Developer Manual for the Community Multiscale Air Quality (CMAQ) Modeling System

Version 5.2 (2017 Release)

Prepared in cooperation with the
Community Modeling and Analysis System
Institute for the Environment
University of North Carolina at Chapel Hill
Chapel Hill, NC

Disclaimer

The information in this Developer Manual has been funded wholly or in part by the United States Environmental Protection Agency. The draft version of this document has not been subjected to the Agency's peer and administrative review, nor has it been approved for publication as an EPA document. The draft document is currently being edited and reviewed by the Community Modeling and Analysis System Center; this content has not yet been reviewed or approved by the EPA. Mention of trade names or commercial products does not constitute endorsement or recommendation for use.

Introduction

This document is intended to describe general development practices within the CMAQ Modeling Community. The information contained should be read prior to starting a project within the CMAQ framework. Instructions can be used by EPA developers, CMAS-Center developers, or external developers. Notes specific to external developers are made where relevant.

Contributions

To contribute source code to the CMAQ Public repository, create a fork of the EPA CMAQ External GitHub repository. Prospective developers should send the following information to the EPA and CMAS-Center code maintainers.

- What is your GitHub user name? (can be acquired at github)
- Can your work be completed in a fork of the release repository?
  - If no, please explain the reason.
- What science module do you intend to develop?
- What work do you intend to contribute to CMAQ?
- Have you reviewed the development strategy including code consistency, benchmarking, configuration testing, compiler testing, model output validation, documentation and merging?
- Are you able to provide ongoing support and technical guidance for your proposed contribution?

Repository Descriptions

The CMAQ project contains several repositories. Some of the repositories are private for development efforts while some are public for users and releases. These repositories are described below.

The main CMAQ repositories are described in this section

Internal EPA Private CMAQ-Dev Repository

This will be where CMAQ development by EPA employees occurs. This is a private repository for development efforts by employees of EPA.

External EPA Public CMAQ-Release Repository

The main CMAQ release repository is located here. Users should refer to this repository for bug fixes, issues, and major releases for CMAQ.

https://github.com/USEPA/CMAQ

Users who wish to implement a new feature, and have that feature merged into the CMAQ repository should follow the instructions on code requirements and repository layout as an CMAQ developer would. Forks should be made of this release repository. New features will have to undergo a code review before any merge onto the release repository. Instructions on using it to add a feature are in this (insert link) section.

External CMAS Public CMAQ-Release Repository

A fork of the main CMAQ-Release release repository that is used for testing and documenation by the CMAS Center Employees.

Repository Layouts

The CMAQ-Dev and CMAQ-Release repository directories for the base model are all laid out as follows.

- CCTM
- docs
  - Releaase_Notes
  - User_Manual
  - Developer_Manual
- scripts
- src
  - ICL
  - MECHS
  - PARIO
  - STENEX
  - areo
  - aero6
  - biog
  - beis3
  - cloud
  - acm_ae6
  - acm_ae6_kmt
  - acm_ae6_mp
  - acm_ae6i_kmti
  - couple
  - gencoor
  - gencoor_wrf
  - depv
  - m3dry
  - driver
  - wrf
  - yamo

  - emis
  - emis
  - gas
  - ebi_cb05e51_ae6_aq
  - ebi_cb05e51_ae6nvPOA_aq
  - ebi_cb05eh51_ae6_aq
  - ebi_cb05mp51_ae6_aq
  - ebi_cb05tucl_ae6_aq
  - ebi_cb05tump_ae6_aq
  - ebi_cb6r3_ae6_aq
  - ebi_cb6r3_ae6nvPOA_aq
  - ebi_racm2_ae6_aq
  - ebi_saprc07tb_ae6_aq
  - ebi_saprc07tc_ae6_aq
  - ebi_saprc07tc_ae6nvPOA_aq
  - ebi_saprc07tic_ae6i_aq
  - ebi_saprc07tic_ae6invPOA_aq
  - ros3
  - smvgear
  - grid
  - cartesian
  - hadv
  - yamo
  - hdiff
  - multiscale
  - init
  - yamo
  - par
  - mpi
  - par_noop
  - phot
  - inline
  - table
  - plrise
  - smoke
  - procan
  - pa
  - pv_o3
  - spcs
  - twoway
  - util
  - util
  - vadv
  - wrf
  - yamo
  - vdiff
  - acm2
- POST
- appendwrf
- bldoverlay
- block_extract
- combine
- hr2day
- merge_aqs_species

- sitecmp
- sitecmp_dailyo3
- writesite
- PREP
- agdust
- bcon
- icon
- jproc
- mcip
- Tutorials
- UTIL

Development Life-cycle

EPA target schedule for CMAQ includes a final stable release of the base model that is available 6 months prior to the availability of the instrumented versions of the code.

Prior to the final release, the model is released to the public as a development version that is not intended for regulatory or research application use.

The purpose of releasing the development version to the public is to encourage the community to assist with configuration testing, compiler testing, benchmarking and verification of model output to help improve the code, scripts and documentation prior to a final release.

The 6 month advance allows time for the CMAQ base model to be fully tested and vetted prior to beginning updates to the instrumented versions that are dependent on and incrementally added to the base model package.

- Versioning
- Current Development Release
  - CMAQ 5.2
  - Alpha
  - Beta release
  - Gamma release
- Current Stable Release
  - CMAQ 5.1
    - CMAQv5.1 Base Model
    - Two-Way WRF34-CMAQ5.02
- Maintenance release
  - CMAQ 5.0.2
  - CMAQv5.0.2 Base Model
  - Two-Way WRF34-CMAQ5.02
  - Integrated Source Apportionment Method (CMAQ-ISAM)
  - Direct Decoupled Method (CMAQ-DDM)
  - Sulfur Tracking Method (CMAQ-STM)
  - Volatility Basis Set (CMAQ-VBS)
  - Advanced Plume Treatment (CMAQ-APT)

CMAQ Code Introduction

General Code Introduction

Parallelization Strategy

Contributions Life-cycle

The repositories are all hosted on github. The typical life-cycle of a project is as follows:

1. Create a design document for the project.
2. Visit appropriate repository website.
3. Create a fork of the repository.
4. Locally clone the newly created fork.
5. Create a branch within the fork, for the new feature or bug x.
6. Develop branch.
7. Push complete branch to remote fork.
8. Submit a pull request to merge branch on fork to ??

Projects don't have to follow this example verbatim, but this at least gives a general overview of the process. Some details related to this life-cycle will be described in the following sections.

Code level requirements are described in chapter 5.

Development Strategy

As developers of CMAQ, we attempt to make the code look as uniform as we can across the entire code-base. In order to enforce this, there are a set of guidelines developers should follow.

Code Consistency

- Each science module has a name and abbreviation, and for each method within each science module there is a defined abbreviation. For example:
- The science module named horizontal advection has the abreviation hadv and the available method is abbreviated as yamo.
- The science module named vertical advection has the abreviation vadv, and the available methods are abbreviated as wrf and yamo.
- All subroutines should be named in a manner which prevents namespace conflicts.
- Subroutine names should all be upper case, with underscores in place of spaces
- Variable names should be upper case, with underscores in place of spaces).
- In general, variable names should be self-descriptive (e.g. NCELLS rather than N).

Benchmarking and Testing

- Developers need to test using multiple compilers, multiple processor configurations, and singel processor configuration runs for single day to verify answers match.
- Developers need to share results of tests and request review of the documentation prior to a merge.
- Developers of CMAQ need reviews from multiple CMAQ maintainers prior to a merge.

Two classes of tests:

- Compiler tests used the default benchmark configuration with different compilers and MPI configurations.
- Configuration tests used the Portland Group 15.7 OpenMPI compiler to generate executables that exercise different scientific configurations of the release software.

Compiler flags:

- PGI: -Mfixed -O3 -Mextend
- GCC: -ffixed-form -ffixed-line-length-132 -O3 -funroll-loops -finit-character=32
- Intel: -fixed -132 -O3 -override-limits -fno-alias -mp1 -fp-model precise -fp-model source -shared-intel -openmp
- In the Intel Basic Test: -fixed -132 -O3 -openmp
- In the NoOpt Tests: -O0 with extend source and fixed line length flags

Testing Manifest Table Example

Scenario	Compiler	netCDF	I/O API	MPI_YN?	VN? (#P)	CMAQ Timing (HH:MM:SS)	CMAQ New Project Timing (HH:MM:SS)
Gfortran Serial	Gfortran version 4.8.1	4.3.3	3.1 (Nov 2015)	N	N/A	8:19:51	7:35:30
Gfortran MVAPICH2	Gfortran version 4.8.1	4.3.2	3.1 (Nov 2015)	Y (16)	mvapich2-1.7	0:45:55	0:42:40

Verification of Model output

m3diff

- see min, max, mean differences between two different model runs

VERDI

- absolute difference plots for multiple variables, timesteps, layers (see spatial differences)

1:1 Scatter Plots

-

Documentation

- Subroutines and modules should be appropriately documented. CMAQ code use git tags to facilitate creation of Release Notes.
- Provide an example of how the EPA creates their Release Notes

Merging Code

- Commit code and documentation to your fork
- Submit pull request
- Development within a science module should follow the practices of that module, for documentation etc.

Copyright Information

Code Maintainers

Maintainers are developers with write access to the main developers repository. Code maintainers will be responsible for pull requests into CMAQ.

Current CMAQ github maintainers are as follows:

- Bill Hudzell, EPA Private CMAQ-Development Repository
- Ben Murphy, EPA Public CMAQ-Release Repository
- Zac Adelman, CMAS-Center Public CMAQ-Release Repository