The Singularity CMAQ Container

Contents

Back to ***Web-site Index***

Introduction

You will be running a virtualized system ("the container") for this package on your own server or workstation (the "host machine"). The container has a complete CMAQ working environment for you to use on that virtualized system, without needing to build anything, and not needing to worry about installation of prerequisite software (compilers, libraries, etc.) except for *singularity* itself.

This package has two components:

- A Singularity container ***cmaq.simg*** that contains a virtualized Linux OS, the CMAQ model, its pre-processors and post-processors, the SMOKE emissions model, as well as various "tool", utility and analysis-&-visualization programs (with all *PATH*s and *alias*es already set up for you on the container); and
- A "local" directory ***cmaq_cmaq/*** for your host-machine, that contains various sample scripts for interacting with CMAQ and SMOKE submodels, tools, and other programs on that container, as well as this documentation.

This singularity container acts as a virtual machine with its own operating system (CentOS-7, in this case), and with compilers, libraries, and applications installed on it. Because of that virtualized set-up, all the necessary dependencies are managed within that environment and you do not have to worry about installing the pre-requisites, building the models, etc.—you can just use Singularity commands to run the models on that virtual machine, (almost) no matter what machine and operating system you're using as the host for it.

All modeling components are compiled for the "64-bit medium memory model" (see https://cjcoats.github.io/ioapi/AVAIL.html#medium) so that runs even on very-large grids are supported. Only the tools *VERDI* and *Panoply* should be problematic in this regard.

Installed in this container are:

CMAQ-git of June 10, 2020 (version `5.3.1`)
including *CCTM*,
preprocessors *bcon*, *create_omi*, *icon*, and *mcip*,
postprocessors *appendwrf*, *block_extract*, *combine*, *sitecmp_dailyo3*, *bldoverlay*, *calc_tmetric*, *hr2day*, *sitecmp*, and *writesite*, and
utility programs *chemmech*, *create_ebi*, *inline_phot_preproc*, and *jproc*;

SMOKE-git of June 10, 2020 (version `4.7`)
including run-scripts, and programs *aggwndw, beld3to2 bluesky2inv, cemscan, cntlmat, elevpoint, extractida, gcntl4carb, gentpro, geofac, grdmat, grwinven, inlineto2d, invsplit, layalloc, laypoint, met4moves, metcombine, metscan, movesmrg, mrgelev, mrggrid, mrgpt, pktreduc, saregroup, smk2emis, smkinven, smkmerge, smkreport, spcmat, surgtool, temporal, tmpbeis3, uam2ncf*.

verdi version `2.0_beta`
visualization tool

pave version `3.0-beta`
I/O API / UAM / CAMX data visualization tool, from MCNC and Carlie J. Coats, Jr., Ph.D.

ncview version 2.1.2
netCDF-file visualization tool, from UCSD

panoply
netCDF, HDF and GRIB Data Viewer tool, from NASA

GrADS version 2.0.2
Grid Analysis and Display System, from GMU

NCAR Graphics and ***NCO***-4.7.5
from the University Corporation for Atmospheric Research (who run NCAR for NSF)

gnuplot-4.6.2
command-line driven graphing utility

I/O API-3.2 version *2020-04-11 17:51:44Z*

M3Tools version *2020-04-18 16:10:51Z*

NetCDF-C 4.3.3.1,
and also NetCDF-Fortran 4.2-16, and NetCDF-C++ 4.2-8

gcc-4.8.5 and ***gfortran***-4.8.5
compilers

MPICH-3, **MVAPICH-2**, and **OpenMPI-3**
MPI libraries, compilers, and utility programs, for *gcc/gfortran*

ddd and ***gdb***
GUI and command-line debuggers

nedit-5.7
GUI programming editor, aliased to *xx*

xxdiff-4.0.1
GUI difference tool, aliased to *xd*

findent
Fortran indentation/code-transformation tool

Note that two-way WRF-CMAQ is not supported on this container.

Because the Singularity container itself is an "immutable image", any new data files (etc.) that you create can not "live" in the container but instead must be in directories that you mount from your host-machine onto the container as part of the use of *singularity* to run commands on the container. The supplied scripts give examples of how this works; more information is given in a section below.

On this container are directories

/opt/CMAQ_REPO/scripts/
scripts designed to run CMAQ modeling components

/opt/CMAQ_REPO/bin/
optimized executables for the CMAQ modeling components

/opt/CMAQ_REPO/CCTM/scripts/BLD_CCTM_v531_gcc[dbg]-*/
optimized and debug CMAQ CCTM executables for various MPI versions.

/opt/SMOKE/scripts/run/
scripts to run SMOKE

/opt/SMOKE/Linux2_x86_64gfort_medium/,
/opt/SMOKE/Linux2_x86_64gfort_mediumdbg/,
optimized and debug SMOKE executables

Accompanying this container and installed on your host-machine will be a directory ***cmaq_cmaq/*** with five subdirectories:

Docs/
with this document *cmaq_cmaq.html*, and with configuration-files indicating how this singularity container was configured;

Logs/
for log-files;

Scripts/
sample host-scripts to run *CMAQ* modeling components, SMOKE, or interactive shell *tcsh* on the container. The paradigm is that these scripts set up environment variables (etc.) on the container, then do *singularity exec* of

"worker scripts" that actually run the modeling programs.
Note that *cmaq_* and *smk_* and *singularity-term.csh* scripts also contain batch-queue directives, e.g., for queue/batch usage on the UNC servers *longleaf* or *dogwood*, where *singularity* is only available on the compute-nodes.

Reference copies of these scripts are available in the list below, for you to view or download:

singularity-shell.csh
: Log on to the container from the host command-line (non-batch!).

singularity-term.csh
: Launch an interactive *rxvt* terminal from the container (e.g., from a debug batch-queue)

cmaq_cctm.csh
: Set up environment on the container for a (multi-day) CMAQ CCTM run and then use the container's *run_cctm.csh* to execute that run.

cmaq_appendwrf.csh
: etc...

cmaq_bcon.csh
cmaq_bldoverlay.csh
cmaq_block_extract.csh
cmaq_calc_tmetric.csh
cmaq_combine.csh
cmaq_icon.csh
cmaq_mcip.csh
cmaq_writesite.csh

smk_area_nctox.csh
: Set up the environment to run a (multi-day_ SMOKE area source run on the container.

smk_bg_nctox.csh
: etc...

smk_edgar_HEMI108k.csh
smk_met4moves.nctox.csh
smk_mrgall_nctox.csh
smk_nonroad_nctox.csh
smk_point_nctox.csh
smk_rateperdistance_nctox.csh
smk_rateperhour_nctox.csh
smk_rateperprofile_nctox.csh
smk_ratepervehicle_nctox.csh

For more about Singularity see the Singularity User Guide at https://sylabs.io/guides/3.5/user-guide/index.html

Back to ***Contents***

Host Dependencies:
Requirements for your Host Machine

Your host machine needs to have *Singularity* installed on it. Frequently, Linux vendors will have native Singularity packages available for you to use, so that Singularity installation is easy and painless (*su root; yum install singularity* or *su root; apt-get install singularity*). If not, the Singularity User Guide gives instructions on how to install it on your own system.

Note: on the compute clusters at UNC (and possibly other sites), Singularity is configured to run on the compute nodes only, but not on the login nodes. The *cmaq_cmaq/Scripts-BATCH/* versions of the scripts are intended for this usage, e.g., on the UNC cluster *dogwood*. For other such situations, consult your cluster's systems administrator for instructions on how to run Singularity applications and (for the CCTM) how to select the appropriate MPI implementation.

CMAQ CCTM NOTE: MPI implementation is the sticky point. Because the different MPI implementations are not compatible with each other (*mpirun* from MPICH-3 will not work with a program built with OpenMPI, for example) your host machine needs to be running the same MPI implementation as the CCTM executable on this Singularity container. There are CCTM builds for **three different MPI implementations: MPICH-3, MVAPICH-2, and OPENMPI-3**; script-variable `MPIVERSION` in the *cmaq_cctm*.csh* script selects which of these will be used.

In this container, the only MPI application affected by this is the CMAQ CCTM; all of the other applications in this container are either "serial" or (shared-memory) OpenMP-parallel (some **m3tools** and **SMOKE** programs) and don't need to use *mpirun* at all.

Back to ***Contents***

Directories, Environment Variables, and the Container

There are three (and a half) parts of this issue:

- *Where is modeling software installed?*
- *What directories are mounted from the container's host (and how do you mount additional data directories)?*
- *How do you establish environment variables on the container?*

On the container, modeling software is installed under directory */opt/* (following UNIX tradition for software that has its own directory-hierarchy) in a fashion generally similar to the usual CMAQ, SMOKE, and I/O API directory hierarchies but adapted to the specifics of this container. Here is a selection of relevant parts the top few levels of that installation hierarchy. Note that all the CMAQ related optimized executables are sym-linked to `/opt/CMAQ_REPO/bin/`; all the extra analysis tools, etc., are in `/opt/bin/` or `/opt/ioapi-3.2/Linux2_x86_64gfort_medium/`, which are already in your `PATH` on the container; the container's run-CMAQ-component scripts are in `/opt/CMAQ_REPO/scripts/`, and data in your host machine data-directory `${HOSTDATA}` is generally mounted on your container's `/opt/CMAQ_REPO/data/`; the container's SMOKE scripts are in `/opt/SMOKE/scripts/run/`, and `${HOSTDATA}` is mounted on `/opt/SMOKE/data/`, as indicated below.

Selected CMAQ-container directories and Files:

```
    /data                                 #  extra mount-point, if needed

    /opt/CMAQ_REPO/
    /opt/CMAQ_REPO/bin/                   #  optimized Linux2_x86_64gfort_medium
executables
        appendwrf_v531.exe
        BCON_v531.exe
        bldmake_gcc.exe
        bldoverlay_v531.exe
        block_extract_v531.exe
        calc_tmetric_v531.exe
        CCTM_v531.exe
        combine_v531.exe
        hr2day_v531.exe
        ICON_v531.exe
        mcip.exe
        sitecmp_dailyo3_v531.exe
        sitecmp_v531.exe
        writesite_v531.exe
    /opt/CMAQ_REPO/data/
    /opt/CMAQ_REPO/scripts/               #  run_<something>.csh model-component
scripts
        run_appendwrf.csh
        run_bcon.csh
        run_bldoverlay.csh
        run_block_extract.csh
        run_calc_tmetric.csh
        run_cctm.csh
        run_combine.csh
        run_hr2day.csh
        run_icon.csh
        run_mcip.csh
        run_writesite.csh
    /opt/CMAQ_REPO/tables/                #  time independent ASCII files and
tables
    /opt/CMAQ_REPO/CCTM/
    /opt/CMAQ_REPO/CCTM/scripts/          #  various bldit, run-cctm, etc.
scripts, and CCTM build-directories
        BLD_CCTM_v531_gcc-mpich3/
        BLD_CCTM_v531_gcc-mvapich2/
        BLD_CCTM_v531_gcc-openmpi/
        BLD_CCTM_v531_gccdbg-mpich3/
        BLD_CCTM_v531_gccdbg-mvapich2/
        BLD_CCTM_v531_gccdbg-openmpi/
    /opt/CMAQ_REPO/CCTM/src/
    /opt/CMAQ_REPO/CCTM/src/MECHS/        # namelists and chemical-mechanism
files
    /opt/CMAQ_REPO/DOCS/
    /opt/CMAQ_REPO/POST/
    /opt/CMAQ_REPO/PREP/
    /opt/CMAQ_REPO/UTIL/

    /opt/SMOKE/
    /opt/SMOKE/assigns/
        ASSIGNS.EDGAR.cmaq.cb05_soa.HEMI_108k
        ASSIGNS.nctox.cmaq.cb05_soa.us12-nc
    /opt/SMOKE/data/
    /opt/SMOKE/scripts/
    /opt/SMOKE/scripts/run/
        cntl_run.csh
        qa_run.csh
        smk_run.csh
```

```
/opt/SMOKE/src/
/opt/SMOKE/Linux2_x86_64gfort_medium/
/opt/SMOKE/Linux2_x86_64gfort_mediumdbg/

/opt/ioapi-3.2/
/opt/ioapi-3.2/ioapi/
/opt/ioapi-3.2/m3tools/
/opt/ioapi-3.2/Linux2_x86_64gfort_medium/
/opt/ioapi-3.2/Linux2_x86_64gfort_mediumdbg/

/opt/bin/
    findent
    panoply
    pave
    verdi.sh
    wfindent
```

Selected Host-machine Directories and Files:

Singularity **mounts various directories from the host-machine**; it is in these directories that you will wish to have the container "do its work". Because the container itself is "immutable" (i.e., read-only), **any outputs you create must be in those directories mounted from the host-machine.**

The assumption in the current "execute a CMAQ model component on the container" scripts is that **a single master data-directory `${HOSTDATA}`** on the host should be mounted onto the container's */opt/CMAQ_REPO/data/*: that **master data-directory will have sub-directories** for all of the input data **and for the CCTM output data and logs**. The expected sub-directory structure for the master directory is given below.
Note that this is a unified-and-simplified directory structure used by all of the CMAQ modeling components. The top level subdirectories of `${HOSTDATA}` are grid or case specific subdirectories named for environment variable `${APPL}` (or possibly more than one of these, e.g., for programs *ICON* and *BCON* that are used with nested-grid applications). For consistency's same among all the scripts, and to avoid "brittleness" (failure to work correctly from version to version without having to make detailed script-changes), component names do not have program-version numbers in them—*met/mcip* for example, instead of *met/mcipv5.0*.

```
${APPL}
${APPL}/GRIDDESC
${APPL}/WRF-CMAQ/
${APPL}/WRF-CMAQ/wrf_inputs/
${APPL}/cctm/
${APPL}/emis/
${APPL}/emis/inln_point/
${APPL}/emis/inln_point/othpt/
${APPL}/emis/inln_point/pt_oilgas/
${APPL}/emis/inln_point/ptegu/
${APPL}/emis/inln_point/ptagfire/
${APPL}/emis/inln_point/ptnonipm/
${APPL}/emis/inln_point/ptfire/
${APPL}/emis/inln_point/ptfire_othna/
${APPL}/emis/inln_point/cmv_c3/
${APPL}/emis/inln_point/stack_groups/
${APPL}/emis/gridded_area/
${APPL}/emis/gridded_area/rwc/
${APPL}/emis/gridded_area/gridded/
${APPL}/icbc/
${APPL}/land/
```

```
${APPL}/logs/
${APPL}/met/
${APPL}/met/wrf/
${APPL}/met/mcip/
${APPL}/POST/
```

where in fact for multi-part or multi-grid studies (and particularly for program *ICON*) there may be several sets of these sub-directories, each having its own distinguishing `${APPL}`.

A number of additional directories are automatically mounted by a *singularity* ... command:

`${HOME}`, your home directory
`${PWD}`, the directory from which *singularity* was invoked
`/tmp`, and various system directories

You can also use the

```
--bind <host-machine-directory>:<container-directory>
```

command-line option for the *singularity* commands to specify what additional host-machine directories are mounted on the container, and at what locations. This is how we will normally deal with input and output directories for model-data. For example, if the container is *${CONTAINER}=/work/cmaq.simg*, and the host-directory is *${HOSTDATA}=/work/SCRATCH/CMAQv5.3.1_Benchmark_2Day*, the following command mounts that directory on container-directory */opt/CMAQ_REPO/data* before invoking container-script */opt/CMAQ_REPO/scripts/run_cctm.csh*:

```
singularity exec \
 --bind ${HOSTDATA}:/opt/CMAQ_REPO/data \
 ${CONTAINER} /opt/CMAQ_REPO/scripts/run_cctm.csh
```

Subdirectories of host data-directory *${HOSTDATA}* will be seen on the container as matching subdirectories of the container data-directory */opt/CMAQ_REPO/data*. Here in this example, */work/SCRATCH/CMAQv5.3.1_Benchmark_2Day/2016_12SE1/met/* on the host corresponds to */opt/CMAQ_REPO/data/2016_12SE1/met/* on the container, etc. The full subdirectory structure of the data directory is given above.

Note that each *--bind* command-line option does only one mount-operation; if you wish to mount multiple directories from the host-machine, you need multiple *--bind*s.
Note also that these mounts do not follow symbolic links, so you can't use *ln -s* ...to add sub-directories to them...

To set environment variables in the container, there is a special *setenv* form that is used in the host environment before invoking a *singularity* command—you prefix the desired environment-variable name with `SINGULARITYENV_`. For example, the following sequence in host-script *Scripts-CMAQ/cmaq_cctm.csh*

```
setenv SINGULARITYENV_START_DATE    "2016-07-01"
setenv SINGULARITYENV_START_TIME    0000000
setenv SINGULARITYENV_RUN_LENGTH    2400000
setenv SINGULARITYENV_TIME_STEP      100000
setenv SINGULARITYENV_END_DATE      "2016-07-02"
setenv SINGULARITYENV_APPL          2016_12SE1
setenv SINGULARITYENV_EMIS          2016ff
setenv SINGULARITYENV_PROC          mpi
```

```
setenv SINGULARITYENV_NPCOL         1
setenv SINGULARITYENV_NPROW         3
setenv SINGULARITYENV_CTM_DIAG_LVL  1
```

will set the following environment variables on the container, where they are used to control the container script *run_cctm.csh* (in the above example):

```
START_DATE
START_TIME
RUN_LENGTH
TIME_STEP
END_DATE
APPL
EMIS
PROC
NPCOL
NPROW
CTM_DIAG_LVL
```

Back to ***Contents***

Script Generalities

All of the scripts have been modified not only to fit with the environment of the container, but also for consistency among themselves, for full control via environment variables, to support correct return of execution status, to support a common set of "verbose" options, and to support debugging.

The **sample scripts** from directory *cmaq_cmaq/Scripts-CMAQ/* and *cmaq_cmaq/Scripts-SMOKE/* are of two types:

1. Scripts that use *singularity exec* to run on-container modeling scripts (found in directory/files */opt/CMAQ_REPO/scripts/*csh* for CMAQ components or */opt/SMOKE/scripts/run/*csh*), after setting up data directories mounted from your host machine, and after setting up environment variables used to control those scripts;
2. A script ***singularity-shell.csh*** that (after setting up environment and mounted directories), uses the *singularity shell* command that gives you a *tcsh* session on the container from your host-machine command-line, to allow you to run interactive programs such as *ncdump*, *ncview*, *m3stat* (etc.), *VERDI*, or *pave* that are installed in the container, e.g., for Interactive Tool Use.
   *NOTE that for the UNC servers, singularity is not available from login-node command-lines; the **singularity-term.csh** can be launched into a debug-queue, where it will launch an X-based terminal from the container, to give you that sort of command-line access there.*

For SMOKE scripts using *singularity exec* to run SMOKE applications; see the section below. Note that the standard SMOKE script-structure runs a (potentially large) set of time-independent SMOKE programs, followed by a sequence of per-day runs of a set of time stepped SMOKE programs, and can be quite complex :-)

CMAQ-component scripts using *singularity exec* to run a CMAQ modeling component, say *foo*, need to mount a data-directory `${HOSTDATA}` on your host machine to the expected data-directory `/opt/CMAQ_REPO/data` on the container (using `--bind`), and to establish environment variables (of the form `SINGULARITYENV_<name>`) on the host that *singularity* maps into environment variables on the container, as shown below, to run on-container modeling script *run _foo.csh* for that modeling component:

```
...    set HOSTDATA  = <path for data directory on your host machine>
set CONTAINER = <path for CMAQ container on your host machine>
...
setenv SINGULARITYENV_<name>   <value>
...
singularity exec \
  --bind ${HOSTDATA}:/opt/CMAQ_REPO/data \
  ${CONTAINER} /opt/CMAQ_REPO/scripts/run_foo.csh
set err_status = ${status}

if ( ${err_status} != 0 ) then
    echo ""
    echo "**********************************************************"
    echo "** Error for /opt/CMAQ_REPO/scripts/run_foo.csh       **"
    echo "**    STATUS=${err_status}                              **"
    echo "**********************************************************"
endif

exit( ${err_status} )
```

Note that the on-container modeling scripts always return the exit status (whether from `M3EXIT()` or `SEGFAULT`, or...) of the program being executed, with an error-message to the log if the status indicates failure. This status is further passed back to the *singularity exec* scripts, which also write appropriate error-messages and return the status to their callers.

Generally, the *singularity exec* scripts will echo all output to the screen; to capture it in a log, you will need to re-direct it. For a modeling-component *foo*, if the package is installed under your home directory, that might look like

```
[ cd ${HOME}/cmaq_cmaq/Scripts-CMAQ ]
cmaq_foo.csh >& ../Logs/cmaq_foo.log &
```

For every such *singularity exec* script on your host machine, you will need to **customize** the following shell variables:

`${HOSTDATA}`
: path for data-directory on your your host-machine

`${CONTAINER}`
: path name for the CMAQ container on your host-machine

For batch-queue use of the scripts you may also need to customize the batch-queue parameters.

For the CCTM scripts, you will also need to customize the MPI-version parameter to match the MPI version on your host system

`MPIVERSION`
: `mpich`, `mvapich`, or `openmpi`,

```
setenv  SINGULARITYENV_MPIVERSION  <value>
```

If you want verbose script operation, you can control it with environment variable `CTM_DIAG_LVL` on the container:

- `CTM_DIAG_LVL = 0`: no extra diagnostics [default]
- `CTM_DIAG_LVL = 1`: log the sorted environment, size of executable, and process limits
- `CTM_DIAG_LVL = 2`: full script *echo*

In order to change values of this environment variable on the container, edit the `value` in following line in your *singularity exec* script:

```
setenv  SINGULARITYENV_CTM_DIAG_LVL  <value>
```

If you want a debug-run for a modeling component, the scripts are also set up to support debugging, if requested. You will need to do the following: First, build a debug-executable for that modeling component (except for the CTM, for which a debug-executable already exists on the container), and make sure it is in a directory mounted on the container. Then customize on environment variables `${DEBUG}` and `${EXEC}`, as follows: In the *singularity exec* script, uncomment the two following statements, and fill in the container-side path to that executable:

```
setenv SINGULARITYENV_DEBUG   1
setenv SINGULARITYENV_EXEC    <path to debug-executable>
```

Note that environment variable `SINGULARITYENV_EXEC` can also be used to override the executable for the modeling component that you are running. The value should be the **path on the container** to the executable (after any host-directory mount-operations). Be aware that this facility may well have problems due to shared-library incompatibilities between your host machine and the container's CentOS-7 virtual OS.

Back to ***Contents***

CMAQ CCTM Restructuring

There are optimized and debug **CMAQ executables** for each of three MPI implementations: MPICH-3, OPENMPI-3, and MVAPICH-2. The executables can be found as `CCTM_v531.exe` in the following CMAQ-container directories:

```
/opt/CMAQ_REPO/CCTM/scripts/
    BLD_CCTM_v531_gcc-mpich3/
    BLD_CCTM_v531_gcc-openmpi/
    BLD_CCTM_v531_gcc-mvapich2/
    BLD_CCTM_v531_gccdbg-mpich3/
    BLD_CCTM_v531_gccdbg-openmpi/
    BLD_CCTM_v531_gccdbg-mvapich2/
```

respectively. In all cases, they are compiled for "64-bit medium memory model" (see https://cjcoats.github.io/ioapi/AVAIL.html#medium) so that even runs on very-large grids are supported.

Note that since these are the only CCTM executables (matching exactly the compilers and MPI implementations on the container), other compiler-choices (Intel, PGI, ...) are not supported. The choice of which executable to use (and whether to invoke the debugger on that executable) is controlled by container-environment variables **`MPIVERSION`** and **`DEBUG`**.

The attempt has been made to **re-structure** the CMAQ run-scripts and the CMAQ directories for use with the container. The reasons for this are two-fold: first, for consistency among the CMAQ CCTM, its pre-processors, post-processors, and utility programs; secondly, so that there might be a single "generic" CCTM run-script on the container, **controlled by the following list of environment variables** (each of which has a default, indicated in square brackets [LIKE THIS]):

`MPIVERSION`
: `mpich`, `openmpi`, or `mvapich`, to select MPI version compatible with that of the host-server [mpich]

`PROC`
: processing-mode: `mpi` or `serial` [mpi]

`DEBUG`
: *if this environment variable is defined:* run the model under debug using *ddd*, in which case the run is confined to the first day of the modeling-period.
Note that `PROC=mpi` debugging has not been tested; frequently the interaction between *mpirun* and debugging is flaky. But one may hope :-)

`START_DATE`
: Run starting-date, formatted `YYYY-MM-DD` [2016-07-01]

`END_DATE`
: Run ending-date, formatted `YYYY-MM-DD` [2016-07-02]

`START_TIME`
: Run starting-date, formatted `HHMMSS` [0000000]

`RUN_LENGTH`
: Run duration, formatted `H*MMSS` [240000

`TIME_STEP`
: Output time step, formatted `HHMMSS` [10000]

`APPL`
: Application name (e.g. gridname) [2016_12SE1]

`EMIS`
: emissions case [2016ff]

`NPCOL`
: number of processor-columns in the horizontal domain decomposition [8]

`NPROW`
: number of processor-rows in the horizontal domain decomposition [4]

`CTM_DIAG_LVL`
: script-diagnostics/logging level:
0: no extra diagnostics
1: environment, file, and directory based diagnostics
2; full scripting-*echo*

`RUNID`
: any no-whitespace combination of parameters to identify the run [${VRSN}_gcc_${APPL}]

Optionally, **`GRIDDESC`**
: path for `GRIDDESC` file on the container [*${HOSTDATA}/${APPL}/GRIDDESC* on your host machine; this binds to container-file */opt/CMAQ_REPO/data/${APPL}/GRIDDESC*]

In the *run_cctm.csh* script on the container, **additional CCTM-control environment variables**, e.g., `GRID_NAME, CONC_SPCS, CTM_MAXSYNC, CTM_OCEAN_CHEM,` etc., are not hard-coded (changeable only by editing the script), but are established, with their default values, after the pattern

```
if ( ! $?FOO )  setenv  FOO  BAR
```

which potentially sets the default value of container-environment variable `FOO` to `BAR`; i.e., if `FOO` exists in the container environment, then use its existing value; else use the default `BAR`. Consequently, one can change all the **other CCTM control variables** in the *cmaq_cctm.csh* script, as follows: To put a different value `QUX` for environment variable `FOO` to override these defaults, you need to do a *setenv* of the following form in the *cmaq_cctm.csh* script, prefixing the environment-variable name `FOO` by `SINGULARITYENV_`)

```
setenv  SINGULARITYENV_FOO  QUX
```

The *run_cctm.csh* script makes potentially **multiple single-day CCTM runs**, one for each day from `START_DATE` through `END_DATE`, inclusive.

Note that both the *run_cctm.csh* script and the *cmaq_cctm.csh* script have been re-structured to capture exit-status (from `M3EXIT()` or from other causes of failure, e.g., `SEGFAULT`) correctly; and in case of such a failure, *run_cctm.csh* terminates the current run with a descriptive message immediately if that status indicates error, rather than to go ahead blindly ahead with more runs after a failure.

Back to ***Contents***

CMAQ Pre-processing

bcon

Host-script *cmaq_bcon.csh* sets up control variables

`FIN_APPL`
: *ICON* case, usually the (fine-grid) output-grid name.

`CRS_APPL`
: input *CCTM* case, usually the (coarse-grid) CONC-file input-grid name.

`BCTYPE`
: **`regrid`** for regridding CMAQ CTM concentration files; or
`profile` for using default profile inputs

`GRID_NAME`
: *GRIDDESC*-name for the output grid

`START_DATE`
: Gregorian-style starting date, formatted `YYYY-MM-DD`

`START_TIME`
: Starting-time, formatted `HHMMSS`

`RUN_LENGTH`
: Run duration, formatted `HHMMSS`

Optionally, **GRIDDESC**

path for GRIDDESC file on the container [*/opt/CMAQ_REPO/data/${CRS_APPL}/GRIDDESC*]

mounts a data-directory (which should contain subdirectories for both the input and output grids, and then executes the container-script */opt/CMAQ_REPO/scripts/run_bcon.csh* which runs program *ICON* on the container.

create_omi

deferred to a later date...

If you want to do it yourself, look at the script */opt/CMAQ_REPO/PREP/create_omi/scripts/cmaq_omi_run.csh* on the container, copy it out to a host-machine directory that will be mounted on the container (*${HOME}*?), edit it there, using

setenv SINGULARITYENV_...

for the environment variables), and then using

singularity exec /opt/CMAQ_REPO/bin/create_omi

to execute the program.

icon

Host-script *cmaq_icon.csh* sets up control variables

FIN_APPL

ICON case, usually the (fine-grid) output-grid name.

CRS_APPL

input *CCTM* case, usually the (coarse-grid) CONC-file input-grid name.

BCTYPE

regrid for regridding CMAQ CTM concentration files; or
profile for using default profile inputs

GRID_NAME

GRIDDESC-name for the output grid

START_DATE

Gregorian-style starting date, formatted YYYY-MM-DD

START_TIME

Starting-time, formatted HHMMSS

RUN_LENGTH

Run duration, formatted HHMMSS

Optionally, **GRIDDESC**

path for GRIDDESC file on the container [*/opt/CMAQ_REPO/data/${CRS_APPL}/GRIDDESC*]

mounts a data-directory (which should contain subdirectories for both the input and output grids), and then executes the container-script */opt/CMAQ_REPO/scripts/run_icon.csh* which runs program *ICON* on the container.

mcip

Host-script *cmaq_mcip.csh* sets up the following control variables (using different conventions than the other CMAQ modeling components):

APPL
: run identifier [160702]

CoordName
: 16-character-max coordinate system name, for `GRIDDESC` [LamCon_40N_97W]

GridName
: 16-character-max grid name, for `GRIDDESC` [2016_12SE1]

EXECUTION_ID
: 80-character-max run-identification string ["mcip.exe $APPL $GridName"]

IfGeo
: Use `InGeoFile` input? [F]

LPV
: 0: Do not compute and output potential vorticity
: 1: Compute and output potential vorticity

LWOUT
: 0: Do not output vertical velocity
: 1: Output vertical velocity

LUVBOUT
: 0: Do not output *u*- and *v*-component winds on B-grid
: 1: Output *u*- and *v*-component winds on both B-grid and C-grid

MCIP_START
: UTC starting date&time, formatted `YYYY-MM-DD-HH:MM:SS.SSSS` [2016-07-02-00:00:00.0000]

MCIP_END
: UTC final date&time, formatted `YYYY-MM-DD-HH:MM:SS.SSSS` [2016-07-02-00:00:00.0000]

INTVL
: Output time step (minutes) [60]

IOFORM
: 1: Models-3 I/O API
: 2: WRF-format "raw" netCDF

BTRIM
: number of meteorology "boundary" points to remove on each of four horizontal sides of MCIP domain, or
: **-1** to use explicit window information `X0, Y0, NCOLS, NROWS`, as below.

X0
: output-grid starting column, if `BTRIM=-1` [13]

Y0
: output-grid starting row, if `BTRIM=-1` [94]

NCOLS
: output-grid column-dimension, if `BTRIM=-1` [89]

NROWS
: output-grid row-dimension, if `BTRIM=-1` [104]

LPRT_COL

column for diagnostic prints on output domain
If `LPRT_COL=0` use domain-center column

`LPRT_ROW`
row for diagnostic prints on output domain
If `LPRT_ROW=0` use domain-center row

`WRF_LC_REF_LAT`
Lambert conformal reference latitude [40]
If -999.0, MCIP will use average of the two true latitudes.

for the container, and mounts the data-directory (which should contain subdirectories for both WRF input data and MCIP output data) on the container, and then executes the container-script */opt/CMAQ_REPO/scripts/run_mcip.csh* which runs program *MCIP* on the container.

Back to ***Contents***

CMAQ Post-Processing

appendwrf

Host-script *cmaq_appendwrf.csh* sets up the data directory `${HOSTDIR}`, optionally the container-subdirectories `INDIR` and `OUTDIR` and the basenames `INFILE1, INFILE2, INFILE3` for the three input files and the one output file for *appendwrf*, and then executes the container-script */opt/CMAQ_REPO/scripts/run_appendwrf.csh* which runs program *appendwrf* on the container.

bldoverlay

Host-script *cmaq_bldoverlay.csh* sets up environment variables `START_DATE, END_DATE, APPL, HOURS_8HRMAX` and optionally `MISS_CHECK, SPECIES, UNITS`, mounts the indicated data-directory `${HOSTDIR}`, and then executes the container-script */opt/CMAQ_REPO/scripts/run_bldoverlay.csh* which runs program *bldoverlay* on the container.

block_extract

Host-script *cmaq_block_extract.csh* sets up the data directory `${HOSTDIR}`, environment variables

`APPL`
run identifier name (e.g., grid-name) [2016_12SE1]

`SPECLIST`
Array of species to extract.
`ALL` is supported also. [`"( O3 NO2 )"`]

`TIME_ZONE`
Time Zone (`GMT` or `EST` [`GMT`]

`OUTFORMAT`
Format of input files (SAS or IOAPI) [IOAPI]

LOCOL
starting column for the extraction region [44]
HICOL
ending column for the extraction region [46]
LOROW
starting row for the extraction region [55]
HIROW
ending row for the extraction region [57]
LOLEV
starting lvel for the extraction region [1]
HILEV
ending level for the extraction region [1]
RUNID
Run identifier for the input files [`gcc_${VRSN}_${APPL}`]
INFILES
array of basenames for the input files [`"( COMBINE_ACONC_${RUNID}_201607.nc )"`]
Note that all these files should be in directory `${HOSTDIR}/${APPL}/POST`

for the container, and mounts the data-directory on the container, then executes the container-script */opt/CMAQ_REPO/scripts/run_block_extract.csh* which runs program *block_extract* on the container.

calc_tmetric

Host-script *cmaq_calc_tmetric.csh* sets up the data directory `${HOSTDIR}`, environment variables

APPL
run identifier name (e.g., grid-name) [2016_12SE1]
RUNID
Run identifier for the input files [`${VRSN}_gcc_${APPL}`]
OPERATION
operation to perform - `SUM` or `AVG` [`AVG`]
SPECIES
Array of species to extract.
`ALL` is supported also. [`"( O3 CO PM25_TOT )"`]
INFILES
array of basenames for the input files [`"( COMBINE_ACONC_${RUNID}_201607.nc )"`]
Note that all these files should be in directory `${HOSTDIR}/${APPL}/POST`

for the container, mounts the data-directory on the container, and then executes the container-script */opt/CMAQ_REPO/scripts/run_calc_tmetric.csh* which runs program *calc_tmetric* on the container.

combine

Host-script *cmaq_combine.csh* sets up the data directory `${HOSTDIR}`, environment variables

MECH
Chemical mechanism name [cb6r3_ae6_aq]
APPL
run identifier name (e.g., grid-name) [2016_12SE1]
RUNID
Run identifier for the input files [`gcc_${VRSN}_${APPL}`]
START_DATE
Gregorian-style starting date, formatted `YYYY-MM-DD`
END_DATE
Gregorian-style final date, formatted `YYYY-MM-DD`

for the container, mounts the data-directory on the container, and then executes the container-script */opt/CMAQ_REPO/scripts/run_combine.csh* which runs program *combine* on the container, with one execution for (3-D) concentration files and one execution for (2-D) deposition files for each day from *START_DATE* through *END_DATE*, inclusive.

hr2day

Host-script *cmaq_hr2day.csh* sets up the data directory `${HOSTDIR}`, environment variables

APPL
run identifier name (e.g., grid-name) [2016_12SE1]
RUNID
Run identifier for the input files [`gcc_${VRSN}_${APPL}`]
USELOCAL
Use local time? [N]
USEDST
Use daylight savings time? [N]
PARTIAL_DAY
Partial day calculation (computes value for last day)? [Y]
HROFFSET
constant hour offset between desired time zone and GMT [0]
START_HOUR
starting hour for daily metrics [0]
END_HOUR
ending hour for daily metrics [23]
HOURS_8HRMAX
Number of 8hr values to use when computing daily maximum 8hr ozone (17 or 24) [24]
START_DATE
Gregorian-style starting date, formatted `YYYY-MM-DD` [2016-07-01]
END_DATE
Gregorian-style final date, formatted `YYYY-MM-DD` [2016-07-02]
SPECIES_1
define species&operations
format: comma-list
`"Name,Units,From_species,Operation"` operations: `{SUM, AVG, MIN, MAX, @MAXT, MAXDIF, 8HRMAX, SUM06}` `["O3,ppbV,O3,8HRMAX"]`

INFILES

array of basenames for the input files ["(COMBINE_ACONC_${RUNID}_201607.nc)"]

Note that all these files should be in directory ${HOSTDIR}/${APPL}/POST

for the container, mounts the data-directory on the container, and then executes the container-script */opt/CMAQ_REPO/scripts/run_hr2day.csh* which runs program *hr2day* on the container.

sitecmp

tbd...

Look at the following scripts on the container and the suggestions for scripting *create_omi*, above (or use the *singuilarity-shell.csh* script to run */opt/CMAQ_REPO/bin/sitecmp* interactively):

```
/opt/CMAQ_REPO//POST/sitecmp/scripts/run_sitecmp_AQS_Daily.csh
/opt/CMAQ_REPO//POST/sitecmp/scripts/run_sitecmp_AQS_Hourly.csh
/opt/CMAQ_REPO//POST/sitecmp/scripts/run_sitecmp_CSN.csh
/opt/CMAQ_REPO//POST/sitecmp/scripts/run_sitecmp_IMPROVE.csh
/opt/CMAQ_REPO//POST/sitecmp/scripts/run_sitecmp_NADP.csh

/opt/CMAQ_REPO//POST/sitecmp/scripts/run_sitecmp_SEARCH_Hourly.csh
```

sitecmp_dailyo3

tbd... look at the following scripts on the container:

```
/opt/CMAQ_REPO//POST/sitecmp_dailyo3/scripts/run_sitecmp_dailyo3_AQS
.csh

/opt/CMAQ_REPO//POST/sitecmp_dailyo3/scripts/run_sitecmp_dailyo3_CAS
TNET.csh
```

writesite

Host-script *cmaq_writesite.csh* sets up the data directory ${HOSTDIR}, environment variables

APPL

run identifier name (e.g., grid-name) [2016_12SE1]

RUNID

Run identifier for the input files [gcc_${VRSN}_${APPL}]

START_DATE

Gregorian-style starting date, formatted YYYY-MM-DD

END_DATE

Gregorian-style ending date, formatted YYYY-MM-DD

SITE_FILE

Name of input file containing sites to process, or ALL (i.e., process all cells) [ALL]

USELOCAL
Use local time? [N]
TIME_SHIFT
constant hour offset between desired time zone and GMT [0]
TIME_SHIFT
Shifts time of data from GMT [0]
USECOLROW
Site file contains column/row values? (else Lat-Lon values) [N]
LAYER
grid layer to output [1]
PRTHEAD
Output header records? [Y]
PRT_XY
Output map projection coordinates *X* and *Y*? [Y]
SPECIES_1
Name of species to process [`O3`]
IN_FILE
Base-name for input file [`COMBINE_ACONC_${RUNID}_201607.nc`]

for the container, mounts the data-directory on the container, and then executes the container-script */opt/CMAQ_REPO/scripts/run_writesite.csh* which runs program *writesite* on the container.

Back to ***Contents***

CMAQ Utilities

chemmech

pending...
Or use the *singularity-shell.csh* script to run it interactively...

create_ebi

pending...

inline_phot_preproc

pending...

jproc

pending...

Back to ***Contents***

SMOKE Modeling

The SMOKE programs have all been built for both optimized and debug on the container, using the *gfortran/gcc* compiler set for "medium" memory model (so that even very large data sets are supported); the executables can be found in directories

/opt/SMOKE/Linux2_x86_64gfort_medium/ and *opt/SMOKE/Linux2_x86_64gfort_mediumdbg/*.

The SMOKE scripts have all been re-structured to make correct use of program exit-status (stopping the sequence of execution when there is a failure), and pass that status back through to the caller. They have also been re-structured so that if debugging is requested by means of environment variable `DEBUGMODE`, it will "just work" (using the *ddd* GUI debugger on the container) without requiring extensive and deep hacking of multiple scripts to make it work. In that case, they will only run for the first day of the episode, rather than running the debugger repeatedly for each separate day of a multi-day run-sequence

There are three relevant sets of SMOKE scripts for use with SMOKE on this container:

On-container *ASSIGNS*-scripts in container directory */opt/SMOKE/assigns/* have been modified to set environment variable `SMK_HOME` correctly for this container, and to look at environment variable `DEBUGMODE` and set environment variable `BIN` appropriately for this container: either `Linux2_x86_64gfort_medium` for optimized, or `Linux2_x86_64gfort_mediumdbg` for debug.

On-container runscripts *smk_run.csh, qa_run.csh, cntl_run.csh* in container directory */opt/SMOKE/scripts/run/* have been re-structured so that if an error occurs (whether reported by `M3EXIT()`, or because of `SEGFAULT`, or ...), they will terminate execution the current set of runs immediately and return the exit-status to the invoking script, rather than blindly going ahead and trying to execute everything that follows, irrespective of the failure. They also properly support running SMOKE component programs under the *ddd* debugger without needing the detailed "script-hacking" needed by their predecessors.
These scripts source the relevant *ASSIGNS*-script (passed in from the on-host runscripts as environment variable `ASSIGNS_FILE`) as needed for their execution.

On-host runscripts such as *smk_ratepervehicle_nctox.csh* in host-machine directory *cmaq_cmaq/Scripts-SMOKE/* pass the basename of the appropriate *ASSIGNS*-script in environment variable `ASSIGNS_FILE` to the container, and invoke the appropriate sequence of *smk_run.csh* and *qa_run.csh* there, making use of the returned exit-status from these scripts to further control the run-sequence: it will stop and log an error message for the first program-run that exits with a failing (non-zero) exit status (or else it will run to completion, if everything succeeds).

For debugging, in the appropriate on-host run-script, replace the statement

```
unsetenv SINGULARITYENV_DEBUGMODE
```

by

```
setenv SINGULARITYENV_DEBUGMODE Y
```

and set the other environment variables to ensure that only the one requested modeling-component is run, and that only for the date of interest.

Back to ***Contents***

Interactive Tool Use

See the annotated copy of *Scripts-CMAQ/singularity-shell.csh* at the bottom of this section, below, which sets up an interactive shell-session on the container for you...

Many of the modeling tasks you wish to do are best done interactively, not from "batch". The

> *singularity shell ...*

command allows you to run an interactive shell (e.g., *tcsh*) in the container, frequently by acting on data in a directory mounted from the host-machine, and generating outputs in a(nother) directory mounted from the host-machine (recalling that attempts to write data into the container's file-system itself will fail, with a "permission denied" nasty-gram); you may recall that your *${HOSTDATA}*, your *${HOME}*, and */tmp/* are examples of such directories mounted on the container from your host-machine...

Note that *PATH*s and *alias*es, etc., have already been set up for you on the container; that set-up can be found in the container's */etc/profile.d/local.csh*.

Examples of commands you might want to run interactively include the following applications installed in the container. For the most part, they are installed under */opt/bin/*; they are all on the default path for *singularity shell*. A few of these tools also have *singularity exec* scripts to run them directly on your host machine; these last scripts need to be customized in the same way that the CMAQ host-machine scripts are.

M3Tools programs version `3.2 2020-04-18 16:10:51Z`

> such as *m3cple*, *m3diff*, *m3probe*, *m3stat*, and a variety of others.
> These are probably best run interactively after you invoke *singularity-shell.csh* (or script them in a directory mounted from your host machine, using the principles described above, and invoke the script on the container after doing *singularity-shell.csh* or launching *singularity-term.csh* to a debug-queue).

verdi.sh version `2.0_beta`

> a gridded Java based netCDF data visualization tool from EPA: see https://www.cmascenter.org/verdi/
> Host script: *cmaq_cmaq/Scripts-CMAQ/cmaq_verdi.csh* will directly invoke *verdi* on the container. Edit this script as indicated above, to suit your host machine and data directory situation.
> *verdi* may also be run interactively on the container, after you invoke *singularity-shell.csh* or launching *singularity-term.csh* to a debug-queue
>
> Note that any output from *verdi* (e.g., any image-files you created, or output from `save project` must be in a directory mounted from your host-machine; you may recall that your *${HOME* is one such directory...

pave version `3.0 beta`

> a visualization tool for I/O API / UAM / CAMX data, from MCNC and Carlie J. Coats, Jr., Ph.D.; see https://cjcoats.github.io/pave/PaveManual.html: this version has been re-structured to offer vastly improved performance for large data sets. (It is so much faster that for animations you will probably need to use environment variable `TENTHS_SECS_BETWEEN_FRAMES` to slow down the animations enough that you can interpret them.)

Built for 64-bit-medium memory model, so that usable data set sizes are limited only by available memory (unlike the other vis tools, which tend to have 2GB limits)
*Note also that the **file-selection GUI** fails, due to software versioning problems ("library rot")*; however,

pave [<config>] -f <path to file> ...

does work, where *${config} = 2, 3, 3a, 3b, 3d, 3g, 5, 6, 51, frac, lu, o3, soil, strm, tk* identifies one of the on-container PAVE configuration-files *pave.${config}.config* found in container directory */opt/pave-3.0/Config/*
A number of these use "zebra" color palettes: *pave.3.config*, for example, uses a 5-hue/50-color palette, where the first ten colors are blues with varying saturation ranging from near-white to fully-saturated.
${config} = frac, lu, o3, soil, strm, tk are for the relevant specific variable, e.g., *tk* for TK, Temperature (Kelvin).
pave is probably best run interactively after you invoke *singularity-shell.csh* or launching *singularity-term.csh* to a debug-queue

ncview version 2.1.2
a netcdf-file visualization tool from UCSD; see
http://meteora.ucsd.edu/~pierce/ncview_home_page.html
Host script: *cmaq_cmaq/Scripts-CMAQ/cmaq_ncview.csh*
Edit this script as indicated above, to suit your host machine and data directory situation, or run *ncview* interactively after you invoke *singularity-shell.csh* or launching *singularity-term.csh* to a debug-queue

panoply
a netCDF, HDF and GRIB data viewer tool from NASA: see
https://www.giss.nasa.gov/tools/panoply/
Host script: *cmaq_cmaq/Scripts-CMAQ/cmaq_panoply.csh*
Edit this script as indicated above, to suit your host machine and data directory situation, or run *panoply* interactively after you invoke *singularity-shell.csh* or launching *singularity-term.csh* to a debug-queue.

GrADS
the Grid Analysis and Display System from GMU: see
http://cola.gmu.edu/grads/
GrADS is probably best run interactively after you invoke *singularity-shell.csh* or launching *singularity-term.csh* to a debug-queue

NCAR Graphics
see http://ngwww.ucar.edu/
NCAR Graphics is probably best run interactively after you invoke *singularity-shell.csh* or launching *singularity-term.csh* to a debug-queue

gnuplot
graphics/plotting tool: see http://www.gnuplot.info/
gnuplot is probably best run interactively after you invoke *singularity-shell.csh* or launching *singularity-term.csh* to a debug-queue

ddd and ***gdb***
debuggers: *ddd* is a GUI "wrapper" for *gdb*

These are invoked automatically when requested by the modeling-component scripts; or you can run them interactively after you invoke *singularity-shell.csh* or launching *singularity-term.csh* to a debug-queue

nedit

GUI text editor for interactive use, after you invoke *singularity-shell.csh*
There is an alias *xx* that runs it in the background: e.g., to bring up edit-windows on files *foo*, *bar*, and *qux*, issue the command

xx foo bar qux

xxdiff

GUI file-differencing tool for interactive use, after you invoke *singularity-shell.csh*
There is an alias *xd* that runs it in the background with "ignore-whitespace" command-line options; to see the differences in files *foo* and *bar*, issue the command

xd foo bar

findent

see https://github.com/wvermin/findent
Fortran source indentation and beautification program for both fixed ("f77-style") and free ("f90-style") format; also converts Fortran fixed format to Fortran free format (and vice-versa). It will accept CMAQ and SMOKE's non-Standard "fixed-132" source format.
There is an alias *tof90* that converts fixed-format Fortran source to free format, using the I/O API's indentation conventions, as in the following:

tof90 < prog.f > prog.f90

cmaq_cmaq/Scripts-CMAQ/singularity-shell.csh is an example of a host-system script that

- sets up some environment variables;
- mounts host-machine directories on the container as described above; and
- then runs *tcsh* on the container, giving you an interactive prompt,

for you to use tools (such as those listed above) on the container. The essential content of it is the following, which establishes various container-environment variables `APPL , EMIS`, etc., and then mounts the host directory `${HOSTDATA}` on container-directory `/opt/CMAQ_REPO/data`, and then invokes an interactive *tcsh* session on the container *${CONTAINER}*, and starting from directory `/opt/CMAQ_REPO/data` on the container:

```
    #!/bin/csh -f
    #
    # Script to Invoke "singularity shell" for cmaq container
    #   Data directory on host:  mounts onto container-directory
"/opt/CMAQ_REPO/data"

    set HOSTDATA  = <path for data directory on your host machine>
    set CONTAINER = <path for CMAQ container on your host machine>

    # Examples of setting up environment variables such as APPL and EMIS
    # for the container:
```

```
setenv SINGULARITYENV_APPL          2016_12SE1
setenv SINGULARITYENV_EMIS          2016ff

#  invoke "singularity shell" using bindings of host-directories to
#  container-directories, and starting tcsh at mount-point of ${HOSTDATA}

cd ${HOSTDATA}

singularity shell -s /usr/bin/tcsh \
 --bind ${HOSTDATA}:/opt/CMAQ_REPO/data \
 ${CONTAINER}
```

You will then probably want to do something like the following (at the *tcsh* prompt within the container):

```
verdi.sh
```

or

```
pave -f /opt/CMAQ_REPO/data/${APPL}/met/mcip/METCRO2D_160701.nc \
     -f
/opt/CMAQ_REPO/data/${APPL}/cctm/CCTM_ACONC_v531_gcc_2016_12SE1_20160701.nc
```

or something like the following *m3stat* run (noting that the report-file created by *m3stat* below must be in a host-machine-mounted directory such as `$HOME`; if it's not a directory mounted from the host-system, the system will give you a nasty-gram indicating "permission denied"):

```
cd /opt/CMAQ_REPO/data/${APPL}/met/mcip
ls
setenv AFILE   $cwd/METCRO2D_160701.nc
setenv REPORT  $HOME/METCRO2D_160701.stats
m3stat AFILE REPORT DEFAULT
```

Back to ***Contents***

Send comments to

Carlie J. Coats, Jr.
cjcoats@email.unc.edu