxRunner generate your documentation using the `CXR_FINISH_MESSAGE_RULE`. For example, generate text you can directly paste into a table, e.g. into a Wiki
- Use Command line arguments sparingly (they are meant for debugging) - they are non-permanent, therefore they are not documented.
- Use a clear naming convention and directory structure for your model out- and input
- Do not use heterogeneous file names (like the ENVIRON Testcase does) - if you have a consistent naming convention across runs, you will be very fast, since you do not need to change your configuration a lot
- When using arrays in a configuration file, make sure to `unset` it first, because it might have been defined earlier in the hierarchy using more elements.
- Use links to safe space, e.g. Re-Use Emission data if a run must be repeated using different BC data
- If you want to safe space, you can compress your (ASCII) input files using `gzip` or `bzip2`. CAMxRunner will detect this and decompress the files if needed.

See also CAMxRunner FAQ

Manage different model versions

As you can see in the directory structure, the CAMxRunner is organised around models and their versions. To save maintenance and storage, scripts and programs used by many different CAMx versions are linked using the UNIX command `ln`. **The philosophy is that the real files reside in the directory corresponding to the lowest supported version, all higher versions contain links.**

The Configuration

CAMxRunner follows the *convention over configuration* approach. Many settings of CAMxRunner come with a sensible default, releasing the user from specifying irrelevant parameters. These defaults are stored in the file `$CXR_RUN_DIR/inc/defaults.inc` and they are loaded before any command line options or configuration files are processed.

Here are some variables that might need your attention:

Name	Comment
<code>CXR_TMP_DIR</code>	Must point to a directory with ample space if you use automatic decompression

When a new run is created, the file `$CXR_RUN_DIR/conf/base.conf` is expanded or copied to `$CXR_RUN_DIR/conf/.conf` (e. g. `CAMx-v4.42-bafu3_winter07.run3.conf`).

The content of `$CXR_RUN_DIR/conf/base.conf` is sourced first, then, `CAMxvX.YZ.conf` (version specific), after that `$CXR_RUN_DIR/conf/<run>.conf` (extended config) is sourced. After this, command line options are processed. Per Default, the base config is expanded.

The initial `base.conf` can be changed during installation using `CAMxRunner.sh -I`.

This way, a hierarchy is built:

Commandline → extended config → version specific config → base config.

Once an extended configuration is derived from a certain `base.conf`, it might happen that the `base.conf` develops further while the extended configuration does not. This is shown to the user by extracting the revision numbers of both the base and the extended config. A warning is issued if these revisions differ. See Version Control Functions for details.

If you change the value of a variable in `base.conf` make sure you change the derived extended configurations as well - else your setting is overwritten as soon as the extended configuration is loaded. Of course this is not permanent, but your run will never see the changed value.

The chemparam file

CAMxRunner will automatically select the chemparam file according to these rules in the directory `${CXR_CAMX_DIR}/chemparam` (e. g. `bin/CAMx/4.51/chemparam`):

- if a file called `${CXR_RUN}_chemparam` (e. g. `CAMx-v4.51-co5-s160-sem063-run1_chemparam`) is found, it is used, this is useful if your run needs a special file
- if not, a file called `CAMx${CXR_MODEL_VERSION}.chemparam.${CHEMICAL_MECHANISM}_${AEROSOL_MECHANISM}` (e. g. `CAMx4.51.chemparam.6_CF`) is sought
- if not, a file called `CAMx${CXR_MODEL_VERSION:0:3}.chemparam.${CHEMICAL_MECHANISM}_${AEROSOL_MECHANISM}` (e. g. `CAMx4.5.chemparam.6_CF`) is sought

If none of these files is present, the run will stop.

You also have the option of specifying your own chemparam file in the variable `$CXR_CHEMPARAM_INPUT_FILE`, but this is not recommended.

Expansion of the Base Configuration

The expanded configurations you get should be self-contained and document a run. Therefore, per Default, the `base.conf` is expanded when a new expanded configuration is derived. That means, all variables that are not protected by single quotes will be replaced by their actual value in the current context. This also illustrates directly the value of these variables and makes the configuration file easier to follow.

This behaviour can be changed by setting the variable `CXR_EXPAND_BASE_CONFIG` in `base.conf` to `false` (you will be asked about this during the installation)

Variable names

The code depends on a naming convention for variables because it reads variables dynamically from the environment using `env`. These are the rules: Therefore all relevant variables are exported to the environment. If you want to run more than 1 instance of the CAMxRunner, you need to run them from different shells. (This is not well tested, and not recommended, though...)

Exporting of arrays is not directly supported by bash, that is why the functions `export_array` and `import_arrays` provide exporting single arrays and importing all known arrays. (Defined in `XX_array_functions.sh`)

- All Variables names (except system variables like `OMP*` which are external to the CAMxRunner) start with `CXR_`
- Names of rules end with `_RULE`
- Output File names end with `_OUTPUT_FILE`
- Input File names end with `_INPUT_FILE`
- Arrays of files are called `_OUTPUT_ARR_FILES` AND `INPUT_ARR_FILES`
- Directories with `_DIR`
- Output directories with `_OUTPUT_DIR`
- Executables with `_EXEC`
- Strings which are used as Floating point numbers, but have an integer value, need a trailing `.`
- Arrays must have an index 0 with a Dummy entry, we work in Fortran Land here.
- entries in arrays which contain spaces must be protected by single (`'`) quotes, because arrays are exported as space separated list
- Variables normally need to be put into the environment using `export` (else, they will not show up in the output of `env`). All variables in the `*.conf` scripts need this treatment. For arrays, the function `export_array` (see Array Functions) must be used.

File Rules (core concept)

To allow for maximum flexibility when naming files, so-called *file rules*, which are variables containing (combinations of) other variables. This only works if the values are written in **'single quotes'**, else, the variables would get evaluated right where they are defined.

This, for example, allows to define a pattern for a file name. For example our MM5-Output produces file names of this form:

The chemparam file

CAMxRunner

20070101/s151/camx_zp_domain1_uw3_s151:2007-01-01_s. This pattern can be described in terms of internal variables (described later) to yield:

```
CXR_PRESSURE_FILE_RULE='${CXR_YEAR}${CXR_MONTH}${CXR_DAY}/${CXR_MM5_SCENARIO}/c
```

When this rule is *evaluated*, the resulting value depends on the context. So if the current simulation day is 02, this will be the corresponding value. When the rule is evaluated later, the resulting value changes accordingly.

The system automatically turns rules into corresponding variables, so in our example, there will be a variable called **CXR_PRESSURE_FILE** with the currently valid value.

These are some examples of input file name rules (there are also output file rules asf.):

```
xport CXR_AHOMAP_OZONE_COLUMN_FILE_RULE='L3_ozone_omi_${CXR_DATE_RAW}.txt'
CXR_LANDUSE_FILE_RULE='${CXR_INPUT_DIR}/terrain_domain${i}_bx3_lucamx.bin'
```

Note the use of the single quotes - if they where missing, the rule would be evaluated on the spot. This means that you will get an error form the shell, because the variables you used (e. g. \${i}) is not defined in this context!

Attention when nesting rules

If you want to use rules that depend on other rules, use the proper quoting. This for example will not work:

```
export CXR_EMISSION_BIN_FILE_RULE='${CXR_EMISSION_DIR}/camx_emiss_domain${i}_${CXR_YEAR}${CXR_MON
export CXR_EMISSION_PREPBOE_BIN_FILE_RULE='${CXR_EMISSION_BIN_FILE_RULE}.prepbioe'
```

The reason is that the second rule is not expanded due to the quotes. The solution is **not to use single quotes when nesting rules**:

```
export CXR_EMISSION_BIN_FILE_RULE='${CXR_EMISSION_DIR}/camx_emiss_domain${i}_${CXR_YEAR}${CXR_MON
export CXR_EMISSION_PREPBOE_BIN_FILE_RULE=${CXR_EMISSION_BIN_FILE_RULE}.prepbioe
```

Always use the rule and not its expanded value for nesting. The reason for this is that the expander:

- Cannot guarantee any order of expansion
- Cannot parse the rule

(All variable names start with CXR, abbreviated here with C) This is an incomplete list of variables known to the CAMxRunner:

Variable Name	Meaning	Example value
C RUN	The name of the current run (determined from the name of the link)	CAMx-v4.42-bafu3_winter07.run2_lcs15a
C RUN_DIR	The path the script is running in	~/CAMxRunner
i	Denotes the current grid (iterates from 1 .. C NUMBER_OF_GRIDS)	1
j	Denotes the current source group number (only defined where it is needed)	1
C MM5_PROJECT	Label	uw3
C MM5_SCENARIO	Label	s151

CAMxRunner

C EMISS_PROJECT	Label	C MM5_PROJECT
C EMISS_SCENARIO	Label	sem050
C CAMX_SCENARIO	Label	C MM5_SCENARIO
C CAMX_PERIOD	Label	winter_C_ YEAR_S
C CAMX_CUSTOMER	Label	bafu3
C MODEL_HOUR	The current model hour (updated once a model day)	168
C CAL	The current simulation day in DD format	01
C DOY	The current simulation day, day of year (01. Jan =1 asf.) DD format	01
C WOY	The current simulation day, week of year (01. Jan =1 asf.) WW format	01
C JUL	The current simulation day, as Julian day	2454802
C YESTERDAY	The previous simulation day in DD format	31
C CAL_METEO	The simulation day in DD format, the first day is called 01_s @PSI	01_s
C YEAR	The simulation year in YYYY format	2007
C YEAR_S	The simulation year in YY format	07
C MONTH	The simulation month in MM format	01
C BASE_DIR	The base directory	~/CAMX4.4/_C_ CAMX_CUSTOMER/_C_ CAMX_PERIOD
C INPUT_DIR	Input base directory	C BASE_DIR/Inputs
C METEO_DIR	Meteo Input directory	/mnt/other/lacfs01/jkeller/linuxmm5/out
C EMISSION_DIR	Emission Input directory	C BASE_DIR/Emiss
C PTRSCE_DIR	Pointsource Input directory	C BASE_DIR/Ptsrce
C OUTPUT_DIR	Output directory	C BASE_DIR/Outputs/_C_ CAMX_SCENARIO
C PA_OUTPUT_DIR	Process analysis Output directory	C BASE_DIR/Outputs/_C_ CAMX_SCENARIO/PA
C AQMFAD_OUTPUT_DIR	AQMFAD Output directory	C BASE_DIR/Outputs/_C_ CAMX_SCENARIO/aqmfad

Of course, we can also access system variables or even call programs like `$ (uname -n)`. You also have access to the whole API of the CAMxRunner, for example you can format numbers quite easily:

```
export CXR_EMISSION_BIN_FILE_RULE='${CXR_EMISSION_DIR}/camx_emiss_domain${cxr_common_n_digits ${i
```

The code above makes sure that the domain number is always 3 characters wide, padded with 0, e. g. 001.

The Task architecture (parallel execution)

- A run will only stop if a task should be run that depends on a task that failed

The directory structure

Most directories contain a **README.txt** which contains a detailed description of the content of the directory in question.

- **CAMxRunner.sh** The Runner script
- **CAMx.in** This is the control file for CAMx. It is actually a link to a file in the state directory (each daily control file is kept there)
- **CAMx-v4.51-ENVIRON_testcase** This is a **run**, in this case of the testcase for 4.51. It is simply a link to CAMxRunner.sh
- **bin** Contains the CAMx code & Binaries and some helpers
 - ◆ icon:<https://wiki.intranet.psi.ch/pub/LAC/CAMxRunner/folder.gif> **NaturalDocs** Contains the code to generate the API documentation
 - ◆ **CAMx**
 - ◇ 4.42
 - **airascii-x86_64-linux** A converter binary (name depends on configuration)
 - **CAMx-OMP-None-x86_64-linux** CAMx 4.42 binary (name depends on configuration)
 - icon:<https://wiki.intranet.psi.ch/pub/LAC/CAMxRunner/folder.gif>
 - **chemparam** The chemical parameter input files for this version
 - **src**
 - **CAMx** The source of CAMx 4.42
 - **airascii** The source of the converter
 - **airconv** The source of the converter
 - **bin2asc** The source of the converter
 - **uamvascii** The source of the converter
 - **uamvbinr** The source of the converter
 - ◇ 4.51
 - **airascii-x86_64-linux** A converter binary (name depends on configuration)
 - **CAMx-OMP-None-x86_64-linux** CAMx 4.51 binary (name depends on configuration)
 - icon:<https://wiki.intranet.psi.ch/pub/LAC/CAMxRunner/folder.gif>
 - **chemparam** Settings for the Chemical mechanisms
 - **src**
 - icon:<https://wiki.intranet.psi.ch/pub/LAC/CAMxRunner/folder.gif>
 - **CAMx** The source of CAMx 4.51
 - **Links to converter sources of 4.42**
 - ◆ **PMCAMx**
 - ◇ icon:<https://wiki.intranet.psi.ch/pub/LAC/CAMxRunner/folder.gif> **2008**
 - **conf** The Configuration for the CAMxRunner (base.conf, CAMx version specific configuration and the configs for each run)
 - ◆ **base.conf** This is the base configuration, each run has a config that was derived (copied) from this file
 - ◆ **installer.conf** This is the configuration for the installer, currently empty
 - ◆ **CAMx-v4.42.conf** This is the 4.42 version specific extension of base.conf
 - ◆ **CAMx-v4.51.conf** This is the 4.51 version specific extension of base.conf
 - ◆ **CAMx-v4.42-bafu3_winter07.run1_llc6.conf** The config for the job CAMx-v4.42-bafu3_winter07.run1_llc6
 - ◆ **CAMx-v4.42-bafu3_winter07.run2_lcl5a.conf** The config for the job CAMx-v4.42-bafu3_winter07.run2_lcl5a
 - ◆ **CAMx-v4.42-bafu3_winter07.run3_lcl5a.conf** The config for the job CAMx-v4.42-bafu3_winter07.run3_lcl5a
 - **doc** Contains the Documentation
 - ◆ **CAMxRunner.pdf** - a pdf containing the manual

CAMxRunner

- ◆ icon:<https://wiki.intranet.psi.ch/pub/LAC/CAMxRunner/folder.gif> **api** This directory contains the description of all functions (intended for programmers, created using NaturalDocs)
- icon:<https://wiki.intranet.psi.ch/pub/LAC/CAMxRunner/folder.gif> **inc** Contains important includes for CAMxRunner.sh
- **lib** Dynamic libraries for CAMx are stored here (e. g. HDF-Support)
 - ◆ **CAMx**
 - ◇ 4.42
 - **src** contains the source files of the libraries
 - **hdf** contains the HDF5 source
 - **zlib** contains the zlib source (used for HDF compression)
 - **x86_64** contains precompiled libraries for 64 Bit Intel architecture
 - **hdf** contains the HDF5 binaries
 - **zlib** contains the zlib binaries
 - ◇ 4.51
 - **src** contains the sourcefiles of the libraries
 - **hdf** contains the HDF5 source
 - **zlib** contains the zlib source (used for HDF compression)
 - **x86_64** contains precompiled libraries for 64 Bit Intel architecture
 - **hdf** contains the HDF5 binaries
 - **zlib** contains the zlib binaries
 - ◆ **PMCAMx**
 - ◇ icon:<https://wiki.intranet.psi.ch/pub/LAC/CAMxRunner/folder.gif> **2008** contains possible libraries for PCAMx (none at the moment)
 - icon:<https://wiki.intranet.psi.ch/pub/LAC/CAMxRunner/folder.gif> **log** Logfiles are stored here
 - **modules** The modules reside here
 - ◆ **common** Contains program modules of the CAMxRunner. Model and version specific subdirectories.
 - ◇ 10_string_functions.sh
 - ◇ 11_date_functions.sh
 - ◇ 20_log_functions.sh
 - ◇ 30_version_control_functions.sh
 - ◇ 40_check_functions.sh
 - ◇ 50_state_functions.sh
 - ◇ 60_variable_functions.sh
 - ◇ 70_camx_functions.sh
 - ◇ 80_array_functions.sh
 - ◇ 90_preprocessor_functions.sh
 - ◇ 91_postprocessor_functions.sh
 - ◇ **CAMx** contains model-specific functions
 - icon:<https://wiki.intranet.psi.ch/pub/LAC/CAMxRunner/folder.gif> 4.42 contains version-specific functions
 - icon:<https://wiki.intranet.psi.ch/pub/LAC/CAMxRunner/folder.gif> 4.51 contains version-specific functions
 - ◇ **PMCAMx** contains model-specific functions
 - icon:<https://wiki.intranet.psi.ch/pub/LAC/CAMxRunner/folder.gif> **2008**
 - ◆ icon:<https://wiki.intranet.psi.ch/pub/LAC/CAMxRunner/folder.gif> **installer** contains the installer modules
 - ◆ icon:<https://wiki.intranet.psi.ch/pub/LAC/CAMxRunner/folder.gif> **model** contains the modules that run models
 - ◆ **postproc** Postprocessors, by CAMx Version
 - ◇ **CAMx**
 - 4.42
 - **daily** Postprocessors that are called for each simulation day
 - ◆ 10_prepare_output_dir.sh

CAMxRunner

- ♦ **20_convert_output.sh**
- ♦ **30_run_aqmfad.sh**
- ♦ **40_extract_nabel.sh**
- ♦ **50_concatenate_station_data.sh**
- icon:<https://wiki.intranet.psi.ch/pub/LAC/CAMxRunner/folder.gif>
 once Postprocessing modules that are called once after all is simulated
- icon:<https://wiki.intranet.psi.ch/pub/LAC/CAMxRunner/folder.gif> **4.51**
- ◊ **PMCAMx**
 - icon:<https://wiki.intranet.psi.ch/pub/LAC/CAMxRunner/folder.gif> **2008**
- ♦ **preproc** Preprocessors, by CAMx Version
 - ◊ **CAMx**
 - **4.42**
 - **daily** Preprocessors that are called for each simulation day
 - ♦ **01_boundary_conditions.sh**
 - ♦ **02_create_emissions.sh**
 - ♦ **10_convert_emissions.sh**
- icon:<https://wiki.intranet.psi.ch/pub/LAC/CAMxRunner/folder.gif>
 once Preprocessing modules that are called once before the simulation starts
 - ♦ **00_convert_input.sh**
 - ♦ **10_albedo_haze_ozone.sh**
 - ♦ **20_photolysis_rates.sh**
 - ♦ **40_initial_conditions.sh**
 - ♦ **50_check_input.sh**
- icon:<https://wiki.intranet.psi.ch/pub/LAC/CAMxRunner/folder.gif> **4.51**
- ◊ **PMCAMx**
 - icon:<https://wiki.intranet.psi.ch/pub/LAC/CAMxRunner/folder.gif> **2008**
- **state** Stores the state database - each run has its own directory
 - ♦ icon:<https://wiki.intranet.psi.ch/pub/LAC/CAMxRunner/folder.gif>
 CAMx-v4.51-bafu3-january-2006 An example run directory. (File list truncated)
 - ◊ **_CAMxRunner.sh.CONTINUE** This is the continue file. If it is deleted, the CAMxRunner(s) executing this run will terminate as soon as possible.
 - ◊ **CAMx.20060101.in** This is the CAMx.in file for January 1, 2006 of this run. CAMxRunner links the appropriate file to CAMx.in
 - ◊ **CAMx@20060101.START** The presence of this file indicates that CAMx was started for January 1, 2006
 - ◊ **CAMx@20060101.STOP** The presence of this file indicates that CAMx finished for January 1, 2006
 - ◊ **postprocess_single_day@concatenate_station_data@20060101.START** The presence of this file indicates that the single day postprocessing module *concatenate_station_data* was started for January 1, 2006
 - ◊ **postprocess_single_day@concatenate_station_data@20060101.STOP** The presence of this file indicates that the single day postprocessing module *concatenate_station_data* finished for January 1, 2006
- **templates** Templates
 - ♦ **processor.tpl** A template for a pre- or postprocessing module
- **testcase** contains the testcases
 - ♦ **CAMx**
 - ◊ **4.51** Testcase directory for CAMx 4.51
 - **rename_inputs4CAMxRunner.sh** a small script to rename emission inputs for the testcase (see restriction of CAMxRunner.)

Pre & Postprocessing (Input and output preparation)

In order to keep the CAMxRunner small and extensible, it provides a framework to run various input or output preparation modules. There are 4 categories of such modules:

(The variable names start with either CXR_PREPROCESSOR_, here abbreviated with *CPE* or with CXR_POSTPROCESSOR_, abbreviated with *CPO*, or with CXR_DISABLED_, abbreviated *CD*)

Category	Meaning	Stored in	Variable name for directory	Variable name to disable	Command line option to activate single step
One-Time Preprocessing modules	These Preprocessing modules are run in sequence before a run is started. Can be used to setup day-independent input such as terrain data.	.../once	<i>CPE</i> ONCE_INPUT_DIR	<i>CD</i> ONCE_PREPROC	-pXX
Daily Preprocessing modules	These Preprocessing modules are run in sequence for each simulation day right before the day is simulated. Most input preparation happens here (e. g. Emissions)	.../daily	<i>CPE</i> DAILY_INPUT_DIR	<i>CD</i> DAILY_PREPROC	-iXX
Daily Postprocessing modules	These Postprocessing modules are run in sequence after a run is completed. Most output processing happens here (e. g. Conversion of output).	.../single	<i>CPO</i> DAILY_INPUT_DIR	<i>CD</i> DAILY_POSTPROC	-oXX
One-Time Postprocessing modules	These Preprocessing modules are run in sequence once all simulations are done. Can be used to perform day-spanning output preparation such as the creation of animations.	.../once	<i>CPO</i> ONCE_INPUT_DIR	<i>CD</i> ONCE_POSTPROC	-fXX

These modules are loaded automatically in alphabetical order (taking their two-digit prefix into account) from the directories mentioned above, as long as they have the extension .sh and are bash scripts (this is needed because the scripts are included using source). They must also have a 2-digit leading number like XX_convert_emissions.sh.

Conditional execution of per- and postprocessing modules

As seen above, there are command line options (-i, -o) to just execute specific modules. However, command line options have the disadvantage that they are not persistent (do you remember the command line options of the program you typed 48 hours ago?).

CAMxRunner

Therefore, single modules or a whole group can be turned off by these variables:

Name	Disables	Example
CXR_DISABLED_ONCE_PREPROC	All given one-time preprocessing modules	topconc
CXR_DISABLED_DAILY_PREPROC	All given daily preprocessing modules	create_emissions
CXR_DISABLED_DAILY_POSTPROC	All given daily postprocessing modules	prepare_output_dir convert_output
CXR_DISABLED_ONCE_POSTPROC	All given one-time postprocessing modules	skip_all

If a variable has the value **skip_all**, the whole group is ignored.

But what if you want to execute only specific module of a type? You could add all unneeded modules (all other modules of that type) to the disabled list, but if new modules are added later, this approach may fail (and its cumbersome, too).

Therefore, there is a counterpart to each CXR_DISABLED_ Variable called CXR_ENABLED_ . Modules that are in the enabled list are executed, even if they are disabled. This allows such a construct:

```
CXR_DISABLED_ONCE_PREPROC=skip_all
CXR_ENABLED_ONCE_PREPROC=albedo_haze_ozone
```

This run is guaranteed to execute only albedo_haze_ozone, even as new modules are added to the one-time preprocessing modules.

Important note: Modules are enabled by default. There is no need to use the CXR_ENABLED variables other than to overrule a DISABLED variable.

Currently available modules

Installers

Installers are stored in a hierarchy:

- General installers
- Model specific installers
- Version specific installers

Name	Purpose	Comment
XX_Converter_installer.sh	Compiles Converters for Pre- and Postprocessing (ASCII/Binary)	
XX_HDF_installer.sh	Compiles the HDF Library for CAMx 4.x	
XX_CAMx_installer.sh	Installs CAMx	
XX_PMCAMx_installer.sh	Installs PMCAMx	
XX_Pre_and_Postprocessor_installer.sh	Compiles Pre- and Postprocessors like ahomap, tuv, avgdif	

Common functions



Common functions are stored in a hierarchy:

- General Common functions
- Model specific Common functions
- Version specific Common functions


One-Time preprocessing modules (run once at the beginning of a run)

Name	Purpose	Comment
XX_albedo_haze_ozone.sh	Calculate the Albedo/Haze/Ozone input file from satellite data	
XX_photolysis_rates.sh	Prepare the Photolysis rate file from the AHOMAP file	
XX_topconc.sh	Determine the concentrations of the species at the ceiling of the grid	Requires IDL
XX_initial_conditions.sh	Determine the Initial conditions from MOZART data	Requires IDL

Daily preprocessing modules (run once each simulation day, before the model)

Name	Purpose	Comment
XX_boundary_conditions.sh	Determine the Boundary conditions from MOZART data	Requires IDL
XX_create_emissions.sh	Runs the PSI Emission generator	 PSI only
XX_convert_emissions.sh	Conversion of ASCII Emission data to binary	
XX_run_emifad.sh	Running the preparatory program emifad using minimal storage	 PSI only
XX_check_input.sh	Runs detailed input checks (currently only on landuse)	

Daily postprocessing modules (run once each simulation day, after the model)

Name	Purpose	Comment
XX_prepare_output_dir.sh	Preparation of the aqmfad directory	PSI only
XX_convert_output.sh	Conversion of the output to ASCII	
XX_run_aqmfad.sh	Running the preparatory program aqmfad using minimal storage	 PSI only
XX_extract_station_data.sh	Extraction of model data for measurement stations. For the first day, adds a header	
XX_avgdif.sh	Comparison of 2 average files	Designed for the testcases

One-Time postprocessing modules (run once at the end of a run)

Name	Purpose	Comment
XX_concatenate_station_data.sh	Concatenation of the extracted data to one file per station	

This topic: LAC > CAMxRunner

Topic revision: r127 - 03 Feb 2010 - 15:27:25 - DanielOderbolz

Ideas, requests, problems regarding PSI Wiki? Send feedback