1 Developers' Guide for the Community Multiscale Air Quality (CMAQ) Modeling System

Consistent with CMAQ model version 5.2.1 (2018 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

1.1 Disclaimer

The information in this Developer Guide 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. Mention of trade names or commercial products does not constitute endorsement or recommendation for use.

2 Motivation

The evolution and robustness of CMAQ depends on contributions from the vibrant CMAS community. The development team at EPA is excited to work with potential contributors and integrate community submissions into the CMAQ code base. In order to facilitate this process, we describe in this document our development process and how external developers may submit new code features.
The information contained here should be read prior to starting a project within the CMAQ framework. Instructions are tailored for external developers but can also be used by developers in the CMAS-Center or within EPA.
Summary of Developer Workflow The public CMAQ release repository is located on GitHub (https://github.com/USEPA/CMAQ). Users should refer to this repository for bug fixes, issues, documentation and major releases for CMAQ. Users can use the watch and star buttons on the public CMAQ release repository page to be notified of updates and changes. Developers interested in submitting code changes should read this Developer Guide and then contact the EPA CMAQ development team as soon as possible to discuss their motivation and plans for submitting a code change (CMAQ_Team@epa.gov).

In order to facilitate incorporation of a contribution, developers should follow the instructions on code requirements and repository layout as described in the CMAQ Operational Guidance Document, particularly Chapter 11. Documentation of the assumptions and results of the new code is a very important part of a meaningful code submission. If the submission involves

a detailed new feature, developers are encouraged to publish the use of their feature in a peer-reviewed journal before submission.

To begin, the developer should fork the public CMAQ release repository within GitHub. This will create a copy of the public CMAQ release repository under your name (https://github.com/{user_name}/CMAQ). Developers should use standard git commands to clone the appropriate version branch (5.2.1, 5.2, 5.2_DDM, ..) from your forked repository to your local machine and then to create a new feature or bug fix branch. Developers will add, commit and push changes to their new feature or bug fix branch on their forked repository, not to the public release version of the repository.

Once a feature or bug fix branch meets requirements for code consistency, benchmark testing, model output evaluation, and documentation including release notes, the developer may submit a pull request from their local feature or bug fix branch of their fork of the CMAQ repository on Github to the CMAQ public repository. This process is described in the Nuts and Bolts section below, and in the following tutorial, which also provides instructions on how to keep a fork up to date with changes on the public release repository.

Contributions will undergo a thorough code review within EPA before being incorporated in the next model release. Depending on the size, scope, and importance of the contribution, the CMAQ development team may or may not agree to support the update through future releases. Decisions regarding ongoing support will be made on a case-by-case basis with input from the developer who submits the contribution. The following sections outline the CMAQ code development and review process in greater detail.

3 Development Life-cycle

3.1 Public Release Versions

CMAQ uses a number versioning system for each release version branch, with major and minor increments. For example, in the case of hypothetical version 14.0 the first number (major version) and second number (minor version) refer to a stable release version. The minor version (second number) of CMAQ increments when one or many new science developments have been adopted. Although these changes may significantly affect model results, the model will still be generally compatible with inputs developed for versions of the same major number. The major version (first number) of CMAQ increments when significant development changes to the code base have been adopted such that backward compatibility or comparability is no longer expected. Modifications to the publically released version without increment are prohibited in order to ensure consistency among published literature referring to a particular model version. In between published releases the development team may publish solutions to model bugs and issues in the public repository under the folder DOCS/Known_Issues. The README located in this folder describes existing known issues, their scope and impact, and how they may be solved.

3.2 Development Versions

Prior to the public release of each major CMAQ version, the unofficial source code 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 give community members: - a reasonable amount of time to complete any pending feature submissions they would like to submit for the stable release. - a role in helping to test, troubleshoot, and debug the unofficial code before the stable release. - an opportunity to comment on the code improvements made in the new version. - the ability to take advantage of improvements for preliminary studies of their own interest. - a reasonable amount of time to ensure the new version is compatible with any features the member may have submitted in the past. The unofficial (or ***beta***) version of the code will first be vetted internally and then released generally 6 months in advance of the corresponding stable CMAQ release; this period is known as the *beta-phase*. At this time, EPA will announce the deadline for community contributions. This deadline will be chosen in order to balance both the time needed by developers to submit their contributions and the time needed by EPA to incorporate submissions before public release. Version numbering for the beta series will append the letter 'b' and an incrementing number to the expected version number of the stable release. The number of beta versions is variable among releases. For example, before the hypothetical release of CMAQv14.3 the following series of version numbers could be expected:

v14.3.b0 (First tested internal EPA version)
v14.3.b1 (Release to public after minor changes)
v14.3.b2 (. . . .incremental testing, ...)
v14.3.b3 (. . . .bug squashing, and . . .)
v14.3.b4 (. . . .documentation updates. . .)
v14.3 (Stable Public Release)

As stated previously, the "Known Issues" section of the documentation will be continually updated as problems are identified in the released code-base. These updates will not be implemented in the default model code and so the version numbering will in general, not increment between public releases. After public release of the stable CMAQ version, the instrumented versions of the code (e.g. DDM, ISAM, STM, etc) should be expected within approximately 6 months to 1 year. v14.3_DDM-3D (Instrumented version for DDM-3D)
v14.3_ISAM (Instrumented version for ISAM)
v14.3_Sulfur_Tracking (Instrumented version for Sulfur Tracking)

4 Making Contributions

4.1 Get in touch

Community members with an idea for a code contribution are encouraged to contact the EPA development team well before the *beta-phase* in order to plan appropriately for the testing and inclusion of the contribution. The EPA team may be interested in knowing information

including but not limited to the following: - What science module or bug do you intend to address? What work do you intend to contribute to CMAQ? - Are you comfortable with 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?

4.2 Nuts and Bolts

As described above, the CMAQ development process follows a "Forking Workflow." Atlassian has provided a helpful explanation. Developers should follow the guidance at GitHub Help and Atlassian in order to:
- fork the CMAQ repo: https://help.github.com/articles/fork-a-repo/#platform-linux
- clone their newly-created fork: https://help.github.com/articles/cloning-a-repository/#platform-linux
- create a feature branch: https://www.atlassian.com/git/tutorials/using-branches
- add and commit changes to the new feature branch: https://www.atlassian.com/git/tutorials/saving-changes
- push the feature branch to the forked repo: https://help.github.com/articles/pushing-to-a-remote/
- submit a pull request to the public CMAQ repo: https://help.github.com/articles/creating-a-pull-request-from-a-fork/

Developers should run and test their contribution before submitting the pull request so that the results of the test can be included in the documentation of the pull request.

4.3 Code Review

CMAQ Developers at EPA will review all code submissions in order to ensure code stability and consistency, and prevent degradation of model performance. After review, the EPA team will either accept the submission, recommend specific improvements to the submission, or in some cases reject the submission. To avoid outright rejection, we urge developers to contact the EPA team early in the development process and maintain contact throughout to help ensure the submission is compatible with the CMAQ code base and is a robust addition.

4.3.1 Code Consistency

Please refer to the Operational Guidance Document, Chapter 11, under the section titled "Guidelines for Developing New CMAQ Source Code". Examples of small, but important guidelines include: - Eliminate global memory references (across modules). In other words, no common blocks across modules, no hidden data paths, and no "back doors." - All subroutines should be named in a manner which prevents namespace conflicts. - In general, variable names should be self-descriptive (e.g. NCELLS rather than N). - Use the Fortran declaration IMPLICIT NONE to maintain some control on typographic errors and undefined variables.

The use of IMPLICIT NONE forces the developer to declare all internal variables. This is standard in Fortran 90. - In general, it is expected that MKS units are used for input and output variables, as these units have been standardized throughout the CMAQ system. If you use alternative units, please document this exhaustively.

4.3.2 Benchmark Testing

Dataset: The U.S. EPA Southeast US 12km domain July 1-14, 2011 testing dataset is provided with the CMAQv5.2 Release. This dataset is distributed for benchmarking and testing the model installation. It is available from CMAS; please go to https://www.epa.gov/cmaq/cmaq-inputs-and-test-case-data for instructions on how to download the test dataset.

Before making code changes, developers should test multiple compilers (if they have access to them; see the following section on **Compiler Tests**), multiple processor configurations, and single processor configuration runs for a single simulation day to verify their results match the previous stable release, and/or that their results are computationally and physically reasonable. After implementing their code changes, developers should repeat these tests and share the results as part of the pull request documentation.

4.3.2.1 Compiler Tests

Compiler tests use the default benchmark configuration with different compilers and MPI configurations. It is important for the user community that CMAQ always compile with Intel Fortran, Gnu Fortran and Portland Group Fortran compilers. If a developer has access to more than one compiler, it is critical that they test all of them. Some errors will cause different behaviors depending on the choice of compiler and may not be detectable with all of the compilers. See appendix 1 for an example of a Compiler Test.

4.3.2.2 Model Performance Tests

Configuration tests use one compiler to test the impact of a model change on results. See appendix 2 for an example of important information to collect when testing science options. The developer should consider submitting similar information with their pull request.

Several tools exist to document the effects of compiler choice and code change on model results. Examples include: **m3diff -** Quantify min, max, mean differences between two different model runs **VERDI -** Create absolute difference plots for multiple variables, timesteps, layers (see spatial differences)
In addition, we recommend utilizing **1:1 Scatter Plots** to demonstrate the differences between two model runs in a concise layout.

4.3.3 Documentation Requirements

Documentation is of course an integral part of the integration of any contribution into the CMAQ code base. The following documentation products are helpful for expediting the review and integration process: - A Release Note written by the developer which describes the motivation, algorithm and impacts of the contribution is required to ensure proper documentation of CMAQ.
- If the contribution is a new feature, developers are encouraged to publish its use in a peer-reviewed journal before submitting it to the CMAQ Public Repository.

CMAQ Documentation Resources:
Documentation for CMAQv5.2.1 is available at https://github.com/USEPA/CMAQ/tree/5.2.1/DOCS. Materials include: - an Operational Guidance Document which describes code structure and regular operation of the model. - several Release Notes describing code improvements relevant for this model release. - several Tutorials that give specific instructions for common tasks like running CMAQ or adding chemical tracers.

4.4 Ongoing Support

Depending on the size, scope, and importance of the contribution, the CMAQ development team may or may not have the resources to support it through future releases. For example, bug fixes and minor, but helpful, changes to the existing code will likely be incorporated into the general code base and supported. Large code additions, like a new process module or an instrumented version of CMAQ may require more effort to support than can be provided by resources of the EPA Office of Research and Development. However, if the feature is particularly of interest for the CMAQ user community, it may be supported. Decisions regarding ongoing support will be made on a case-by-case basis.

5 Copyright Information

Contact EPA (CMAQ_Team@epa.gov) with questions and concerns.

6 Appendix

6.1 Appendix 1: Compiler Tests

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

6.1.1 Compilation Testing Manifest Table (Example)

Scenario	Compiler	netCDF	I/O API	MPI YN (#P)	MPI	CMAQv5.1 Timing (hh:mm:ss)	CMAQv5.2 Timing (hh:mm:ss)	Notes
Gfortran Serial	Gfort version 4.8.1	4.3.3	3.1	N	N/A	8:19:51	7:35:30	UNC module gcc/4.8.1
Gfortran mvapich	Gfort version 4.8.1	4.3.2	3.1	Y (16)	mvapich2 1.7	0:45:55	0:42:40	
Intel Serial	Intel For-tran version 16.2.0	4.3.2	3.1	N	N/A	6:01:42	5:10:16	UNC module intel/16.2
Intel Open-MPI (EPA Config)	Intel For-tran v15.0.0	4.3.2	3.1	Y (16)	openMPI 1.42	0:34:27		UNC module openmpi_intel/15.0
Intel OpenMPI	Intel For-tran v16.2.0	4.3.2	3.1	Y (16)	openMPI 1.4.2	0:35:29		UNC module openmpi_intel/16.2
Intel mvapich2	Intel For-tran v16.2.0	4.3.2	3.1	Y (16)	mvapich2 1.7	0:36:34		UNC module mvapich2_intel/16.2
Portland Serial	PGI For-tran v16.1	4.3.2	3.1	N	N/A	7:33:36	6:26:31	UNC module pgi/16.1
Portland OpenMPI	PGI For-tran v15.7	4.3.2	3.1	Y (16)	openMPI 1.4.2	0:40:20	0:36:16	UNC module openmpi_pgi/15.7

6.2 Appendix 2: Model Performance Test Metadata

Scenario	Description	Mechanism	Notes	Timing (16PE) hh:mm:ss
Benchmark Case	Online emissions processing, inline photolysis, inline lightning from MCIP RC, no windblown dust, surface HONO, bidirectional NH3 and Hg, no potential vorticity scaling	cb05e51_ae6_aq	Done; LTNGNO InLine, LTNGPARM = N, LOG_START = 2.0	0:40:20
MOSAIC	Benchmark case with MOSAIC and additional stomatal flux files activated	cb05e51_ae6_aq	Done. set CTM_MOSAIC = Y; set CTM_FST = Y	0:44:02
Dust	Benchmark case with dust, including new MODIS FP input	cb05e51_ae6_aq	Done. setenv CTM_WB_DUST Y; setenv CTM_ERODE_AGLAND Y; setenv CTM_WBDUST_BELD BELD3	0:38:28
Hourly NLDN	Benchmark with lightning NOx calculated using hourly bNLDN strikes	cb05e51_ae6_aq	Done; LTNGNO InLine, LTNGPARM = Y, USE_NLDN Y	0:40:18

Scenario	Description	Mechanism	Notes	Timing (16PE) hh:mm:ss
POA Sensitivity	Benchmark with new POA mechanism	cb05e51_ae6nvPOA_ab	Done	0:34:42