
GEOS-Chem Classic

Release 14.3.0

GEOS-Chem Support Team

Apr 01, 2024

GETTING STARTED

1	Quickstart Guide	3
1.1	1. Clone GEOS-Chem Classic	3
1.2	2. Create a run directory	4
1.3	3. Load your Environment	4
1.4	4. Configure your build	5
1.5	5. Compile and install	5
1.6	6. Configure your run directory	6
1.7	7. Run GEOS-Chem Classic	6
2	Meet all hardware requirements	9
2.1	Computer system	9
2.2	Memory requirements	10
2.3	Disk space requirements	10
3	Meet all software requirements	13
3.1	Supported compilers	13
3.2	Required software packages	14
3.3	Optional but recommended software packages	15
4	Customize your login environment	17
4.1	Environment files	17
4.2	Sample environment file for GNU 10.2.0 compilers	17
4.3	Sample environment file for Intel 2023 compilers	19
4.4	Set environment variables for compilers	20
4.5	Set environment variables for parallelization	21
5	Key references	23
6	Download source code	25
6.1	Source code repositories	25
6.2	Download instructions	26
7	Create a run directory	31
7.1	First-time user registration	31
7.2	Example: Create a full-chemistry simulation run directory	32
7.3	Example: Create a CH ₄ simulation run directory	36
7.4	Run directory files and folders	39
8	Compile the source code	43
8.1	Configure with CMake	43
8.2	Compile with Make	50

8.3	Get a summary of compilation options	53
9	Configure your simulation	55
9.1	Commonly-updated configuration files	55
9.2	Less-commonly updated configuration files	85
10	Download input data	89
10.1	Input data for GEOS-Chem Classic	89
10.2	Restart files	91
10.3	Download data with a dry-run simulation	96
11	Run your simulation	101
11.1	Create a run script	101
11.2	Complete this pre-run checklist	103
11.3	Submit your run script to a scheduler	103
11.4	Or run the script from the command line	104
11.5	Verify a successful simulation	105
11.6	Minimize differences in multi-stage runs	105
11.7	Speed up a slow simulation	106
12	View output files	107
12.1	Log files	107
12.2	History diagnostics output	108
12.3	Other diagnostic output files	110
13	Diagnostics reference	111
13.1	GEOS-Chem History diagnostics	111
13.2	HEMCO diagnostics	111
13.3	Planeflight diagnostic	111
13.4	ObsPack diagnostic	115
14	GEOS-Chem Classic folder tree	121
14.1	GEOS-Chem folder tree	121
14.2	HEMCO folder tree	122
15	Sample GEOS-Chem run scripts	123
15.1	For clusters using the Slurm scheduler	123
15.2	For Amazon Web Services EC2 cloud instances	124
16	Load software into your environment	127
16.1	On the Amazon Web Services Cloud	127
16.2	On a shared computer cluster	127
17	Build required software with Spack	131
17.1	Introduction	131
17.2	Install Spack and do first-time setup	132
17.3	Clone a copy of GCCClassic, GCHP, or HEMCO	133
17.4	Install the recommended compiler	133
17.5	Build GEOS-Chem dependencies and useful tools	134
17.6	Add spack load commands to your environment file	135
17.7	Clean up	137
18	Customize simulations with research options	139
18.1	Aerosols	139
18.2	Chemistry	140
18.3	Diagnostics	142

18.4	Emissions	143
18.5	Photolysis	145
18.6	Wet deposition	145
19	Understand what error messages mean	147
19.1	Where does error output get printed?	147
19.2	Compile-time errors	148
19.3	Run-time errors	149
19.4	File I/O errors	155
19.5	Segmentation faults and similar errors	156
19.6	Less common errors	158
20	Debug GEOS-Chem and HEMCO errors	161
20.1	Check if a solution has been posted to Github	161
20.2	Check if your computational environment is configured properly	161
20.3	Check any code modifications that you have added	161
20.4	Check if your runs exceeded time or memory limits	162
20.5	Send debug printout to the log files	162
20.6	Look at the traceback output	163
20.7	Identify whether the error happens consistently	163
20.8	Isolate the error to a particular operation	163
20.9	Compile with debugging options	164
20.10	Use a debugger	164
20.11	Print it out if you are in doubt!	165
20.12	Use the brute-force method when all else fails	165
20.13	Identify poorly-performing code with a profiler	165
21	Manage a data archive with bashdatacatalog	167
21.1	What is bashdatacatalog?	167
21.2	Usage instructions	167
22	Work with netCDF files	169
22.1	Useful tools	169
22.2	Examine the contents of a netCDF file	170
22.3	Read the contents of a netCDF file	173
22.4	Determining if a netCDF file is COARDS-compliant	174
22.5	Edit variables and attributes	175
22.6	Concatenate netCDF files	178
22.7	Regrid netCDF files	179
22.8	Crop netCDF files	180
22.9	Add a new variable to a netCDF file	180
22.10	Chunk and deflate a netCDF file to improve I/O	182
23	Prepare COARDS-compliant netCDF files	185
23.1	The COARDS netCDF standard	185
23.2	COARDS dimensions	185
23.3	COARDS coordinate vectors	186
23.4	COARDS data arrays	190
23.5	COARDS Global attributes	192
23.6	For more information	193
24	View GEOS-Chem species properties	195
24.1	Species properties defined	195
24.2	Access species properties in GEOS-Chem	204

25	Parallelize GEOS-Chem and HEMCO source code	209
25.1	Overview of OpenMP parallelization	209
25.2	Example using OpenMP directives	210
25.3	Environment variable settings for OpenMP	211
25.4	OpenMP parallelization FAQ	211
25.5	MPI parallelization	215
26	Run nested-grid simulations	217
26.1	Run a global simulation to create boundary conditions	217
26.2	Set up your nested grid run directory	219
26.3	Frequently asked questions	220
27	Update chemical mechanisms with KPP	223
27.1	Using KPP: Quick start	223
27.2	Using KPP: Reference section	226
28	View related documentation	235
29	GEOS-Chem version history	237
30	Known bugs and issues	239
30.1	Current bug reports	239
31	Contributing Guidelines	241
31.1	We use GitHub and ReadTheDocs	241
31.2	When should I submit updates?	241
31.3	How can I submit updates?	241
31.4	How can I request a new feature?	243
31.5	How can I report a bug?	243
31.6	Where can I ask for help?	243
32	Support Guidelines	245
32.1	How to report a bug	245
32.2	Where can I ask for help?	245
32.3	What type of support can I expect?	245
32.4	How to submit changes	246
32.5	How to request an enhancement	246
33	Editing this User Guide	247
33.1	Quick start	247
33.2	Learning reST	247
33.3	Style guidelines	248
	Bibliography	249
	Index	251

This site provides instructions for **GEOS-Chem Classic**, the single-node mode of operation of the [GEOS-Chem model](#). We provide instruction for downloading and compiling GEOS-Chem Classic, plus its required software libraries.

Note: If you would like to run GEOS-Chem on more than one node of a computing system, consider using [GEOS-Chem High Performance \(GCHP\)](#).

GEOS-Chem is a global 3-D model of atmospheric composition driven by assimilated meteorological observations from the Goddard Earth Observing System (GEOS) of the [NASA Global Modeling and Assimilation Office](#). It is applied by [research groups around the world](#) to a wide range of atmospheric composition problems.

- [GEOS-Chem Overview](#)
- [Narrative description of GEOS-Chem](#)

Cloning and building from source code ensures you will have direct access to the latest available versions of GEOS-Chem Classic, provides additional compile-time options, and allows you to make your own modifications to GEOS-Chem Classic source code.

QUICKSTART GUIDE

This quickstart guide is for quick reference on how to download, build, and run **GEOS-Chem Classic**, which is the single-node instance of GEOS-Chem.

Tip: Please also see our [GCHP Quickstart Guide](#) if you would like to run GEOS-Chem across using more than one computational node.

This guide assumes that your environment satisfies GEOS-Chem Classic *hardware* and *software* requirements. This means you should load a *compute environment* such that programs like **cmake** are available before continuing. If you do not have some of the required software dependencies, you can find instructions for installing external dependencies in our Spack instructions.

For simplicity we will also refer to **GEOS-Chem Classic** as simply **GEOS-Chem** on this page. More detailed instructions on downloading, compiling, and running GEOS-Chem can be found in the User Guide elsewhere on this site.

1.1 1. Clone GEOS-Chem Classic

Download the source code:

```
$ git clone --recurse-submodules https://github.com/geoschem/GCClassic.git GCClassic
$ cd GCClassic
```

Tip: If you wish, you may choose a different name for the source code folder, e.g.

```
$ git clone --recurse-submodules https://github.com/geoschem/GCClassic.git my_code_dir
$ cd my_code_dir
```

Upon download you will have the most recently released version. You can check what this is by printing the last commit in the git log and scanning the output for tag.

```
$ git log -n 1
```

Tip: To use an older GEOS-Chem Classic version (e.g. 14.0.0), follow these additional steps:

```
$ git checkout tags/14.0.0          # Points HEAD to the tag "14.0.0"
$ git branch version_14.0.0        # Creates a new branch at tag "14.0.0"
```

(continues on next page)

(continued from previous page)

```
$ git checkout version_14.0.0          # Checks out the version_14.0.0 branch
$ git submodule update --init --recursive # Reverts submodules to the "14.0.0" tag
```

You can do this for any tag in the version history. For a list of all tags, type:

```
$ git tag
```

If you have any unsaved changes, make sure you commit those to a branch prior to updating versions.

1.2 2. Create a run directory

Navigate to the `run/` subdirectory. To *create a run directory*, run the script `./createRunDir.sh`:

```
$ cd run/
$ ./createRunDir.sh
```

Creating a run directory is interactive, meaning you will be asked multiple questions to set up the simulation. For example, running `createRunDir.sh` will prompt questions about configurable settings such as simulation type, grid resolution, meteorology source, and number of vertical levels. It will also ask you where you want to store your run directory and what you wish to name it, including whether you want to use the default name, e.g. `gc_4x5_merra2_fullchem`. We recommend storing run directories in a place that has a large storage capacity. It does not need to be in the same location as your source code. When creating a run directory you can quit and start from scratch at any time.

For demonstration purposes, we will use a full chemistry simulation run directory with the default name (`gc_merra2_4x5_fullchem`). The steps to setup and run other types of GEOS-Chem simulations follow the same pattern as the examples shown below.

Attention: The first time you create a run directory, you will be asked to provide *registration information*. Please answer all of the questions, as it will help us to keep track of GEOS-Chem usage worldwide. We will also add your information to the [GEOS-Chem People and Projects web page](#).

1.3 3. Load your Environment

Prior to building GEOS-Chem always make sure all libraries and environment variables are loaded. An easy way to do this is to write an *environment file* and load that file every time you work with GEOS-Chem. To make this extra easy you can create a symbolic link to your environment file within your run directory or for reference. For example, do the following in your new run directory to have a handy link to the environment you plan on using.

```
$ cd /path/to/gc_4x5_merra2_fullchem # Skip if you are already here
$ ln -s ~/envs/gcc.gfortran10.env gcc.env
```

Then every time you start up a session to work with GEOS-Chem in your run directory you can easily load your environment.

```
$ source gcc.env
```

1.4 4. Configure your build

You may build GEOS-Chem from within the run directory or from anywhere else on your system. But we recommend that you always build GEOS-Chem from within the run directory. This is convenient because it keeps all build files in close proximity to where you will run the model. For this purpose the GEOS-Chem run directory includes a build directory called `build/`.

First, navigate to the `build/` folder of your run directory:

```
$ cd /path/to/gc_4x5_merra2_fullchem # Skip if you are already here
$ cd build
```

The next step is to *configure your build*. These are persistent settings that are saved to your build directory. A useful configuration option is `-DRUNDIR`. This option lets you specify one or more run directories that GEOS-Chem is “installed” to; that is, where the executable is copied, when you do **make install**.

Configure your build so it installs GEOS-Chem to the run directory you created in Step 2. The run directory is one directory level higher than the `build` directory. Also located one level higher than the `build` directory is the `CodeDir` symbolic link to the top-level GEOS-Chem source code directory. Use the following command to configure your build:

```
$ cmake ../CodeDir -DRUNDIR=..
```

GEOS-Chem has a number of *additional configuration options* you can add here. For example, to compile with `RRTMG` after running the above command:

Note: The `.` in the **cmake** command above is important. It tells CMake that your current working directory (i.e., `.`) is your build directory.

```
$ cmake . -DRRTMG=y
```

A useful configuration option is to build in **debug mode**. Doing this is a good idea if you encountered an error (such as a segmentation fault) in a previous run and need more information about where the error happened and why.

```
$ cmake . -DCMAKE_BUILD_TYPE=Debug
```

See the *GEOS-Chem documentation* for more information on configuration options.

1.5 5. Compile and install

Compiling GEOS-Chem Classic should take about a minute, but it can vary depending on your system, your compiler, and your configuration options. To maximize build speed you should compile GEOS-Chem in parallel using as many cores as are available. Do this with the `-j` flag from the `build/` directory:

```
# cd /path/to/gc_4x5_merra2_fullchem/build # Skip if you are already here
$ make -j
```

Upon successful compilation, install the compiled executable to your run directory:

```
$ make install
```

This copies executable `build/bin/gcclassic` and supplemental files to your run directory.

Note: You can update build settings at any time:

1. Navigate to your build directory.
 2. Update your build settings with **cmake** (only if they differ since your last execution of cmake)
 3. Recompile with **make -j**. Note that the build system automatically figures out what (if any) files need to be recompiled.
 4. Install the rebuilt executable with **make install**.
-

If you do not install the executable to your run directory you can always get the executable from the directory **build/bin**.

1.6 6. Configure your run directory

Now, navigate to your run directory:

```
$ cd /path/to/gcc_4x5_merra2_fullchem
```

You should review these files before starting a simulation:

- *geoschem_config.yml*
 - Controls several frequently-updated simulation settings (e.g. start and end time, which operations to turn on/off, etc.)
- *HISTORY.rc*
 - Controls GEOS-Chem diagnostic settings.
- *HEMCO_Diagn.rc*
 - Controls emissions diagnostic settings via *HEMCO*.
- *HEMCO_Config.rc*
 - Controls which emissions inventories and other non-emissions data will be read from disk (via *HEMCO*).

Please see our *Customize simulations with research options* Supplemental Guide to learn how you can customize your simulation by activating alternate science options in your simulations.

Once you are satisfied that your simulation settings are correct, you may proceed to run GEOS-Chem.

1.7 7. Run GEOS-Chem Classic

If you used an *environment file* to load software libraries prior to building GEOS-Chem then you should load that file prior to running. To *run GEOS-Chem Classic*, type at the command line:

```
$ ./gcclassic
```

If you wish to send output to a log file, use:

```
$ ./gcclassic > GC.log 2>&1
```

We recommend *running GEOS-Chem Classic as a batch job*, although you can also do short runs interactively. Running GEOS-Chem as a batch job means that you *write a script (usually bash)* and then you submit that script to your local job scheduler (SLURM, LSF, etc.). If you write a batch script you can include sourcing your *environment file*

within the script to ensure you always use the intended environment. Submitting GEOS-Chem as a batch job is slightly different depending on your scheduler. If you aren't familiar with scheduling jobs on your system, ask your system administrator for guidance.

Those are the basics of using GEOS-Chem Classic! See this user guide, step-by-step guides, and reference pages for more detailed instructions.

MEET ALL HARDWARE REQUIREMENTS

If you are a first-time GEOS-Chem Classic user, please take a moment to make sure that your computer system meets certain hardware and software requirements. These are described in the following chapters.

2.1 Computer system

You will need to have access to one (or both) of these types of computational resources in order to use GEOS-Chem Classic:

2.1.1 A Unix-like computer system

GEOS-Chem Classic can only be used on computers with operating systems that are **Unix-like**. This includes all flavors of Linux (e.g. Ubuntu, Fedora, Red-Hat, Rocky Linux, Alma Linux, etc) and BSD Unix (including MacOS X, which is a BSD derivative).

If your institution has computational resources (e.g. a shared computer cluster with many cores, sufficient *disk storage* and *memory*), then you can run GEOS-Chem Classic there. Contact your sysadmin or IT support staff for assistance.

2.1.2 An account on the Amazon Web Services cloud

If your institution lacks computational resources (or if you need additional computational resources), then you should consider signing up for access to the Amazon Web Services cloud. Using the cloud has the following advantages:

- You can run GEOS-Chem without having to invest in local hardware and maintenance personnel.
- You won't have to download any meteorological fields or emissions data. All of the necessary data input for GEOS-Chem will be available on the cloud.
- You can initialize your computational environment with all of the required software (e.g. compilers, libraries, utilities) that you need for GEOS-Chem.
- Your GEOS-Chem runs will be 100% reproducible, because you will initialize your computational environment the same way every time.
- You will avoid GEOS-Chem compilation errors due to library incompatibilities.
- You will be charged for the computational time that you use, and if you download data off the cloud.

You can learn more about how to use GEOS-Chem on the cloud by [visiting this tutorial \(cloud.geos-chem.org\)](https://cloud.geos-chem.org).

2.2 Memory requirements

If you plan to run GEOS-Chem on a local computer system, please make sure that your system has sufficient memory to run your simulations.

2.2.1 Sufficient memory to run GEOS-Chem

For the $4^{\circ} \times 5^{\circ}$ “standard” simulation

- 8-15 GB RAM

For the $2^{\circ} \times 2.5^{\circ}$ “standard” simulation:

- 30-40 GB RAM
- 20 GB memory (MaxRSS)
- 26 GB virtual memory (MaxVMSize)

Our standard GEOS-Chem Classic 1-month full-chemistry benchmark simulations use a little under 14 GB of system memory. This is mostly due to the fact that the benchmark simulations archive the “kitchen sink”—that is, most diagnostic outputs are requested so that the benchmark simulation can be properly evaluated. But a typical GEOS-Chem Classic production simulation would not require all of these diagnostic outputs, and thus would use much less memory than the benchmark simulations.

2.2.2 Extra memory for special simulations

You may want to consider at least 30 GB RAM if you plan on doing any of the following:

- Running high-resolution (e.g. $1^{\circ} \times 1.25^{\circ}$ or higher resolution) global simulations
- Running high-resolution (e.g. $0.25^{\circ} \times 0.3125^{\circ}$ or $0.5^{\circ} \times 0.625^{\circ}$)
- Running $2^{\circ} \times 2.5^{\circ}$ and generating a lot of diagnostic output. The more diagnostics you turn on, the more memory GEOS-Chem Classic will require).

2.3 Disk space requirements

The following sections will help you assess how much disk space you will need on your server to store GEOS-Chem Classic *input data* and *output data*.

2.3.1 Space for GEOS-Chem Classic input data

The data format used by GEOS-Chem Classic is *COARDS-compliant netCDF*. This is a standard file format used for Earth Science applications. See our *netCDF guide* for more information.

Emissions input fields

Please see our [Emissions input data](#) section for more information.

Meteorology fields

The amount of disk space that you will need depends on two things:

1. Which type of met data you will use, and
2. How many years of met data you will download

Table 1: Disk space needed for 1-year of MERRA-2 data

Resolution	Type	Size GB/yr
$1^{\circ} \times 1.25^{\circ}$	Global	~30
$2^{\circ} \times 2.5^{\circ}$	Global	~110
$0.5^{\circ} \times 0.625^{\circ}$	Nested Asia (aka AS)	~115
$0.5^{\circ} \times 0.625^{\circ}$	Nested Europe (aka EU)	~58
$0.5^{\circ} \times 0.625^{\circ}$	Nested North America (aka NA)	~110

Table 2: Disk space needed for 1-year of GEOS-FP data

Resolution	Type	Size GB/yr
$1^{\circ} \times 1.25^{\circ}$	Global	~30
$2^{\circ} \times 2.5^{\circ}$	Global	~120
$0.25^{\circ} \times 0.3125^{\circ}$	Nested Asia (aka AS)	~175
$0.25^{\circ} \times 0.3125^{\circ}$	Nested Europe (aka EU)	~175
$0.25^{\circ} \times 0.3125^{\circ}$	Nested North America (aka NA)	~175

GCAP 2.0: to be added

Obtaining emissions data and met fields

There are several ways to obtain the input data required for GEOS-Chem classic. These are described in more detail in the following sections.

1. Perform a *GEOS-Chem dry-run simulation*;
2. Download and manage data with the *bashdatacatalog tool*;
3. Transfer data with [Globus GEOS-Chem data \(WashU\) endpoint](#).

Also see our [Input data for GEOS-Chem Classic](#) for more data download options.

2.3.2 Space for data generated by GEOS-Chem Classic

Monthly-mean output

We can look to the **GEOS-Chem Classic** full-chemistry benchmark simulations for a rough upper limit of how much disk space is needed for diagnostic output. The [GEOS-Chem 13.0.0 vs. 12.9.0 1-month benchmark simulation](#) generated approximately 837 MB/month of output. Of this amount, diagnostic output files accounted for ~646 MB and restart files accounted for ~191 MB.

We say that this is an upper limit, because benchmark simulations archive the “kitchen sink”—all species concentrations, various aerosol diagnostics, convective fluxes, dry dep fluxes and velocities, J-values, various chemical and meteorological quantities, transport fluxes, wet deposition diagnostics, and emissions diagnostics. Most GEOS-Chem users would probably not need to archive this much output.

GEOS-Chem Classic specialty simulations—simulations for species with first-order loss by prescribed oxidant fields (i.e. Hg, CH₄, CO₂, CO)—will produce much less output than the benchmark simulations. This is because these simulations typically only have a few species.

Reducing output file sizes

You may subset the horizontal and vertical size of the diagnostic output files in order to save space. For more information, please see our section on [GEOS-Chem History diagnostics](#).

Furthermore, since GEOS-Chem 13.0.0, we have modified the diagnostic code so that diagnostic arrays are only dimensioned with enough elements necessary to save out the required output. For example, if you only wish to output the SpeciesConc_O3 diagnostic, GEOS-Chem will dimension the relevant array with (NX,NY,NZ,1) elements (1 because we are only archiving 1 species). This can drastically reduce the amount of memory that your simulation will require.

Timeseries output

Archiving hourly or daily timeseries output would require much more disk space than the monthly-mean output. The disk space actually used will depend on how many quantities are archived and what the archival frequency is.

MEET ALL SOFTWARE REQUIREMENTS

If you are a first-time GEOS-Chem Classic user, please take a moment to make sure that your computer system meets certain hardware and software requirements. These are described in the following chapters.

3.1 Supported compilers

GEOS-Chem is written in the Fortran programming language. However, you will also need C and C++ compilers to install certain libraries (like *netCDF*) on your system.

3.1.1 Intel

The **Intel Compiler Suite** is our recommended proprietary compiler suite.

Intel compilers produce well-optimized code that runs extremely efficiency on machines with Intel CPUs. Many universities and institutions will have an Intel site license that allows you to use these compilers.

The GCST has tested **GEOS-Chem Classic** with these versions (but others may work as well):

- 23.0.0
- 19.0.5.281
- 19.0.4
- 18.0.5
- 17.0.4
- 15.0.0
- 13.0.079
- 11.1.069

Best way to install: [Direct from Intel](#) (may require purchase of a site license or a student license)

Tip: Intel 2021 may be obtained for free, or installed with a package manager such as [Spack](#).

3.1.2 GNU

The **GNU Compiler Collection** (or **GCC** for short) is our recommended open-source compiler suite.

Because the GNU Compiler Collection is free and open source, this is a good choice if your institution lacks an Intel site license, or if you are running GEOS-Chem on the Amazon EC2 cloud environment.

The GCST has tested **GEOS-Chem Classic** with these versions (but others may work as well):

- 12.2.0
- 11.2.0
- 11.1.0
- 10.2.0
- 9.3.0
- 9.2.0
- 8.2.0
- 7.4.0
- 7.3.0
- 7.1.0
- 6.2.0

Best way to install: *With Spack.*

3.2 Required software packages

3.2.1 Git

Git is the de-facto software industry standard package for source code management. A version of Git usually ships with most Linux OS builds.

The GEOS-Chem source code can be downloaded using the Git source code management system. GEOS-Chem software repositories are stored at the <https://github.com/geoschem> organization page.

Best way to install: git-scm.com/downloads. But first check if you have a version of Git pre-installed.

3.2.2 CMake

CMake is software that directs how the GEOS-Chem source code is compiled into an executable. You will need **CMake** version 3.13 or later to build GEOS-Chem Classic.

Best way to install: *With Spack.*

3.2.3 GNU Make

GNU Make is software that can build executables from Makefiles that are created by *CMake*.

While GNU Make is not required for GEOS-Chem 13.0.0 and later, some external libraries that you might need to build will require GNU Make. Therefore it is best to download GNU Make along with CMake.

Best way to install: *With Spack.*

3.2.4 netCDF

GEOS-Chem input and output data files use the netCDF file format (cf. *netCDF*). NetCDF is a self-describing file format that allows metadata (descriptive text) to be stored alongside data values.

Best way to install: *With Spack.*

3.3 Optional but recommended software packages

3.3.1 GCPy

GCPy is our recommended python companion software to GEOS-Chem.

While **GCPy** is not a general-purpose plotting package, it does contain many useful functions for creating zonal mean and horizontal plots from GEOS-Chem output. It also contains scripts to generate plots and tables from GEOS-Chem benchmark simulations.

Best way to install: With Mamba or Conda (see gcpy.readthedocs.io)

3.3.2 gdb and cgdb

The GNU debugger (*gdb*) and its graphical interface (*cgdb*) are very useful tools for tracking down the source of GEOS-Chem errors, such as segmentation faults, out-of-bounds errors, etc.

Best way to install: *With Spack.*

3.3.3 ncview

The *ncview* program is a netCDF file viewer. While it does not produce publication-quality output, ncview can let you easily examine the contents of a netCDF data file (such as those which are input and output by GEOS-Chem). Ncview is very useful for debugging and development.

3.3.4 nco

The netCDF operators (*nco*) are powerful command-line tools for editing and manipulating data in netCDF format.

Best way to install: *With Spack.*

3.3.5 cdo

The Climate Data Operators (*cdo*) are powerful command-line utilities for editing and manipulating data in netCDF format.

Best way to install: *With Spack.*

3.3.6 KPP

The Kinetic PreProcessor (*KPP*) translates a chemical mechanism specification from user-configurable input files to Fortran-90 source code. You will need to use **KPP** if you plan on updating any of the chemical mechanisms that ship with GEOS-Chem.

Best way to install: Clone from github.com/KineticPreProcessor/KPP and build the the *KPP* executable from source.

3.3.7 flex and bison

Flex is the Fast Lexical Analyzer, and *bison* is a general purpose parser-generator. *KPP* uses both **flex** and **bison** to parse chemical mechanism definition files. Depending on your setup, these packages might have already been installed for you.

Best way to install: *With Spack.*

CUSTOMIZE YOUR LOGIN ENVIRONMENT

Tip: You may *skip ahead* if you will be using **GEOS-Chem Classic** on an Amazon EC2 cloud instance. When you initialize the EC2 instance with one of the pre-configured Amazon Machine Images (AMIs) all of the required software libraries will be automatically loaded.

Each time you log in to your computer system, you'll need to load the *software libraries* needed by GEOS-Chem into your environment. You can do this with a script known as an **environment file**, as described in the following chapters:

4.1 Environment files

An environment file is a script that:

1. Loads software libraries into your login environment. This is often done with a module manager such as **lmod**, **spack**, or **environment-modules**.
2. Stores settings for GEOS-Chem and its dependent libraries in shell variables called **environment variables**.

You will **source** the environment file each time you log in with a command such as:

```
$ . ~/my-environment-file    # or whatever you name it
```

Tip: Keep a separate environment file for each combination of modules that you will use. Example environment files for *GNU* and *Intel* compilers and related software are provided in the following sections.

For general information about how libraries are loaded, see *Load software into your environment*.

4.2 Sample environment file for GNU 10.2.0 compilers

Below is a sample environment file (based on an environment file for the Harvard Cannon computer cluster). This file will load software libraries built with the GNU 10.2.0 compilers.

Save the code below (with any appropriate modifications for your own computer system) to a file named `~/gcclassic.gnu10.env`.

```
#####  
# Load software packages (EDIT AS NEEDED)  
#####
```

(continues on next page)

(continued from previous page)

```

# Unload all modules first
module purge

# Load modules
module load gcc/10.2.0-fasrc01          # gcc / g++ / gfortran
module load openmpi/4.1.0-fasrc01      # MPI
module load netcdf-c/4.8.0-fasrc01     # netcdf-c
module load netcdf-fortran/4.5.3-fasrc01 # netcdf-fortran
module load flex/2.6.4-fasrc01         # Flex lexer (needed for KPP)
module load cmake/3.25.2-fasrc01       # CMake (needed to compile)

#####
# Environment variables and related settings
# (NOTE: Lmod will define <module>_HOME variables for each loaded module)
#####

# Make all files world-readable by default
umask 022

# Set number of threads for OpenMP. If running in a SLURM environment,
# use the number of requested cores. Otherwise use 8 cores for OpenMP.
if [[ "x${SLURM_CPUS_PER_TASK}" == "x" ]]; then
    export OMP_NUM_THREADS=8
else
    export OMP_NUM_THREADS="${SLURM_CPUS_PER_TASK}"
fi

# Max out the stacksize memory limit
export OMP_STACKSIZE="500m"

# Compilers
export CC="gcc"
export CXX="g++"
export FC="gfortran"
export F77="${FC}"

# netCDF
if [[ "x${NETCDF_HOME}" == "x" ]]; then
    export NETCDF_HOME="${NETCDF_C_HOME}"
fi
export NETCDF_C_ROOT="${NETCDF_HOME}"
export NETCDF_FORTRAN_ROOT="${NETCDF_FORTRAN_HOME}"

# KPP 3.0.0+
export KPP_FLEX_LIB_DIR="${FLEX_HOME}/lib64"

#####
# Set limits
#####

ulimit -c unlimited    # coredumpsize
ulimit -u 50000        # maxproc
ulimit -v unlimited    # vmemoryuse
ulimit -s unlimited    # stacksize

#####
# Print information

```

(continues on next page)

(continued from previous page)

```
#####
module list
```

Tip: Ask your sysadmin how to load software libraries. If you are using your institution's computer cluster, then chances are there will be a software module system installed, with commands similar to those listed above.

Then you can activate these settings from the command line by typing:

```
$ . ~/gcclassic.gnu10.env
```

You may also place the above command within your *GEOS-Chem run script*, which will be discussed in a subsequent chapter.

4.3 Sample environment file for Intel 2023 compilers

Below is a sample environment file (based on an environment file for the Harvard Cannon computer cluster). This file will load software libraries built with the Intel 2023 compilers.

Add the code below (with the appropriate modifications for your system) into a file named `~/gcclassic.intel23.env`.

```
#####
# Load software packages (EDIT AS NEEDED)
#####

# Unload all modules first
module purge

# Load modules
module load intel/23.0.0-fasrc01          # icc / i++ / gfortran
module load intelmpi/2021.8.0-fasrc01    # MPI
module load netcdf-fortran/4.6.0-fasrc03  # netCDF-Fortran
module load flex/2.6.4-fasrc01           # Flex lexer (needed for KPP)
module load cmake/3.25.2-fasrc01         # CMake (needed to compile)

#####
# Environment variables and related settings
# (NOTE: Lmod will define <module>_HOME variables for each loaded module)
#####

# Make all files world-readable by default
umask 022

# Set number of threads for OpenMP. If running in a SLURM environment,
# use the number of requested cores. Otherwise use 8 cores for OpenMP.
if [[ "x${SLURM_CPUS_PER_TASK}" == "x" ]]; then
    export OMP_NUM_THREADS=8
else
    export OMP_NUM_THREADS="${SLURM_CPUS_PER_TASK}"
fi

# Max out the stacksize memory limit
export OMP_STACKSIZE="500m"
```

(continues on next page)

(continued from previous page)

```

# Compilers
export CC="icx"
export CXX="icx"
export FC="ifort"
export F77="${FC}"

# netCDF
if [[ "x${NETCDF_HOME}" == "x" ]]; then
    export NETCDF_HOME="${NETCDF_C_HOME}"
fi
export NETCDF_C_ROOT="${NETCDF_HOME}"
export NETCDF_FORTRAN_ROOT="${NETCDF_FORTRAN_HOME}"

# KPP 3.0.0+
export KPP_FLEX_LIB_DIR="${FLEX_HOME}/lib64"

#=====
# Set limits
#=====

ulimit -c unlimited      # coredumpsize
ulimit -u 50000          # maxproc
ulimit -v unlimited      # vmemoryuse
ulimit -s unlimited      # stacksize

#=====
# Print information
#=====

module list

```

Tip: Ask your sysadmin how to load software libraries. If you are using your institution's computer cluster, then chances are there will be a software module system installed, with commands similar to those listed above.

Then you can activate these settings from the command line by typing:

```
$ . ~/gcclassic.intel23.env
```

You may also place the above command within your *GEOS-Chem run script*, which will be discussed in a subsequent chapter.

4.4 Set environment variables for compilers

The sample *GNU* and *Intel* environment files set the **environment variables** listed below in order to select the desired C, C++, and Fortran compilers:

Table 1: Environment variables that specify compilers

Variable	Specifies the:	GNU name	Intel name
CC	C compiler	gcc	icx
CXX	C++ compiler	g++	icx
FC	Fortran compiler	gfortran	ifort

Note: GEOS-Chem Classic only requires the Fortran compiler. But you will also need the C and C++ compilers if you plan to build other software packages (*such as KPP*) or *install libraries manually*.

Also, older Intel compiler versions used `icc` as the name for the C compiler and `icpc` as the name of the C++ compiler. These names have been deprecated in Intel 2023 and will be removed from future Intel compiler releases.

The commands used to define `CC`, `CXX`, and `FC` are:

```
# for GNU
export CC=gcc
export CXX=g++
export FC=gfortran
```

or

```
# for Intel
export CC=icx
export CXX=icx
export FC=ifort
```

4.5 Set environment variables for parallelization

GEOS-Chem Classic uses [OpenMP parallelization](#), which is an implementation of shared-memory (aka serial) parallelization.

Important: OpenMP-parallelized programs (such as GEOS-Chem Classic) cannot execute on more than 1 computational node. Most modern computational nodes typically contain between 16 and 64 cores. Therefore, **GEOS-Chem Classic** simulations will not be able to take advantage of more cores than these.

We recommend that you consider using [GCHP](#) for more computationally-intensive simulations.

In the the sample environment files for [GNU](#) and [Intel](#), we define the following **environment variables** for OpenMP parallelization:

OMP_NUM_THREADS

The `OMP_NUM_THREADS` environment variable sets the number of computational cores (aka threads) that you would like GEOS-Chem Classic to use.

For example, the command below will tell **GEOS-Chem Classic** to use 8 cores within parallel sections of code:

```
$ export OMP_NUM_THREADS=8
```

We recommend that you define `OMP_NUM_THREADS` not only in your environment file, but also in your [GEOS-Chem run script](#).

OMP_STACKSIZE

In order to use **GEOS-Chem Classic** with [OpenMP parallelization](#), you must request the maximum amount of stack memory in your Unix environment. (The stack memory is where local automatic variables and temporary `$OMP PRIVATE` variables will be created.)

Add the following lines to your system startup file (e.g. `.bashrc`) and to your [GEOS-Chem run scripts](#):

```
ulimit -s unlimited
export OMP_STACKSIZE=500m
```

The **ulimit -s unlimited** will tell the bash shell to use the maximum amount of stack memory that is available.

The environment variable `OMP_STACKSIZE` must also be set to a very large number. In this example, we are nominally requesting 500 MB of memory. But in practice, this will tell the GNU Fortran compiler to use the maximum amount of stack memory available on your system. The value **500m** is a good round number that is larger than the amount of stack memory on most computer clusters, but you can increase this if you wish.

4.5.1 Errors caused by incorrect environment variable settings

Be on the lookout for these errors:

1. If `OMP_NUM_THREADS` is set to 1, then your simulation will execute using only one computational core. This will make your simulation take much longer than necessary.
2. If `OMP_STACKSIZE` environment variable is not included in your environment file (or if it is set to a very low value), you might encounter a [segmentation fault error after the TPCORE transport module is initialized](#). In this case, GEOS-Chem Classic “thinks” that it does not have enough memory to perform the simulation, even though sufficient memory may be present.

KEY REFERENCES

Bey *et al.* [[Bey et al., 2001]] is the first reference to GEOS-Chem that includes a detailed model description. It is suitable as an original reference for the model. It only describes a model for gas-phase tropospheric oxidant chemistry. Subsequent original references for major additional model features are:

1. Park *et al.* [[Park et al., 2004]] for aerosol chemistry;
2. Y.X. Wang *et al.* [[Wang et al., 2004]] for the nested model;
3. Henze *et al.* [[Henze et al., 2007]] for the model adjoint;
4. Selin *et al.* [[Selin et al., 2007]] for the mercury simulation;
5. Trivitayanurak *et al.* [[Trivitayanurak et al., 2008]] for TOMAS aerosol microphysics;
6. Yu and Luo [[Yu and Luo 2009]] for APM aerosol microphysics;
7. Eastham *et al.* [[Eastham et al., 2014]] and for stratospheric chemistry;
8. Keller *et al.* [[Keller et al., 2014]] and Lin *et al.* [[Lin et al., 2021]] for HEMCO;
9. Long *et al.* [[Long et al., 2015]] for the grid-independent GEOS-Chem;
10. Eastham *et al.* [[Eastham et al., 2018]] for the high-performance GEOS-Chem (GCHP);
11. Hu *et al.* [[Hu et al., 2018]] for GEOS-Chem within the GEOS ESM (GEOS-GC);
12. Lin *et al.* [[Lin et al., 2020]] for GEOS-Chem within WRF (WRF-GC);
13. Zhuang *et al.* [[Zhuang et al., 2019]] and Zhuang *et al.* [[Zhuang et al., 2020]] for implementations of GEOS-Chem Classic and GCHP on the cloud;
14. Bindle *et al.* [[Bindle et al., 2021]] for the stretched-grid capability in GCHP;
15. Murray *et al.* [[Murray et al., 2021]] for GEOS-Chem driven by GISS GCM fields (GCAP 2.0);
16. Bukosa *et al.* [[Bukosa et al., 2023]] for the carbon simulation;
17. Lin *et al.* [[Lin et al., 2023]] for KPP 3.0.0 with adaptive auto-reduction solver.

References

DOWNLOAD SOURCE CODE

In the following chapters, you will learn how to download the GEOS-Chem source code from Github.

6.1 Source code repositories

The **GEOS-Chem Classic** source code is distributed into 3 Github repositories, as described below. This setup allows the GEOS-Chem core science routines to be easily integrated into several modeling contexts, such as:

- GEOS-Chem Classic
- GCHP
- GEOS-Chem within the NASA/GEOS ESM
- GEOS-Chem within CESM
- GEOS-Chem withn WRF (aka WRF-GC)

This repository setup also aligns with our [GEOS-Chem Vision](#) and [Mission](#) statements.

6.1.1 GEOS-Chem Science Codebase

The GEOS-Chem “Science” Codebase repository (<https://github.com/geoschem/geos-chem>) contains the GEOS-Chem science routines, plus:

- Scripts to create GEOS-Chem run directories, plus template configuration files for all simulations;
- Scripts to create GEOS-Chem integration tests;
- Interfaces (i.e. the driver programs) for GEOS-Chem Classic, GCHP, etc.

6.1.2 HEMCO

The HEMCO repository (<https://github.com/geoschem/HEMCO>) contains the source code for the [Harmonized Emissions Component](#), which is used to read and regrid emissions, met fields, and other inputs to GEOS-Chem.

6.1.3 GCClassic

The GCClassic repository (<https://github.com/geoschem/GCClassic>) is a lightweight wrapper that encompasses GEOS-Chem and HEMCO. We say that GCClassic is the **superproject** (i.e. top-level source code folder), and that GEOS-Chem (science codebase) and HEMCO are **submodules**.

6.2 Download instructions

Follow these directions to download the GEOS-Chem Classic source code.

6.2.1 Clone GCClassic and fetch submodules

To download the latest stable GEOS-Chem Classic version, type:

```
$ git clone --recurse-submodules https://github.com/geoschem/GCClassic.git
```

This command does the following:

1. Clones the *GCClassic* repo from GitHub to a local folder named GCClassic;
2. Clones the *GEOS-Chem Science Codebase* repo from GitHub to GCClassic/src/GEOS-Chem; and
3. Clones the *HEMCO* repo from GitHub to GCClassic/src/HEMCO.

Tip: To download GEOS-Chem Classic source code into a folder named something other than GCClassic, supply the name of the folder at the end of the **git clone** command. For example:

```
git clone --recurse-submodules https://github.com/geoschem/GCClassic.git my-code-dir
```

will download the GEOS-Chem Classic source code into my-code-dir instead of GCClassic.

Once the **git clone** process starts, you should see output similar to this:

```
Cloning into 'GCClassic'...
remote: Enumerating objects: 2680, done.
remote: Counting objects: 100% (1146/1146), done.
remote: Compressing objects: 100% (312/312), done.
remote: Total 2680 (delta 858), reused 1099 (delta 825), pack-reused 1534
Receiving objects: 100% (2680/2680), 1.74 MiB | 13.16 MiB/s, done.
Resolving deltas: 100% (1411/1411), done.
Submodule 'docs/source/geos-chem-shared-docs' (https://github.com/geoschem/geos-chem-
↳ shared-docs.git) registered for path 'docs/source/geos-chem-shared-docs'
Submodule 'src/GEOS-Chem' (https://github.com/geoschem/geos-chem.git) registered for
↳ path 'src/GEOS-Chem'
Submodule 'src/HEMCO' (https://github.com/geoschem/hemco.git) registered for path
↳ 'src/HEMCO'
Cloning into '/local/ryantosca/GC/rundirs/epa-kpp/tmp/GCClassic/docs/source/geos-chem-
↳ shared-docs'...
remote: Enumerating objects: 148, done.
remote: Counting objects: 100% (148/148), done.
remote: Compressing objects: 100% (103/103), done.
remote: Total 148 (delta 77), reused 116 (delta 45), pack-reused 0
Receiving objects: 100% (148/148), 162.29 KiB | 2.90 MiB/s, done.
Resolving deltas: 100% (77/77), done.
```

(continues on next page)

(continued from previous page)

```

Cloning into '/local/ryantosca/GC/rundirs/epa-kpp/tmp/GCClassic/src/GEOS-Chem'...
remote: Enumerating objects: 75574, done.
remote: Counting objects: 100% (410/410), done.
remote: Compressing objects: 100% (187/187), done.
remote: Total 75574 (delta 238), reused 364 (delta 216), pack-reused 75164
Receiving objects: 100% (75574/75574), 85.23 MiB | 30.59 MiB/s, done.
Resolving deltas: 100% (62327/62327), done.
Cloning into '/local/ryantosca/GC/rundirs/epa-kpp/tmp/GCClassic/src/HEMCO'...
remote: Enumerating objects: 3178, done.
remote: Counting objects: 100% (638/638), done.
remote: Compressing objects: 100% (195/195), done.
remote: Total 3178 (delta 476), reused 585 (delta 438), pack-reused 2540
Receiving objects: 100% (3178/3178), 2.24 MiB | 11.87 MiB/s, done.
Resolving deltas: 100% (2270/2270), done.
Submodule path 'docs/source/geos-chem-shared-docs': checked out
  ↪ '228507857eb53740dacf4055ce9268aa8ccf520d'
Submodule path 'src/GEOS-Chem': checked out '7e51a0674aba638c8322fef493ac9251095e8cf4'
Submodule path 'src/HEMCO': checked out '4a66bae48f33e6dc22cda5ec9d4633192dee2f73'
Submodule 'docs/source/geos-chem-shared-docs' (https://github.com/geoschem/geos-chem-
  ↪ shared-docs.git) registered for path 'src/HEMCO/docs/source/geos-chem-shared-docs'
Cloning into '/local/ryantosca/GC/rundirs/epa-kpp/tmp/GCClassic/src/HEMCO/docs/source/
  ↪ geos-chem-shared-docs'...
remote: Enumerating objects: 148, done.
remote: Counting objects: 100% (148/148), done.
remote: Compressing objects: 100% (103/103), done.
remote: Total 148 (delta 77), reused 116 (delta 45), pack-reused 0
Receiving objects: 100% (148/148), 162.29 KiB | 3.00 MiB/s, done.
Resolving deltas: 100% (77/77), done.
Submodule path 'src/HEMCO/docs/source/geos-chem-shared-docs': checked out
  ↪ '645401baa35b6a6838b9bedede309a01a311517f'

```

When the **git clone** process has finished, navigate into the GCClassic folder and get a directory listing:

```

$ cd GCClassic
$ ls -CF src/*

```

and you will see output similar to this:

```

src/CMakeLists.txt  src/gc_classic_version.H@  src/main.F90@

src/GEOS-Chem:
APM/              CMakeScripts/  GeosUtil/      History/        lib/            ObsPack/        run/
AUTHORS.txt       doc/           GTMM/          Interfaces/     LICENSE.txt     PKUCPL/
bin/              GeosCore/     Headers/       ISORROPIA/     mod/            README.md
CMakeLists.txt    GeosRad/      help/          KPP/           NcdfUtil/      REVISIONS

src/HEMCO:
AUTHORS.txt  CMakeLists.txt  CMakeScripts/  LICENSE.txt  README.md  run/  src/

```

This confirms that the GCClassic/src/GEOS-Chem and GCClassic/src/HEMCO folders have been populated with source code from the *GEOS-Chem Science Codebase* and *HEMCO* GitHub repositories.

Tip: To use an older GEOS-Chem Classic version (e.g. 14.0.0), follow these additional steps:

```

$ git checkout tags/14.0.0          # Points HEAD to the tag "14.0.0"
$ git branch version_14.0.0        # Creates a new branch at tag "14.0.0"

```

(continues on next page)

(continued from previous page)

```
$ git checkout version_14.0.0      # Checks out the version_14.0.0 branch
$ git submodule update --init --recursive  # Reverts submodules to the "14.0.0" tag
```

You can do this for any tag in the version history. For a list of all tags, type:

```
$ git tag
```

If you have any unsaved changes, make sure you commit those to a branch prior to updating versions.

6.2.2 Create a branch in src/GEOS-Chem for your work

When the `git clone` command *described above* finishes, the *GEOS-Chem Science Codebase* submodule code (in folder `GCClassic/src/GEOS-Chem`) and the *HEMCO* submodule code (in folder `GCClassic/src/HEMCO`) will be in **detached HEAD state**. In other words, the code is checked out but a branch is not created. Adding new code to a detached HEAD state is very dangerous and should be avoided. You should instead make a branch at the same point as the detached HEAD, and then add your own modifications into that branch.

Navigate from `GCClassic` to `GCClassic/src/GEOS-Chem`:

```
$ cd src/GEOS-Chem
```

and then type:

```
$ git branch
```

You will see output similar to this:

```
* (HEAD detached at xxxxxxxx)
main
```

where `xxxxxxx` denotes the hash of the commit at which the code has been checked out.

At this point, you may now create a branch in which to store your own modifications to the GEOS-Chem science codebase. Type:

```
$ git branch feature/my-git-updates
$ git checkout feature/my-git-updates
```

Note: This naming convention adheres to the [Github Flow](#) conventions (i.e. new feature branches start with `feature/`, bug fix branches start with `bugfix/`, etc).

Instead of `feature/my-git-updates`, you may choose a name that reflects the nature of your updates (e.g. `feature/new_reactions`, etc.) If you now type:

```
$ git branch
```

You will see that we are checked out onto the branch that you just created and are no longer in detached HEAD state.

```
* feature/my-git-updates
main
```

At this point, you may proceed to add your modifications into the GEOS-Chem Science Codebase.

Note: If you need to also modify *HEMCO* source code, repeat the process above to create your own working branch in `GCClassic/src/HEMCO`.

6.2.3 See additional resources

For more information about downloading the GEOS-Chem source code, please see the following Youtube video tutorials:

- [Getting started with GEOS-Chem 13](#) (by Melissa Sulprizio)
- [Managing branches between superproject and submodules](#) (by Bob Yantosca)

CREATE A RUN DIRECTORY

We have greatly simplified run directory creation in [GEOS-Chem Classic 13.0.0](#) and later versions. You no longer need to download the separate GEOS-Chem Unit Tester repository, but can create run directories from a script in the GEOS-Chem source code itself.

Please see the following sections for more information on how to create run directories for GEOS-Chem Classic simulations:

7.1 First-time user registration

We have introduced a online user registration system starting with GEOS-Chem Classic 14.0.0. The first time that you create a run directory, you will be prompted to provide contact information and a summary about how you plan to use GEOS-Chem. This information will be kept at a secure cloud-based server.

Even if you are a long-time GEOS-Chem user, we ask that you answer all of the questions. Your responses will help us to keep an accurate count of GEOS-Chem users and to keep the [list of GEOS-Chem users](#) current.

The user registration dialog (and where you will type in your responses) is shown below.

```
=====
GEOS-CHEM RUN DIRECTORY CREATION
=====

Initiating User Registration:
You will only need to fill this information out once.
Please respond to all questions.

-----
What is your name?
-----
>>> type your response and hit ENTER

-----
What is your email address?
-----
>>> type your response and hit ENTER

-----
What is the name of your research institution?
-----
>>> type your response and hit ENTER

-----
```

(continues on next page)

(continued from previous page)

```

What is the name of your principal investigator?
(Enter 'self' if you are the principal investigator.)
-----
>>> type your response and hit ENTER

-----

Please provide the web site for your institution
(group website, company website, etc.)?
-----
>>> type your response and hit ENTER

-----

Please provide your github username (if any) so that we
can recognize you in submitted issues and pull requests.
-----
>>> type your response and hit ENTER

-----

Where do you plan to run GEOS-Chem?
(e.g. local compute cluster, AWS, other supercomputer)?
-----
>>> type your response and hit ENTER

-----

Please briefly describe how you plan on using GEOS-Chem
so that we can add you to 'GEOS-Chem People and Projects'
(https://geoschem.github.io/geos-chem-people-projects-map/)
-----
>>> type your response and hit ENTER
Successful Registration

```

If you do not see the Successful Registraton message, check your internet connection and try again. If the problem persists, [open a new Github issue](#).

7.2 Example: Create a full-chemistry simulation run directory

Let us walk through the process of creating a run directory for a global GEOS-Chem full-chemistry simulation.

1. Navigate to the GCClassic superproject folder and get a directory listing:

```

$ cd /path/to/your/GCClassic
$ ls -CF

```

You should see this output:

AUTHORS.txt	CMakeScripts/	LICENSE.txt	SUPPORT.md	run@	test@
CMakeLists.txt	CONTRIBUTING.md	README.md	docs/	src/	

As mentioned previously, run@ is a symbolic link. It actually points to the to the src/GEOS-Chem/run/ GCClassic folder. This folder contains several scripts and template files for run directory creation.

2. Navigate to the run folder and get a directory listing:

```

$ cd run
$ ls -CF

```

and you should see this output:

```
HEMCO_Config.rc.templates/  geoschem_config.yml.templates/
HEMCO_Diagn.rc.templates/  getRunInfo*
HISTORY.rc.templates/      gitignore
README                     init_rd.sh*
archiveRun.sh*             runScriptSamples/
createRunDir.sh*
```

You can see several folders (highlighted in the directory display with /) and a few executable scripts (highlighted with *). The script we are interested in is `createRunDir.sh`.

3. Run the `createRunDir.sh` script. Type:

```
$ ./createRunDir.sh
```

4. You will then be prompted to supply information about the run directory that you wish to create:

```
=====
GEOS-CHEM RUN DIRECTORY CREATION
=====

-----
Choose simulation type:
-----

  1. Full chemistry
  2. Aerosols only
  3. CH4
  4. CO2
  5. Hg
  6. POPs
  7. Tagged CH4
  8. Tagged CO
  9. Tagged O3
 10. TransportTracers
 11. Trace metals
 12. Carbon
>>>
```

To create a run directory for the full-chemistry simulation, type **1** followed by the **ENTER** key.

Tip: To exit, the run directory creation process, type `Ctrl-C` at any prompt.

5. You will then be asked to specify any additional options for the full-chemistry simulation (such as adding the RRTMG radiative transfer model, APM or TOMAS microphysics, etc.)

```
-----
Choose additional simulation option:
-----

  1. Standard
  2. Benchmark
  3. Complex SOA
  4. Marine POA
```

(continues on next page)

(continued from previous page)

```

5. Acid uptake on dust
6. TOMAS
7. APM
8. RRTMG
>>>

```

For the standard full-chemistry simulation, type **1** followed by **ENTER**.

To add an option to the full-chemistry simulation, type a number between **2** and **8** and press **ENTER**.

6. You will then be asked to specify the meteorology type for the simulation (**GEOS-FP**, **MERRA-2**), or **GCAP 2.0**):

```

-----
Choose meteorology source:
-----
1. MERRA-2 (Recommended)
2. GEOS-FP
3. GISS ModelE2.1 (GCAP 2.0)
>>>

```

You should use the recommended option (MERRA-2) if possible. Type **1** followed by **ENTER**.

7. The next menu will prompt you for the horizontal resolution that you wish to use:

```

-----
Choose horizontal resolution:
-----
1. 4.0 x 5.0
2. 2.0 x 2.5
3. 0.5 x 0.625
>>>

```

If you wish to set up a global simulation, type either **1** or **2** followed by **ENTER**.

If you wish to set up a nested-grid simulation, type **3** and hit **ENTER**. Then you will be followed by a nested-grid menu:

```

-----
Choose horizontal grid domain:
-----
1. Global
2. Asia
3. Europe
4. North America
5. Custom
>>>

```

Select your preferred horizontal domain, followed by **ENTER**.

8. You will then be prompted for the vertical dimension of the grid.

```

-----
Choose number of levels:
-----
1. 72 (native)
2. 47 (reduced)
>>>

```


For most simulations, you will want to use **72** levels. Type **1** followed by **ENTER**.

For some memory-intensive simulations (such as nested-grid simulations), you can use 47 levels. Type **2** followed by **ENTER**.

9. You will then be prompted for the folder in which you wish to create the run directory.

```
-----
Enter path where the run directory will be created:
-----
>>>
```

You may enter an absolute path (such as \$HOME/myusername/ followed by **ENTER**).

You may also enter a relative path (such as ~/rundirs followed by **ENTER**). In this case you will see that the ./createRunDir.sh script will expand the path to:

```
Expanding to: /n/home09/myusername/rundirs |br|
|br|
```

10. The next menu will prompt you for the run directory name.

```
-----
Enter run directory name, or press return to use default:

NOTE: This will be a subfolder of the path you entered above.
-----
>>>
```

You should use the default run directory name whenever possible. Type **ENTER** to select the default.

The script will display the following output:

```
-- Using default directory name gc_4x5_merra2_fullchem
```

or if you are creating a nested grid simulation:

```
-- Using default directory name gc_05x0625_merra2_fullchem
```

and then:

```
-- This run directory has been set up for 20190701 - 20190801.
   You may modify these settings in geoschem_config.yml.

-- The default frequency and duration of diagnostics is set to monthly.
   You may modify these settings in HISTORY.rc and
   HEMCO_Config.rc.
```

11. The last menu will prompt you with:

```
-----
Do you want to track run directory changes with git? (y/n)
-----
```

Type **y** and then **ENTER**. Then you will be able to track changes that you make to GEOS-Chem configuration files with Git. This can be a lifesaver when debugging – you can revert to an earlier state and then start fresh.

12. The script will display the full path to the run directory. You can navigate there and then start editing the *GEOS-Chem configuration files*.

7.3 Example: Create a CH4 simulation run directory

The process of creating run directories for the GEOS-Chem specialty simulations is similar to that as listed in Example 1 above. However, the number of menus that you need to select from will likely be fewer than for the full-chemistry simulation. We'll use the methane simulation as an example.

1. Navigate to the GCClassic superproject folder and get a directory listing:

```
$ cd /path/to/your/GCClassic
$ ls -CF
```

You should see this output:

```
AUTHORS.txt      CMakeScripts/    LICENSE.txt      SUPPORT.md      run@    test@
CMakeLists.txt   CONTRIBUTING.md  README.md       docs/          src/
```

As mentioned previously, `run@` is a symbolic link. It actually points to the `src/GEOS-Chem/run/` GCClassic folder. This folder contains several scripts and template files for run directory creation.

2. Navigate to the run folder and get a directory listing:

```
$ cd run
$ ls -CF
```

and you should see this output:

```
HEMCO_Config.rc.templates/  geoschem_config.yml.templates/
HEMCO_Diagn.rc.templates/  getRunInfo*
HISTORY.rc.templates/      gitignore
README                     init_rd.sh*
archiveRun.sh*             runScriptSamples/
createRunDir.sh*
```

You can see several folders (highlighted in the directory display with `/`) and a few executable scripts (highlighted with `*`). The script we are interested in is `createRunDir.sh`.

3. Run the **createRunDir.sh** script.. Type:

```
$ ./createRunDir.sh
```

4. You will then be prompted to supply information about the run directory that you wish to create:

```
=====
GEOS-CHEM RUN DIRECTORY CREATION
=====

-----
Choose simulation type:
-----

 1. Full chemistry
 2. Aerosols only
 3. CH4
 4. CO2
 5. Hg
 6. POPs
 7. Tagged CH4
```

(continues on next page)

(continued from previous page)

```

8. Tagged CO
9. Tagged O3
10. TransportTracers
11. Trace metals
12. Carbon
>>>

```

To select the GEOS-Chem methane specialty simulation, type **3** followed by **ENTER**.

Tip: To exit, the run directory creation process, type `Ctrl-C` at any prompt.

5. You will then be asked to specify the meteorology type for the simulation (**GEOS-FP**, **MERRA-2**), or GCAP 2.0):

```

-----
Choose meteorology source:
-----
1. MERRA-2 (Recommended)
2. GEOS-FP
3. GISS ModelE2.1 (GCAP 2.0)
>>>

```

To accept the recommended meteorology (MERRA-2), type **1** followed by **ENTER**.

6. The next menu will prompt you for the horizontal resolution that you wish to use:

```

-----
Choose horizontal resolution:
-----
1. 4.0 x 5.0
2. 2.0 x 2.5
3. 0.5 x 0.625
>>>

```

If you wish to set up a global simulation, type either **1** or **2** followed by **ENTER**.

If you wish to set up a nested-grid simulation, type **3** and hit **ENTER**. Then you will be followed by a nested-grid menu:

```

-----
Choose horizontal grid domain:
-----
1. Global
2. Asia
3. Europe
4. North America
5. Custom
>>>

```

Type the number of your preferred option and then hit **ENTER**.

7. You will then be prompted for the vertical dimension of the grid.

```
-----
Choose number of levels:
-----
```

```
  1. 72 (native)
  2. 47 (reduced)
>>>
```

For most simulations, you will want to use 72 levels. Type **1** followed by **ENTER**.

For some memory-intensive simulations (such as nested-grid simulations), you can use 47 levels. Type **2** followed by **ENTER**.

8. You will then be prompted for the folder in which you wish to create the run directory.

```
-----
Enter path where the run directory will be created:
-----
```

```
>>>
```

You may enter an absolute path (such as `$HOME/myusername/` followed by **ENTER**).

You may also enter a relative path (such as `~/rundirs` followed by **ENTER**). In this case you will see that the `./createRunDir.sh` script will expand the path to:

```
Expanding to: /n/home09/myusername/rundirs
```

9. The next menu will prompt you for the run directory name.

```
-----
Enter run directory name, or press return to use default:
-----
```

```
NOTE: This will be a subfolder of the path you entered above.
-----
```

```
>>>
```

You should use the default run directory name whenever possible. Type **ENTER**. The script will display the following output:

```
-- Using default directory name gc_4x5_merra2_CH4
```

or if you are creating a nested grid simulation:

```
-- Using default directory name gc_05x0625_merra2_CH4
```

and then

```
-- This run directory has been set up for 20190701 - 20190801.
   You may modify these settings in geoschem_config.yml.

-- The default frequency and duration of diagnostics is set to monthly.
   You may modify these settings in HISTORY.rc and HEMCO_Config.rc.
```

10. The last menu will prompt you with:

```

-----
Do you want to track run directory changes with git? (y/n)
-----
>>>

```

Type **y** and then **ENTER**. Then you will be able to track changes that you make to GEOS-Chem configuration files with Git. This can be a lifesaver when debugging – you can revert to an earlier state and then start fresh.

11. The script will display the full path to the run directory. You can navigate there and then start editing the *GEOS-Chem configuration files*.

The procedure to set up run directories for other GEOS-Chem Classic simulations is similar to that shown above.

7.4 Run directory files and folders

Each GEOS-Chem Classic run directory that you create will contain the files and folders listed below. The *GEOS-Chem and HEMCO configuration files* in the run directory will be appropriate to the type of simulation that you have selected.

archiveRun.sh

This script can be used to create an archive of the run directory. Run this script with:

```
$ ./archiveRun.sh directory-name
```

Where `directory-name` is the name of the archive folder. This can be either a relative path or an absolute path.

build/

This is a blank directory where you can direct **CMake** to *configure and build* the GEOS-Chem source code.

build_info/

This folder is created when you *compile GEOS-Chem*. It contains information about the options that were passed to **CMake** during the configuration and build process.

cleanRunDir.sh

Typing

```
$ ./cleanRunDir.sh
```

will remove log files and diagnostic output files left over from a previous GEOS-Chem simulation.

CodeDir

Symbolic link to the top-level source code folder (i.e. the `GCClassic` superproject folder).

CreateRunDirLogs/rundir_vars.txt

Log file containing environment variable settings used in run directory creation. Running the `init_rd.sh` script on this file will create a duplicate run directory.

download_data.py

Use this Python script to download data from one of the GEOS-Chem data portals to your disk space. See our *Download data with a dry-run simulation* chapter for more information.

download_data.yml

Configuration file for `download_data.py`.

geoschem_config.yml

The main GEOS-Chem configuration file (see *Configure your simulation*).

getRunInfo

This file is now deprecated and will be removed in a future version.

HEMCO_Config.rc

The main HEMCO configuration file (see [Configure your simulation](#)).

HEMCO_Config.rc.gmao_metfields

HEMCO configuration file snippet containing entries for reading the GMAO meteorological fields. This file will only be present if you are using GEOS-FP or MERRA-2 meteorology to drive your GEOS-Chem simulation.

HEMCO_Config.rc.gcap2_metfields

HEMCO configuration file snippet containing entries for reading the GCAP2 meteorological fields. This file will only be present if you are using GCAP2 meteorology to drive your GEOS-Chem simulation.

HEMCO_Diagn.rc

Configuration file for HEMCO diagnostics (see [Configure your simulation](#)).

HISTORY.rc

Configuration file for GEOS-Chem History diagnostics (see [Configure your simulation](#)).

metrics.py

This Python script can be used to print the OH metrics for a full-chemistry simulation. Typing:

```
$ ./metrics.py
```

will generate output such as:

```
=====
GEOS-Chem FULL-CHEMISTRY SIMULATION METRICS

Simulation start : 2019-07-01 00:00:00z
Simulation end   : 2019-07-01 01:00:00z
=====

Mass-weighted mean OH concentration      = 10.04682154969 x 10^5 molec cm-3

CH3CCl3 lifetime w/r/t tropospheric OH = 6.3189 years

CH4 lifetime w/r/t tropospheric OH      = 10.6590 years
```

OutputDir/

Blank directory where GEOS-Chem diagnostic output files will be created.

README.md

README file (in Markdown format) with containing links to information about GEOS-Chem.

Restarts/

Directory where GEOS-Chem [restart files](#) will be created.

Restarts/GEOSChem.Restart.YYYYMMDD_hhmmzz.nc4

[Restart file](#) containing initial conditions for the GEOS-Chem simulation.

Attention: The restart file that is created when you generate a run directory should not be used to start a production simulation. We recommend that you “spin up” your simulation for at least 6 months to a year in order to remove the signature of the initial conditions.

runScriptSamples

Symbolic link to the folder in the [GEOS-Chem “Science Codebase”](#) repository that contains [sample scripts](#) for running GEOS-Chem.

species_database.yml

YAML file containing metadata (e.g. molecular weight, Henry's law constants, wetdep and drydep parameters, etc.) for each species used in the various GEOS-Chem simulations. You should not have to edit this file unless you are adding new species to your GEOS-Chem simulation. The *species_database.yml* file will be discussed in more detail in a following section.

COMPILE THE SOURCE CODE

In this chapter, we will describe how you can compile GEOS-Chem Classic. Compiling creates an **executable file** that you can run on your computer system.

The compilation process involves the following steps:

8.1 Configure with CMake

You should think of **CMake** as an interactive tool for configuring GEOS-Chem Classic's build. For example, compile-time options like disabling multithreading and turning on components (e.g. APM, RRTMG) are all configured with CMake commands.

Besides configuring GEOS-Chem's build, CMake also performs checks on your build environment to detect problems that would cause the build to fail. If it identifies a problem, like a missing dependency or mismatched run directory and source code version numbers, CMake will print an error message that describes the problem.

If you are new to CMake and would like a rundown of how to use the **cmake** command, check out [Liam Bindle's Cmake Tutorial](#). This tutorial is not necessary, but it will make you more familiar with using CMake and help you better understand what is going on.

Below are the steps for building GEOS-Chem with CMake.

8.1.1 Navigate to your run directory

In this example, we will compile the GEOS-Chem code for the full-chemistry simulation. Navigate to your run directory and get a directory listing as shown below:

```
$ cd /path/to/gc_4x5_merra2_fullchem
$ ls
archiveRun.sh*      GEOSChem.Restart.20190701_0000z.nc4  metrics.py
build/              getrunInfo*                          OutputDir/
cleanRunDir.sh*    HEMCO_Config.rc                     README
CodeDir@           HEMCO_Diagn.rc                      rundir.version
download_data.py*  HISTORY.rc                          runScriptSamples
download_data.yml  input.geos                          species_database.yml
```

Note that each GEOS-Chem run directory that you generate has a folder named `build/`. This is where we will run CMake.

8.1.2 Navigate to the build directory

The build directory is where CMake and your compilers are going to put the files they generate. For this example, we will use the `build/` folder that was automatically generated in the GEOS-Chem Classic run directory. For GCHP you will need to create one.

```
$ cd build
```

Tip: If you find yourself switching between different compilers, you can create multiple build directories with different names (e.g. `build_gfortran10`, `build_ifort19`, etc).

8.1.3 Initialize the build directory

Next, we need to initialize the build directory. Type:

```
$ cmake ../CodeDir -DRUNDIR=..
```

where `../CodeDir` is the symbolic link from our run directory to the GEOS-Chem source code directory. CMake will generate output similar to this:

```
-- The Fortran compiler identification is GNU 11.2.0
-- Detecting Fortran compiler ABI info
-- Detecting Fortran compiler ABI info - done
-- Check for working Fortran compiler: /n/home09/ryantosca/spack/var/spack/
↳environments/gc-classic/.spack-env/view/bin/gfortran - skipped
-- Checking whether /n/home09/ryantosca/spack/var/spack/environments/gc-classic/.
↳spack-env/view/bin/gfortran supports Fortran 90
-- Checking whether /n/home09/ryantosca/spack/var/spack/environments/gc-classic/.
↳spack-env/view/bin/gfortran supports Fortran 90 - yes
=====
GCCclassic X.Y.Z (superproject wrapper)
Current status: X.Y.Z
=====
-- Found NetCDF: /n/home09/ryantosca/spack/opt/spack/linux-centos7-x86_64/gcc-8.3.0/
↳netcdf-fortran-4.5.3-tb3oqspkitgcbkcyp623tdq2a16gxmom/lib/libnetcdf.so
-- Useful CMake variables:
+ CMAKE_PREFIX_PATH: /path/to/netcdf-c /path/to/netcdf-fortran
+ CMAKE_BUILD_TYPE: Release
-- Run directory setup:
+ RUNDIR: /n/holyscratch01/jacob_lab/ryantosca/tests/test/test_cc
-- Threading:
* OMP: **ON** OFF
-- Found OpenMP_Fortran: -fopenmp (found version "4.5")
-- Found OpenMP: TRUE (found version "4.5")
-- General settings:
* MECH: **fullchem** carbon Hg custom
* BPCH_DIAG: **ON** OFF
* USE_REAL8: **ON** OFF
* SANITIZE: ON **OFF**
-- Components:
* TOMAS: ON **OFF**
* TOMAS_BINS: **NA** 15 40
* APM: ON **OFF**
* RRTMG: ON **OFF**

```

(continues on next page)

(continued from previous page)

```

* GTMM:          ON  **OFF**
* HCOSA:         ON  **OFF**
* LUO_WETDEP:    ON  **OFF**
* FASTJX:        ON  **OFF**
=====
HEMCO A.B.C
Current status: A.B.C
=====
=====
GEOS-Chem T.U.V (science codebase)
Current status: T.U.V
=====
Creating /n/holyscratch01/jacob_lab/ryantosca/tests/test/test_cc/CodeDir/src/GEOS-
↪Chem/Interfaces/GCClassic/gc_classic_version.H
-- Configuring done
-- Generating done
-- Build files have been written to: /n/holyscratch01/jacob_lab/ryantosca/tests/test/
↪test_cc/build

```

Your CMake command's output contains important information about your build's configuration.

Note: The text X.Y.Z, A.B.C, and T.U.V refer to the version numbers (in [semantic versioning](#) style) of the GCClassic, HEMCO, and GEOS-Chem “science codebase” repositories.

8.1.4 Configure your build with extra options

Your build directory is now configured to compile GEOS-Chem using all default options. If you do not wish to change anything further, you may *skip ahead to the next section*.

However, if you wish to modify your build's configuration, simply invoke CMake once more with optional parameters. Use this format:

```
$ cmake . -DOPTION=value
```

Note that the `.` argument is necessary. It tells CMake that your current working directory (i.e. `.`) is your build directory. The output of **cmake** tells you about your build's configuration. Options are prefixed by a `+` or `*` in the output, and their values are displayed or highlighted.

Tip: If you are colorblind or if you are using a terminal that does not support colors, refer to the CMake FAQ for instructions on disabling colorized output. For a detailed explanation of CMake output, see the next section.

The table below contains the list of GEOS-Chem build options that you can pass to CMake. GEOS-Chem will be compiled with the default build options, unless you explicitly specify otherwise.

RUNDIR

Defines the path to the run directory.

In this example, our build directory is a subfolder of the run directory, so we can use `-DRUNDIR=.` ... If your build directory is somewhere else, then specify the path to the run directory as an absolute path.

CMAKE_BUILD_TYPE

Specifies the type of build. Accepted values are:

Release

Tells CMake to configure GEOS-Chem in **Release** mode. This means that all optimizations will be applied and all debugging options will be disabled. **(Default option)**.

Debug

Turns on several runtime error checks. This will make it easier to find errors but will adversely impact performance. Only use this option if you are actively debugging.

MECH

Specifies the chemical mechanism that you wish to use:

fullchem

Activates the **fullchem** mechanism. The source code files that define this mechanism are stored in KPP / fullchem. **(Default option)**

Hg

Activates the **Hg** mechanism. The source code files that define this mechanism are stored in KPP / Hg.

carbon

Activates the **carbon** mechanism (CH₄-CO-CO₂-OCS). The source code files that define this mechanism are stored in KPP / carbon.

custom

Activates a **custom** mechanism defined by the user. The source code files that define this mechanism are stored in KPP / custom.

OMP

Determines if GEOS-Chem Classic will activate [OpenMP parallelization](#). Accepted values are:

Y

Activates OpenMP parallelization. **(Default option)**

GEOS-Chem Classic will execute on as many computational cores as is specified with [OMP_NUM_THREADS](#).

n

Deactivates OpenMP parallelization. GEOS-Chem Classic will execute on a single computational core. Useful for debugging.

TOMAS

Configure GEOS-Chem with the [TOMAS aerosol microphysics package](#). Accepted values are:

Y

Activate TOMAS microphysics.

n

Deactivate TOMAS microphysics **(Default option)**

TOMAS_BINS

Specifies the number of size-resolved bins for TOMAS. Accepted values are:

15

Use 15 size-resolved bins with TOMAS simulations.

40

Use 40 size-resolved bins with TOMAS simulations.

BPCH_DIAG

Toggles the legacy binary punch diagnostics on.

Attention: This option is deprecated and will be removed soon. Most binary-punch format diagnostics have been replaced by *netCDF-based History diagnostics*.

Accepted values are:

- y** Activate legacy binary-punch diagnostics.
- n** Deactivate legacy binary-punch diagnostics. **(Default option)**

APM

Configures GEOS-Chem to use the [APM microphysics package](#). Accepted values are:

- y** Activate APM microphysics.
- n** Deactivate APM microphysics. **(Default option)**

RRTMG

Configures GEOS-Chem to use the [RRTMG radiative transfer model](#). Accepted values are:

- y** Activates the RRTMG radiative transfer model.
- n** Deactivates the RRTMG radiative transfer model. **(Default option)**

LUO_WETDEP

Configures GEOS-Chem to use the [Luo et al., 2020](#) wet deposition scheme.

Note: The Luo et al 2020 wet deposition scheme will eventually become the default wet deposition scheme in GEOS-Chem. We have made it an option for the time being while further evaluation is being done.

Accepted values are:

- y** Activates the Luo et al., 2020 wet deposition scheme.
- n** Deactivates the Luo et al., 2020 wet deposition scheme. **(Default option)**

FASTJX

Configures GEOS-Chem to use the legacy FAST-JX v7.0 photolysis mechanism instead of its successor Cloud-J.

Note: We recommend using FAST-JX for the mercury simulation instead of Cloud-J. Further work is needed to make the mercury simulation compatible with Cloud-J. Once that work is completed the legacy FAST-JX option will be deleted from the model.

Accepted values are:

- y** Uses the legacy FAST-JX v7.0 photolysis scheme rather than Cloud-J.
- n** Uses the Cloud-J photolysis scheme rather than legacy FAST-JX. **(Default option)**

SANITIZE

Activates the AddressSanitizer/LeakSanitizer functionality in GNU Fortran to identify memory leaks. Accepted values are:

y

Activates AddressSanitizer/LeakSanitizer

n

Deactivates AddressSanitizer/LeakSanitizer (**Default option**).

If you plan to use the **make -j install** option (recommended) to copy your executable to your run directory, you must reconfigure CMake with the **RUNDIR=/path/to/run/dir** option. Multiple run directories can be specified by a semicolon separated list. A warning is issued if one of these directories does not look like a run directory. These paths can be relative paths or absolute paths. Relative paths are interpreted as relative to your build directory. For example:

```
$ cmake . -DRUNDIR=/path/to/run/dir
```

For example if you wanted to build GEOS-Chem with all debugging flags on, you would type:

```
$ cmake . -DCMAKE_BUILD_TYPE=Debug
```

or if you wanted to turn off OpenMP parallelization (so that GEOS-Chem executes only on one computational core), you would type:

```
$ cmake . -DOMP=n
```

etc.

8.1.5 Understand CMake output

As you can see from the example CMake output listed above, GEOS-Chem Classic contains code from 3 independent repositories:

1. **GCClassic wrapper (aka “the superproject”):**

```
=====
GCClassic X.Y.Z (superproject wrapper)
Current status: X.Y.Z
=====
```

where *X.Y.Z* specifies the GEOS-Chem Classic “major”, “minor”, and “patch” version numbers.

Note: If you are cloning GEOS-Chem Classic between official releases, you may see `Current status` reported like this:

```
X.Y.Z-alpha.n-C-gabcd1234.dirty or
X.Y.Z.rc.n-C-gabcd1234.dirty
```

We will explain these formats below.

2. **HEMCO (Harmonized Emissions Component) submodule:**

```
=====
HEMCO A.B.C
Current status: A.B.C
=====
```

where A.B.C specifies the HEMCO “major”, “minor”, and “patch” version numbers. The HEMCO version number differs from GEOS-Chem because it is kept in a separate repository, and is considered a separate package.

3. GEOS-Chem submodule:

```
=====
GEOS-Chem X.Y.Z (science codebase)
Current status: X.Y.Z
=====
```

The GEOS-Chem science codebase and GEOS-Chem Classic wrapper will always share the same version number.

During the build configuration stage, CMake will display the **version number** (e.g. X.Y.Z) as well as the **current status of the Git repository** (e.g. TAG-C-gabcd1234.dirty) for GCCClassic, GEOS-Chem, and HEMCO.

Let’s take the Git repository status of GCCClassic as our example. The status string uses the same format as the **git describe --tags** command, namely:

```
TAG-C-gabcd1234.dirty
```

where

TAG

Indicates the most recent tag in the [GCCClassic superproject repository](#).

Tags may use the following notations:

- X.Y.Z: Denotes an official release
- X.Y.Z-rc.n: Denotes a release candidate
- X.Y.Z-alpha.n: Denotes an internal “alpha” benchmark

where n is the number of the release candidate or alpha benchmark (starting from 0).

C

Indicates the number of commits that were made on top of the commit that is referred to by [TAG](#).

g

Indicates that the version control system is Git.

abcd1234

Indicates the Git commit hash. This is an alphanumeric string that denotes the commit at the HEAD of the GCCClassic repository.

.dirty

If present, indicates that there are uncommitted updates atop the [abcd1234](#) commit in the GCCClassic repository.

Under each header are printed the various *options that have been selected*.

8.2 Compile with Make

Now that CMake has created the Makefiles that are needed to compile GEOS-Chem, you may proceed as follows:

8.2.1 Build the GEOS-Chem Classic executable

Use the **make** command to build the GEOS-Chem executable. Type:

```
$ make -j
```

You will see output similar to this:

```
Scanning dependencies of target HeadersHco
Scanning dependencies of target Isorropia
Scanning dependencies of target KPP_FirstPass
[ 1%] Building Fortran object src/HEMCO/src/Shared/Headers/CMakeFiles/HeadersHco.dir/
↳hco_inquireMod.F90.o
[ 1%] Building Fortran object src/HEMCO/src/Shared/Headers/CMakeFiles/HeadersHco.dir/
↳hco_precision_mod.F90.o
[ 1%] Building Fortran object src/HEMCO/src/Shared/Headers/CMakeFiles/HeadersHco.dir/
↳hco_charpak_mod.F90.o
[ 3%] Building Fortran object src/GEOS-Chem/KPP/fullchem/CMakeFiles/KPP_FirstPass.
↳dir/gckpp_Monitor.F90.o
[ 3%] Building Fortran object src/GEOS-Chem/KPP/fullchem/CMakeFiles/KPP_FirstPass.
↳dir/gckpp_Precision.F90.o
[ 3%] Building Fortran object src/GEOS-Chem/KPP/fullchem/CMakeFiles/KPP_FirstPass.
↳dir/gckpp_Parameters.F90.o
[ 3%] Linking Fortran static library libKPP_FirstPass.a
[ 3%] Built target KPP_FirstPass
Scanning dependencies of target Headers
[ 3%] Building Fortran object src/GEOS-Chem/ISORROPIA/CMakeFiles/Isorropia.dir/
↳isorropiaII_main_mod.F.o
[ 3%] Building Fortran object src/GEOS-Chem/Headers/CMakeFiles/Headers.dir/charpak_
↳mod.F90.o
[ 3%] Building Fortran object src/GEOS-Chem/Headers/CMakeFiles/Headers.dir/
↳dictionary_m.F90.o
[ 3%] Building Fortran object src/GEOS-Chem/Headers/CMakeFiles/Headers.dir/CMN_SIZE_
↳mod.F90.o
[ 3%] Building Fortran object src/GEOS-Chem/Headers/CMakeFiles/Headers.dir/qfyaml_
↳mod.F90.o
[ 4%] Building Fortran object src/GEOS-Chem/Headers/CMakeFiles/Headers.dir/CMN_O3_
↳mod.F90.o
[ 6%] Building Fortran object src/GEOS-Chem/Headers/CMakeFiles/Headers.dir/
↳inquireMod.F90.o
... etc ...

[ 93%] Building Fortran object src/GEOS-Chem/GeosCore/CMakeFiles/GeosCore.dir/sulfate_
↳mod.F90.o
[ 93%] Building Fortran object src/GEOS-Chem/GeosCore/CMakeFiles/GeosCore.dir/
↳fullchem_mod.F90.o
[ 93%] Building Fortran object src/GEOS-Chem/GeosCore/CMakeFiles/GeosCore.dir/mixing_
↳mod.F90.o
[ 93%] Building Fortran object src/GEOS-Chem/GeosCore/CMakeFiles/GeosCore.dir/carbon_
↳mod.F90.o
[ 95%] Building Fortran object src/GEOS-Chem/GeosCore/CMakeFiles/GeosCore.dir/
↳chemistry_mod.F90.o
```

(continues on next page)

(continued from previous page)

```
[ 95%] Building Fortran object src/GEOS-Chem/GeosCore/CMakeFiles/GeosCore.dir/gc_
↪environment_mod.F90.o
[ 96%] Building Fortran object src/GEOS-Chem/GeosCore/CMakeFiles/GeosCore.dir/
↪emissions_mod.F90.o
[ 96%] Building Fortran object src/GEOS-Chem/GeosCore/CMakeFiles/GeosCore.dir/cleanup.
↪F90.o
[ 98%] Linking Fortran static library libGeosCore.a
[ 98%] Built target GeosCore
Scanning dependencies of target gcclassic
[ 98%] Building Fortran object src/CMakeFiles/gcclassic.dir/GEOS-Chem/Interfaces/
↪GCClassic/main.F90.o
[100%] Linking Fortran executable ../bin/gcclassic
[100%] Built target gcclassic
```

Tip: The `-j` argument tells **make** that it can execute as many jobs as it wants simultaneously. For example, if you have 8 cores, then the build process may attempt to compile 8 files at a time.

If you want to restrict the number of simultaneous jobs (e.g. you are compiling on a machine with limited memory), you can use e.g. **make -j4**, which should only try to compile 4 files at a time.

8.2.2 Install the executable in your run directory

Now that the `gclassic` executable is built, install it to your run directory with **make install**. For this to work properly, you must tell CMake where to find your run directory by configuring CMake with `-DRUNDIR=/path/to/run/directory` *as described above*. Type:

```
$ make install
```

and you will see output similar to this:

```
[ 1%] Built target HeadersHco
[ 3%] Built target KPP_FirstPass
[ 3%] Built target Isorropia
[ 4%] Built target JulDayHco
[13%] Built target Headers
[18%] Built target NcdfUtilHco
[19%] Built target JulDay
[19%] Built target GeosUtilHco
[25%] Built target NcdfUtil
[40%] Built target HCO
[46%] Built target GeosUtil
[56%] Built target HCOX
[59%] Built target Transport
[62%] Built target History
[63%] Built target ObsPack
[71%] Built target KPP
[71%] Built target HCOI_Shared
[98%] Built target GeosCore
[100%] Built target gclassic
Install the project...
-- Install configuration: "Release"
-- Up-to-date: /home/ubuntu/gc_merra2_fullchem/build_info/CMakeCache.txt
```

(continues on next page)

(continued from previous page)

```
-- Up-to-date: /home/ubuntu/gc_merra2_fullchem/build_info/summarize_build
-- Up-to-date: /home/ubuntu/gc_merra2_fullchem/gcclassic
```

Let's now navigate back to the run directory and get a directory listing:

```
$ cd ..
$ ls
CodeDir@                                cleanRunDir.sh*
GEOSChem.Restart.20190701_0000z.nc4    download_data.py*
HEMCO_Config.rc                       download_data.yml
HEMCO_Config.rc.gmao_metfields        gcclassic*
HEMCO_Diagn.rc                       geoschem_config.yml
HISTORY.rc                           getRunInfo*
OutputDir/                           metrics.py*
README                               runScriptSamples@
archiveRun.sh*                       rundirConfig/
build/                               species_database.yml
build_info/
```

You should now see the **gcclassic** executable and a `build_info` directory there. GEOS-Chem has now been configured, compiled, and installed in your run directory.

Please see the [Run directory files and folders](#) section for more information about the contents of the run directory.

You are now ready to run a GEOS-Chem simulation!

8.2.3 Remove compiler-generated files when no longer needed

In older versions of GEOS-Chem, you could use a GNU Make command such as **make clean** or **make realclean** to remove all object (`.o`), library (`.a`), module (`.mod`) files, as well as the previously-built executable file from the GEOS-Chem source code folder.

All of the files created by Cmake during the configuration and compilation stages are placed in the `build/` folder in your run directory (or in the location that you have specified with the `-DRUNDIR=/path/to/run/dir` option.). Therefore, if you wish to build the **GEOS-Chem Classic** executable from scratch, all you have to do is to remove all of the files from the build folder. It's as simple as that!

You can also create a new build folder with this command:

```
$ mv build was.build
$ mkdir build
```

and then later on, you can remove the old build folder:

```
$ rm -rf was.build
```

This avoids the temptation to use **rm -rf ***, which can potentially wipe out all of your files if used incorrectly.

8.3 Get a summary of compilation options

The compilation process will create a folder in your run directory named `build_info`. Navigate into this folder and get a directory listing:

```
$ cd build_info
$ ls -CF
CMakeCache.txt  summarize_build*
```

`CMakeCache.txt` contains the **CMake cache**, which is a complete listing of all compilation settings. `summarize_build` is a script that will print the most important of these CMake cache settings.

If you run `summarize_build`:

```
$ ./summarize_build
```

You will get output similar to this:

```
## Compiler Info
# Family:   GNU
# Version:  11.2.0
# Which:    /path/to/gfortran

## Compiler Options (global)
-DCMAKE_Fortran_FLAGS=""
-DCMAKE_Fortran_FLAGS_DEBUG="-g"

## Compiler Options (GEOS-Chem)
-DGEOSChem_Fortran_FLAGS_GNU="-g;-cpp;-w;-std=legacy;-fautomatic;-fno-align-commons;-
↪fconvert=big-endian;-fno-range-check;-mmodel=medium;-fbacktrace;-g;-DLINUX_
↪GFORTRAN;-ffree-line-length-none"
-DGEOSChem_Fortran_FLAGS_DEBUG_GNU="-O0;-Wall;-Wextra;-Wconversion;-Warray-
↪temporaries;-fcheck=array-temps;-ffpe-trap=invalid,zero,overflow;-finit-real=snan;-
↪fcheck=bounds;-fcheck=pointer"

## Compiler Options (HEMCO)
-DHEMCO_Fortran_FLAGS_GNU="-cpp;-w;-std=legacy;-fautomatic;-fno-align-commons;-
↪fconvert=big-endian;-fno-range-check;-mmodel=medium;-fbacktrace;-g;-DLINUX_
↪GFORTRAN;-ffree-line-length-none"
-DHEMCO_Fortran_FLAGS_DEBUG_GNU="-g;-gdwarf-2;-gstrict-dwarf;-O0;-Wall;-Wextra;-
↪Wconversion;-Warray-temporaries;-fcheck=array-temps;-ffpe-trap=invalid,zero,
↪overflow;-finit-real=snan;-fcheck=bounds;-fcheck=pointer;-fcheck=no-recursion"

## GEOS-Chem Components Settings
-DTOMAS="OFF"
-DTOMAS_BINS="NA"
-DAPM="OFF"
-DRRTMG="OFF"
-DGTMM="OFF"
-DHCOSA="OFF"
-DLUO_WETDEP="OFF"
-DFASTJX="OFF"
```

Here you can see the compiler flags that were used as well as the options that were selected.

CONFIGURE YOUR SIMULATION

Note: We recommend that you configure your simulation before downloading data files. You can use the configuration settings with a *dry-run simulation* to download only the data that you will need.

You will need to edit various **configuration files** in order to specify options for your GEOS-Chem Classic simulation. These are described below.

9.1 Commonly-updated configuration files

When starting a new GEOS-Chem Classic simulation, you will usually edit most (if not all) of these configuration files:

9.1.1 geoschem_config.yml

Starting with GEOS-Chem 14.0.0, the `input.geos` configuration file (plain text) has been replaced with by the `geoschem_config.yml` file. This file is in **YAML** format, which is a text-based markup syntax used for representing dictionary-like data structures.

Note: The `geoschem_config.yml` file contains several sections. Only the sections relevant to a given type of simulation are present. For example, *fullchem* simulation options (such as aerosol settings and photolysis settings) are omitted from the `geoschem_config.yml` file for the *CH4* simulation.

Simulation settings

```
#=====
# Simulation settings
#=====
simulation:
  name: fullchem
  start_date: [20190701, 000000]
  end_date: [20190801, 000000]
  root_data_dir: /path/to/ExtData
  met_field: MERRA2
  species_database_file: ./species_database.yml
  species_metadata_output_file: OutputDir/geoschem_species_metadata.yml
  verbose:
```

(continues on next page)

(continued from previous page)

```

activate: false
on_cores: root      # Allowed values: root all
use_gcclassic_timers: false

```

The simulation section contains general simulation options:

name

Specifies the type of GEOS-Chem simulation. Accepted values are

fullchem

Full-chemistry simulation.

aerosol

Aerosol-only simulation.

carbon

Coupled carbon gases simulation (CH₄-CO-CO₂-OCS), implemented as a KPP mechanism (cf Bukosa *et al.* [[Bukosa et al., 2023]]).

You must *configure your build* with `-DMECH=carbon` in order to use this simulation.

CH4

Methane simulation.

This simulation will eventually be superseded by the *carbon* simulation.

CO2

Carbon dioxide simulation.

This simulation will eventually be superseded by the *carbon* simulation.

Hg

Mercury simulation.

You must *configure your build* with `-DMECH=Hg` in order to use this simulation.

POPs

Persistent organic pollutants (aka POPs) simulation.

Attention: The POPs simulation is currently stale. We look to members of the GEOS-Chem user community take the lead on updating this simulation.

tagCH4

Methane simulation with species tagged by geographic region or other criteria.

This simulation will eventually be superseded by the *carbon* simulation.

tagCO

Carbon dioxide simulation, with species tagged by geographic region and other criteria.

This simulation will eventually be superseded by the *carbon* simulation.

tagO3

Ozone simulation (using specified production and loss rates), with species tagged by geographical region.

TransportTracers

Transport Tracers simulation, with both radionuclide and passive_species. Useful for evaluating model transport.

metals

Trace metals simulation

start_date

Specifies the starting date and time of the simulation in list notation [YYYYMMDD, hhmmss].

end_date

Specifies the ending date and time of the simulation in list notation [YYYYMMDD, hhmmss].

root_data_dir

Path to the root data directory. All of the data that GEOS-Chem Classic reads must be located in subfolders of this directory.

met_field

Name of the meteorology product that will be used to drive GEOS-Chem Classic. Accepted values are:

MERRA2

The [MERRA-2](#) meteorology product from NASA/GMAO. MERRA-2 is a stable reanalysis product, and extends from approximately 1980 to present. **(Recommended option)**

GEOS-FP

The [GEOS-FP](#) meteorology product from NASA/GMAO. GEOS-FP is an operational data product and, unlike MERRA-2, periodically receives science updates.

GCAP2

The GCAP-2 meteorology product, archived from the GISS-2 GCM. GCAP-2 has hundreds of years of data available, making it useful for simulations of historical climate.

species_database_file

Path to the [GEOS-Chem Species Database](#) file. This is stored in the run directory file `./species_database.yml`. You should not have to edit this setting.

species_metadata_output_file

Path to the `geoschem-species-metadata.yml` file. This file contains echoback of information from [species_database.yml](#), but only for species that are defined in this simulation (instead of all possible species). This facilitates interfacing GEOS-Chem with external models such as CESM.

verbose:

Menu controlling verbose printout. Starting with GEOS-Chem 14.2.0 and HEMCO 3.7.0, most informational printouts are now deactivated by default. You may choose to activate them (e.g. for debugging and/or testing) with the options below:

activate

Activates (`true`) or deactivates (`false`) printing extra informational printout to the screen and/or log file.

on_cores:

Specify on which computational cores informational printout should be done.

root

Print extra informational output only on the root core. Use this setting for GEOS-Chem Classic.

all

Print extra informational output on all cores. Consider using this when using GEOS-Chem as GCHP, or in MPI-based external models (NASA GEOS, CESM, etc.).

use_gcclassic_timers

Activates (`true`) or deactivates (`false`) the GEOS-Chem Classic timers. If activated, information about how long each component of GEOS-Chem took to execute will be printed to the screen and/or [GEOS-Chem log file](#). The same information will also be written in JSON format to a file named [gcclassic_timers.json](#).

You can set this option to `false` unless you are running benchmark or timing simulations.

Grid settings

```
#####  
# Grid settings  
#####  
grid:  
  resolution: 4.0x5.0  
  number_of_levels: 72  
  longitude:  
    range: [-180.0, 180.0]  
    center_at_180: true  
  latitude:  
    range: [-90.0, 90.0]  
    half_size_polar_boxes: true  
  nested_grid_simulation:  
    activate: true  
    buffer_zone_NSEW: [0, 0, 0, 0]
```

The `grid` section contains settings that define the grid used by GEOS-Chem Classic:

resolution

Specifies the horizontal resolution of the grid. Accepted values are:

4.0x5.0

The global $4^{\circ} \times 5^{\circ}$ GEOS-Chem Classic grid.

2.0x2.5

The global $2.0^{\circ} \times 2.5^{\circ}$ GEOS-Chem Classic grid.

0.5x0.625

The global $0.5^{\circ} \times 0.625^{\circ}$ GEOS-Chem Classic grid ([MERRA2](#) only). Can be used for global or nested simulations.

0.5x0.625

The global $0.25^{\circ} \times 0.3125^{\circ}$ GEOS-Chem Classic grid ([GEOS-FP](#) and [MERRA2](#)). Can be used for global or nested simulations.

number_of_levels

Number of vertical levels to use in the simulation. Accepted values are:

72

Use 72 vertical levels. This is the native vertical resolution of [MERRA2](#) and [GEOS-FP](#).

47

Use 47 vertical levels (for [MERRA2](#) and [GEOS-FP](#)).

40

Use 40 vertical levels (for [GCAP2](#)).

longitude

Settings that define the longitude dimension of the grid. There are two sub-options:

range

The minimum and maximum longitude values (grid box centers), specified in list format.

center_at_180

If `true`, then westernmost grid boxes are centered at -180° longitude (the International Date Line). This is true for both [MERRA2](#) and [GEOS-FP](#).

If `false`, then the westernmost grid boxes have their westernmost edges at -180° longitude. This is true for the [GCAP2](#) grid.

latitude

Settings to define the latitude dimension of the grid. There are two sub-options:

range

The minimum and maximum latitude values (grid box centers), specified in list format.

use_halfpolar_boxes

If `true`, then the northernmost and southernmost grid boxes will be $\frac{1}{2}$ the extent of other grid boxes. This is true for both *MERRA2* and *GEOS-FP*.

If `false`, then all grid boxes will have the same extent in latitude. This is true for the *GCAP2* grid.

nested_grid_simulation

Settings for nested-grid simulations. There are two sub-options:

activate

If `true`, this indicates that the simulation will use a sub-window of the horizontal grid.

If `false`, this indicates that the simulation will use the entire global grid extent.

buffer_zone_NSEW

Specifies the nested grid latitude offsets (# of grid boxes) in list format [N-offset, S-offset, E-offset, W-offset]. These offsets are used to define an inner window region in which transport is actually done (aka the “transport window”). This “transport window” is always smaller than the actual size of the nested grid region in order to properly account for the boundary conditions.

- For global simulations, use: [0, 0, 0, 0].
- For nested-grid simulations, we recommend using: [3, 3, 3, 3].

Timesteps settings

```
#####
# Timesteps settings
#####
timesteps:
  transport_timestep_in_s: 600
  chemistry_timestep_in_s: 1200
  radiation_timestep_in_s: 10800
```

The `timesteps` section specifies the frequency at which various GEOS-Chem operations occur:

transport_timestep_in_s

Specifies the “heartbeat” timestep of GEOS-Chem.. This is the frequency at which transport, cloud convection, PBL mixing, and wet deposition will be done.

- Recommended value for global simulations: 600
- Recommended value for nested simulations: 300 or smaller

chemistry_timestep_in_s

Specifies the frequency at which chemistry and emissions will be done.

- Recommended value for global simulations 1200
- Recommended value for nested simulations 600 or smaller

radiation_timestep_in_s

Specifies the frequency at which the *RRTMG* radiative transfer model will be called (valid for *fullchem* simulations only).

Operations settings

This section of `geoschem_config.yml` is included for all simulations. However, some of the options listed below will be omitted for simulations that do not require them.

There are several sub-sections under `operations`:

Chemistry

```
#####  
# Settings for GEOS-Chem operations  
#####  
operations:  
  
chemistry:  
  activate: true  
  linear_chemistry_aloft:  
    activate: true  
    use_linoz_for_O3: true  
  active_strat_H2O:  
    activate: true  
    use_static_bnd_cond: true  
  gamma_HO2: 0.2  
  autoreduce_solver:  
    activate: false  
    use_target_threshold:  
      activate: true  
      oh_tuning_factor: 0.00005  
      no2_tuning_factor: 0.0001  
    use_absolute_threshold:  
      scale_by_pressure: true  
      absolute_threshold: 100.0  
  keep_halogens_active: false  
  append_in_internal_timestep: false  
  
# ... following sub-sections omitted ...
```

The `operations:chemistry` section contains settings for chemistry:

activate

Activates (`true`) or deactivates (`false`) chemistry in GEOS-Chem.

linear_chemistry_aloft

Determines how linearized chemistry will be applied in the stratosphere and/or mesosphere. (Only valid for *fullchem* simulations).

There are two sub-options:

activate

Activates (`true`) or deactivates (`false`) linearized stratospheric chemistry in the stratosphere and/or mesosphere.

use_linoz_for_O3

If `true`, *Linoz stratospheric ozone chemistry* will be used.

If `false`, *Synoz* (i.e. a synthetic flux of ozone across the tropopause) will be used instead of *Linoz*.

active_strat_H2O

Determines if water vapor as modeled by GEOS-Chem will be allowed to influence humidity fields. (Only valid for *fullchem* simulations)

There are two sub-options:

activate

Allows (*true*) or disallows (*false*) the H₂O species in GEOS-Chem to influence specific humidity and relative humidity.

use_static_bnd_cond

Allows (*true*) or disallows (*false*) a static boundary condition.

TODO Clarify this

gamma_HO2

Specifies γ , the uptake coefficient for HO₂ heterogeneous chemistry.

Recommended value: 0.2.

autoreduce_solver

Menu for controlling the adaptive mechanism auto-reduction feature, which is available in KPP 3.0.0. and later versions. See Lin *et al.* [[Lin et al., 2023]] for details.

activate

If *true*, the mechanism will be integrated using the Rosenbrock method with the adaptive auto-reduction feature.

If *false*, the mechanism will be integrated using the traditional Rosenbrock method.

Default value: *false*.

use_target_threshold

Contains options for defining ∂ (the partitioning threshold between “fast” and “slow” species”) by considering the production and loss of key species (OH for daytime, NO₂ for nighttime).

activate

Activates (*true*) or deactivates (*false*) using OH and NO₂ to determine ∂ .

Default value: *true*.

oh_tuning_factor

Specifies α_{OH} , which is used to compute ∂ .

no2 tuning factor

Specifies α_{NO_2} , which is used to compute ∂ .

use_pressure_threshold

Contains options for setting an absolute threshold ∂ that may be weighted by pressure.

scale_by_pressure

Activates (*true*) or deactivates (*false*) using a pressure-dependent method to determine ∂ .

absolute_threshold

The absolute partitioning threshold ∂ .

If *scale_by_pressure* is *true*, and *use_target_threshold:activate* is *false*, the value for ∂ specified here will be scaled by the ratio P/P_{sfc} , where P is the grid box pressure and P_{sfc} is the surface pressure for the column.

keep_halogens_active

If *true*, then all halogen species will be considered “fast”. This may be necessary in order to obtain realistic results for ozone and other important species.

If false, then halogen species will be determined as “slow” or “fast” depending on the partitioning threshold θ .

Default value: true

append_in_internal_timestep

If true, any “slow” species that later become “fast” will be appended to the list of “fast” species.

If false, any “slow” species that later become “fast” will NOT be appended to the list of “fast” species.

Default value: false

Convection

```
#####
# Settings for GEOS-Chem operations
#####
operations:

# .. preceding sub-sections omitted ...

convection:
  activate: true

# ... following sub-sections omitted ...
```

The **operations:convection** section contains settings for cloud convection:

activate

Activates (true) or deactivates (false) cloud convection in GEOS-Chem.

Dry deposition

```
#####
# Settings for GEOS-Chem operations
#####
operations:

# .. preceding sub-sections omitted ...

dry_deposition:
  activate: true
  CO2_effect:
    activate: false
    CO2_level: 600.0
    reference_CO2_level: 380.0
  diag_alt_above_sfc_in_m: 10

# ... following sub-sections omitted ...
```

The **operations:dry_deposition** section contains settings that for dry deposition:

activate

Activates (true) or deactivates (false) dry deposition.

CO2_effect

This sub-section contains options for applying the [simple parameterization for the CO2 effect on stomatal resistance](#).

activate

Activates (`true`) or deactivates (`false`) the CO2 effect on stomatal resistance in dry deposition.

Default value: `false`.

CO2_level

Specifies the CO2 level (in ppb).

reference_CO2_level

Specifies the reference CO2 level (in ppb).

diag_alt_above_sfc_in_m:

Specifies the altitude above the surface (in m) to used with the [ConcAboveSfc](#) diagnostic collection.

PBL mixing

```
#####
# Settings for GEOS-Chem operations
#####
operations:

# .. preceding sub-sections omitted ...

pbl_mixing:
  activate: true
  use_non_local_pbl: true

# ... following sub-sections omitted ...
```

The `operations:pbl_mixing` section contains settings that for [planetary boundary layer \(PBL\) mixing](#):

activate

Activates (`true`) or deactivates (`false`) planetary boundary layer mixing in GEOS-Chem Classic.

use_non_local_pbl

If `true`, then the [non-local PBL mixing scheme \(VDIFF\)](#) will be used. (Default option)

If `false`, then the [full PBL mixing scheme \(TURBDAY\)](#) will be used.

Photolysis

```
#####
# Settings for GEOS-Chem operations
#####
operations:

# .. preceding sub-sections omitted ...

photolysis:
  activate: true
  input_directories:
    fastjx_input_dir: /path/to/ExtData/CHEM_INPUTS/FAST_JX/v2021-10/
```

(continues on next page)

(continued from previous page)

```

cloudj_input_dir: /path/to/ExtData/CHEM_INPUTS/CLOUD_J/v2023-05/
overhead_O3:
  use_online_O3_from_model: true
  use_column_O3_from_met: true
  use_TOMS_SBUV_O3: false
photolyze_nitrate_aerosol:
  activate: true
  NITs_Jscale_JHNO3: 0.0
  NIT_Jscale_JHNO2: 0.0
  percent_channel_A_HONO: 66.667
  percent_channel_B_NO2: 33.333

# ... following sub-sections omitted ...

```

The `operation:photolysis` section contains settings for photolysis. This section only applies to fullchem, Hg, and aerosol-only simulations.

activate

Activates (`true`) or deactivates (`false`) photolysis.

Attention: You should always keep photolysis turned on in your simulations. Disabling photolysis should only be done when debugging.

input_directories

Specifies the location of directories containing photolysis configuration files.

fastjx_input_dir

Specifies the path to the legacy FAST_JX configuration files containing information about species cross sections and quantum yields. Note that FAST-JX is off by default and Cloud-J is used instead. You can use legacy FAST-JX instead of Cloud-J by configuring with `-DFASTJX=y` during build.

cloudj_input_dir

Specifies the path to the Cloud-J configuration files containing information about species cross sections and quantum yields.

overhead_O3

This section contains settings that control which overhead ozone sources are used for photolysis

use_online_O3_from_model

Activates (`true`) or deactivates (`false`) using online O3 from GEOS-Chem in the extinction calculations for photolysis.

Recommended value: `true`

use_column_O3_from_met

Activates (`true`) or deactivates (`false`) using ozone columns (e.g. TO3) from the meteorology fields.

Recommended value: `true`.

use_TOMS_SBUV_O3

Activates (`true`) or deactivates (`false`) using ozone columns from the TOMS-SBUV archive will be used.

Recommended value: `false`.

photolyze_nitrate_aerosol:

This section contains settings that control options for nitrate aerosol photolysis.

activate

Activates (`true`) or deactivates (`false`) nitrate aerosol photolysis.

Recommended value: `true`.

NITs_Jscale_JHNO3

Scale factor (percent) for JNO3 that photolyzes NITs aerosol.

NIT_Jscale_JHNO2

Scale factor (percent) for JHNO2 that photolyzes NIT aerosol.

percent_channel_A_HONO

Fraction of JNITs/JNIT in channel A (HNO2) for NITs photolysis.

percent_channel_B_NO2

Fraction of JNITs/JNIT in channel B (NO2) for NITs photolysis.

RRTMG radiative transfer model

```
#####
# Settings for GEOS-Chem operations
#####
operations:

# .. preceding sub-sections omitted ...

rrtmg_rad_transfer_model:
  activate: false
  aod_wavelengths_in_nm:
    - 550
  longwave_fluxes: false
  shortwave_fluxes: false
  clear_sky_flux: false
  all_sky_flux: false

# .. following sub-sections omitted ...
```

The `operations:rrtmg_rad_transfer_model` section contains settings for the **RRTMG radiative transfer model**:

This section only applies to *fullchem* simulations.

activate

Activates (`true`) or deactivates (`false`) the RRTMG radiative transfer model.

Default value: `false`.

aod_wavelengths_in_nm

Specify wavelength(s) for the aerosol optical properties in nm (in **YAML sequence format**) Up to three wavelengths can be selected. The specified wavelengths are used for the photolysis mechanism (either legacy FAST-JX or Cloud-J) regardless of whether the RRTMG radiative transfer model is used.

longwave_fluxes

Activates (`true`) or deactivates (`false`) RRTMG longwave flux calculations.

Default value: `false`.

shortwave_fluxes

Activates (`true`) or deactivates (`false`) RRTMG shortwave calculations.

Default value: `false`.

clear_sky_flux

Activates (`true`) or deactivates (`false`) RRTMG clear-sky flux calculations.

Default value: `false`.

all_sky_flux

Activates (`true`) or deactivates (`false`) RRTMG all-sky flux calculations.

Default value: `false`.

Transport

```
#####
# Settings for GEOS-Chem operations
#####
operations:

# .. preceding sub-sections omitted ...

transport:
  gcclassic_tpcore:                # GEOS-Chem Classic only
    activate: true                 # GEOS-Chem Classic only
    fill_negative_values: true     # GEOS-Chem Classic only
    iord_jord_kord: [3, 3, 7]     # GEOS-Chem Classic only
  transported_species:
    - ACET
    - ACTA
    - AERI
    # ... etc more transported species ...

# .. following sub-sections omitted ...
```

The `operations:transport` section contains settings for [species transport](#):

gclassic_tpcore

Contains options that control species transport in GEOS-Chem Classic with the [TPCORE](#) advection scheme:

activate

Activates (`true`) or deactivates (`false`) species transport in GEOS-Chem Classic.

Default value: `true`.

fill_negative_values

If `true`, negative species concentrations will be replaced with zeros.

If `false`, no change will be made to species concentrations.

Default value: `true`.

iord_jord_kord

Specifies advection options (in list format) for [TPCORE](#) in the longitude, latitude, and vertical dimensions. The options are listed below:

1. 1st order upstream scheme (use for debugging only)
2. 2nd order van Leer (full monotonicity constraint)
3. Monotonic PPM
4. Semi-monotonic PPM (same as 3, but overshoots are allowed)
5. Positive-definite PPM

6. Un-constrained PPM (use when fields & winds are very smooth) this option only when the fields and winds are very smooth.
7. Huynh/Van Leer/Lin full monotonicity constraint (KORD only)

Default (and recommended) value: [3, 3, 7]

transported_species

A list of species names (in [YAML sequence format](#)) that will be transported by the TPCORE advection scheme.

Wet deposition

```
#####
# Settings for GEOS-Chem operations
#####
operations:

# .. preceding sub-sections omitted ...

wet_deposition:
  activate: true
```

The `operations:wet_deposition` section contains settings for [wet deposition](#).

activate

Activates (`true`) or deactivates (`false`) wet deposition in GEOS-Chem Classic.

Aerosols settings

This section of `geoschem_config.yml` is included for [fullchem](#) and [aerosol](#) simulations.

There are several sub-sections under `aerosols`:

Carbon aerosols

```
#####
# Settings for GEOS-Chem aerosols
#####
aerosols:

  carbon:
    activate: true
    brown_carbon: false
    enhance_black_carbon_absorption:
      activate: true
      hydrophilic: 1.5
      hydrophobic: 1.0

# .. following sub-sections omitted ...
```

The `aerosols:carbon` section contains settings for [carbon aerosols](#):

activate

Activates (`true`) or deactivates (`false`) carbon aerosols in GEOS-Chem.

Default value: `true`.

brown_carbon

Activates (`true`) or deactivates (`false`) brown carbon aerosols in GEOS-Chem.

Default value: `false`.

enhance_black_carbon_absorption

Options for enhancing the absorption of black carbon aerosols due to external coating.

activate

Activates (`true`) or deactivates (`false`) black carbon absorption enhancement.

Default value: `true`.

hydrophilic

Absorption enhancement factor for hydrophilic black carbon aerosol (species name **BCPI**).

Default value: `1.5`

hydrophobic

Absorption enhancement factor for hydrophilic black carbon aerosol (species name **BCPO**).

Default value: `1.0`

Complex SOA

The `aerosols:complex_SOA` section contains settings for the complex SOA scheme used in GEOS-Chem.

```
#=====
# Settings for GEOS-Chem aerosols
#=====
aerosols:

# ... preceding sub-sections omitted ...

complex_SOA:
  activate: true
  semivolatile_POA: false

# ... following sub-sections omitted ...
```

activate

Activates (`true`) or deactivates (`false`) the complex SOA scheme.

Default value:

- `true` for the *fullchem* benchmark simulation
- `false` for all other *fullchem* simulations

semivolatile_POA

Activates (`true`) or deactivates (`false`) the semi-volatile primary organic aerosol (POA) option.

Default value: `false`

Mineral dust aerosols

The `aerosols:dust` section contains settings for mineral dust aerosols.

```
#####
# Settings for GEOS-Chem aerosols
#####
aerosols:

# ... preceding sub-sections omitted ...

dust:
  activate: true
  acid_uptake_on_dust: false

# ... following sub-sections omitted ...
```

activate

Activates (`true`) or deactivates (`false`) mineral dust aerosols in GEOS-Chem.

Default value: `true`

acid_uptake_on_dust

Activates (`true`) or deactivates (`false`) the acid uptake on dust option, which includes 12 additional species.

Default value: `false`

Sea salt aerosols

The `aerosols:sea_salt` section contains settings for sea salt aerosols:

```
#####
# Settings for GEOS-Chem aerosols
#####
aerosols:

# ... preceding sub-sections omitted ...

sea_salt:
  activate: true
  SALA_radius_bin_in_um: [0.01, 0.5]
  SALC_radius_bin_in_um: [0.5, 8.0]
  marine_organic_aerosols: false

# ... following sub-sections omitted ...
```

activate

Activates (`true`) or deactivates (`false`) sea salt aerosols.

Default value: `true`

SALA_radius_bin_in_um

Specifies the upper and lower boundaries (in nm) for accumulation-mode sea salt aerosol (aka **SALA**).

Default value: 0.01 nm – 0.5 nm

SALC_radius_bin_in_um

Specifies the upper and lower boundaries (in nm) for coarse-mode sea salt aerosol (aka **SALC**).

Default value: 0.5 nm – 8.0 nm

marine_organic_aerosols

Activates (true) or deactivates (false) emission of marine primary organic aerosols. This option includes two extra species (MOPO and MOPI).

Default value: false

Stratospheric aerosols

The aerosols:sulfate section contains settings for stratospheric aerosols.

```
#####
# Settings for GEOS-Chem aerosols
#####
aerosols:

# ... preceding sub-sections omitted ...

stratosphere:
  settle_strat_aerosol: true
  polar_strat_clouds:
    activate: true
    het_chem: true
  allow_homogeneous_NAT: false
  NAT_supercooling_req_in_K: 3.0
  supersat_factor_req_for_ice_nucl: 1.2
  calc_strat_aod: true

# ... following sub-sections omitted ...
```

settle_strat_aerosol

Activates (true) or deactivates (false) gravitational settling of stratospheric solid particulate aerosols (SPA, trapezoidal scheme) and stratospheric liquid aerosols (SLA, corrected Stokes' Law).

Default value: true

polar_strat_clouds

Contains settings for how aerosols are handled in polar stratospheric clouds (PSC):

activate

Activates (true) or deactivates (false) formation of polar stratospheric clouds.

Default value: true

het_chem

Activates (true) or deactivates (false) heterogeneous chemistry within polar stratospheric clouds.

Default value: true

allow_homogeneous_NAT

Activates (true) or deactivates (false) heterogeneous formation of NAT from freezing of HNO₃.

Default value: false

NAT_supercooling_req_in_K

Specifies the cooling (in K) required for homogeneous NAT nucleation.

Default value: 3.0

supersat_factor_req_for_ice_nucl

Specifies the supersaturation factor required for ice nucleation.

Recommended values: 1.2 for coarse grids; 1.5 for fine grids.

calc_strat_aod

Includes (true) or excludes (false) online stratospheric aerosols in extinction calculations for photolysis.

Default value: true

Sulfate aerosols

The aerosols:sulfate section contains settings for sulfate aerosols:

```
#=====
# Settings for GEOS-Chem aerosols
#=====
aerosols:

# ... preceding sub-sections omitted ...

sulfate:
  activate: true
  metal_cat_SO2_oxidation: true
```

activate

Activates (true) or deactivates (false) sulfate aerosols.

Default value: true

metal_cat_SO2_oxidation

Activates (true) or deactivates (false) the metal catalyzed oxidation of SO₂.

Default value: true

Extra diagnostics

The extra_diagnostics section contains settings for GEOS-Chem Classic diagnostics that are not archived by *History* or *HEMCO*:

Obspack diagnostic

The extra_diagnostics:obspack section contains settings for the Obspack diagnostic:

```
#=====
# Settings for diagnostics (other than HISTORY and HEMCO)
#=====
extra_diagnostics:

  obspack:
    activate: false
    quiet_logfile_output: false
    input_file: ./obspack_co2_1_OCO2MIP_2018-11-28.YYYYMMDD.nc
    output_file: ./OutputDir/GEOSChem.ObsPack.YYYYMMDD_hhmmz.nc4
    output_species:
      - CO
```

(continues on next page)

(continued from previous page)

```
- 'NO'
- O3

# ... following sub-sections omitted ...
```

activate

Activates (`true`) or deactivates (`false`) ObsPack diagnostic output.

Default value: `true`

quiet_logfile_output

Deactivates (`true`) or activates (`false`) printing informational output to `stdout` (i.e. the screen or log file).

Default value: `false`

input_file

Specifies the path to an ObsPack data file (in netCDF format).

output_file

Specifies the path to the ObsPack diagnostic output file. This will be a file that contains data at the same locations as specified in *input_file*.

output_species

A list of GEOS-Chem species (as a YAML sequence) to archive to the output file.

Planeflight diagnostic

The `extra_diagnostics:planeflight` section contains settings for the [GEOS-Chem planeflight diagnostic](#):

```
=====
# Settings for diagnostics (other than HISTORY and HEMCO)
=====
extra_diagnostics:

# ... preceding sub-sections omitted ...

planeflight:
  activate: false
  flight_track_file: Planeflight.dat.YYYYMMDD
  output_file: plane.log.YYYYMMDD

# ... following sub-sections omitted ...
```

activate

Activates (`true`) or deactivates (`false`) the Planeflight diagnostic output.

Default value: `false`

flight_track_file

Specifies the path to a flight track file. This file contains the coordinates of the plane as a function of time, as well as the requested quantities to archive.

output_file

Specifies the path to the Planeflight output file. Requested quantities will be archived from GEOS-Chem along the flight track specified in *flight_track_file*.

Hg simulation options

This section of `geoschem_config.yml` is included for the mercury (Hg) simulation:

Hg sources

The `Hg_simulation_options:sources` section contains settings for various mercury sources.

```
#####
# Settings specific to the Hg simulation
#####
Hg_simulation_options:

  sources:
    use_dynamic_ocean_Hg: false
    use_preindustrial_Hg: false
    use_arctic_river_Hg: true

# ... following sub-sections omitted ...
```

use_dynamic_ocean_Hg

Activates (`true`) or deactivates (`false`) the online slab ocean mercury model.

Default value: `false`

use_preindustrial_Hg

Activates (`true`) or deactivates (`false`) the preindustrial mercury simulation. This will turn off all anthropogenic emissions.

Default value: `false`

use_arctic_river_Hg

Activates (`true`) or deactivates (`false`) the source of mercury from arctic rivers.

Default value: `true`

Hg chemistry

The `Hg_simulation_options:chemistry` section contains settings for mercury chemistry:

```
#####
# Settings specific to the Hg simulation
#####
Hg_simulation_options:

# ... preceding sub-sections omitted ...

chemistry:
  tie_HgIIaq_reduction_to_UVB: true

# ... following sub-sections omitted ...
```

tie_HgIIaq_reduction_to_UVB

Activates (`true`) or deactivates (`false`) linking the reduction of aqueous oxidized mercury to UVB radiation. A lifetime of -1 seconds indicates the species has an infinite lifetime.

Default value: `true`

Options for simulations with carbon gases

These sections of `geoschem_config.yml` are included for simulations with carbon gases (*carbon*, *CH4*, *CO2*, *tagCO*, *tagCH4*).

CH4 observational operators

The `CH4_simulation_options:use_observational_operators` section contains options for using satellite observational operators for CH4:

```
#####
# Settings specific to the CH4 simulation / Integrated Methane Inversion
#####
CH4_simulation_options:

  use_observational_operators:
    AIRS: false
    GOSAT: false
    TCCON: false

# ... following sub-sections omitted ...
```

AIRS

Activates (`true`) or deactivates (`false`) the AIRS observational operator.

Default value: `false`

GOSAT

Activates (`true`) or deactivates (`false`) the GOSAT observational operator.

Default value: `false`

TCCON

Activates (`true`) or deactivates (`false`) the GOSAT observational operator.

Default value: `false`

CH4 analytical inversion options

The `ch4_simulation_options:analytical_inversion` section contains options for analytical inversions with the [Integrated Methane Inversion workflow \(aka IMI\)](#). The IMI will automatically modify several of these options based on the inversion parameters that you specify.

```
#####
# Settings specific to the CH4 simulation / Integrated Methane Inversion
#####
CH4_simulation_options:

# ... preceding sub-sections omitted ...

analytical_inversion:
  perturb_OH_boundary_conditions: false
  CH4_boundary_condition_ppb_increase_NSEW: [0.0, 0.0, 0.0, 0.0]
```

perturb_CH4_boundary_conditions

Activates (`true`) or deactivates (`false`) perturbation of CH4 nested-grid boundary conditions in analytical inversions.

Default value: false

CH4_boundary_condition_ppb_increase_NSEW

Specifies the perturbation amount (in ppbv) to apply to the north, south, east and west CH4 nested-grid boundary conditions. Used in conjunction with the *perturb_CH4_boundary_conditions* option.

Default value: [0.0, 0.0, 0.0, 0.0] (no perturbation)

CO2 Sources

The `CO2_simulation_options:sources` section contains toggles for activating sources of CO_2 :

```
#####
# Settings specific to the CO2 simulation
#####
CO2_simulation_options:

  sources:
    fossil_fuel_emissions: true
    ocean_exchange: true
    balanced_biosphere_exchange: true
    net_terrestrial_exchange: true
    ship_emissions: true
    aviation_emissions: true
    3D_chemical_oxidation_source: true

# ... following sub-sections omitted ...
```

fossil_fuel_emissions

Activates (true) or deactivates (false) using CO_2 fossil fuel emissions as computed by HEMCO.

Default value: true

ocean_exchange

Activates (true) or deactivates (false) CO_2 ocean-air exchange.

Default value: true

balanced_biosphere_exchange

Activates (true) or deactivates (false) CO_2 balanced-biosphere exchange.

Default value: true

net_terrestrial_exchange

Activates (true) or deactivates (false) CO_2 net terrestrial exchange.

Default value: true

ship_emissions

Activates (true) or deactivates (false) CO_2 ship emissions as computed by HEMCO.

Default value: true

aviation_emissions

Activates (true) or deactivates (false) CO_2 aviation emissions as computed by HEMCO.

Default value: true

3D_chemical_oxidation_source

Activates (true) or deactivates (false) CO_2 production by archived chemical oxidation, as read by HEMCO.

Default value: true

CO₂ tagged species

The `CO2_simulation_options:tagged_species` section contains toggles for activating tagged CO_2 species:

Attention: Tagged CO_2 tracers should be customized by each user and the present configuration will not work for resolutions other than $2.0^\circ \times 2.5^\circ$.

```
#####
# Settings specific to the CO2 simulation
#####
CO2_simulation_options:

# ... preceding sub-sections omitted ...

tagged_species:
  save_fossil_fuel_in_background: false
  tag_bio_and_ocean_CO2: false
  tag_land_fossil_fuel_CO2:
  tag_global_ship_CO2: false
  tag_global_aircraft_CO2: false
```

save_fossil_fuel_in_background

Activates (`true`) or deactivates (`false`) saving the CO_2 background.

Default value: `false`

tag_bio_and_ocean_CO2

Activates (`true`) or deactivates (`false`) tagging of biosphere regions (28), ocean regions (11), and the rest of the world (ROW) as specified in `Regions_land.dat` and `Regions_ocean.dat` files.

.. following sub-sections omitted ...

CO chemical sources

The `tagged_CO_simulation_options` section contains settings for the *carbon* simulation and tagged CO simulation.

```
#####
# Settings specific to the tagged CO simulation
#####

tagged_CO_simulation_options:
  use_fullchem_PCO_from_CH4: true
  use_fullchem_PCO_from_NMVOC: true
```

use_fullchem_PCO_from_CH4

Activates (`true`) or deactivates (`false`) applying the production of CO from CH_4 . This field is archived from a 1-year or 10-year *fullchem* benchmark simulation and is read from disk via HEMCO.

Default value: `true`

use_fullchem_PCO_from_NMVOC

Activates (`true`) or deactivates (`false`) applying the production of CO from non-methane volatile organic compounds (VOCs). This field is archived from a 1-year or 10-year *fullchem* benchmark simulation and is read from disk via HEMCO.

Default value: true

9.1.2 HEMCO_Config.rc

GEOS-Chem Classic relies on the [Harmonized Emissions Component \(aka HEMCO\)](#) for file I/O, regridding, and computing emissions fluxes. Settings for HEMCO can be updated in the [HEMCO configuration file](#), which is named `HEMCO_Config.rc`.

The HEMCO online manual at hemco.readthedocs.io contains detailed instructions about the structure and contents of `HEMCO_Config.rc`, so we will not replicate that content in this Guide. Instead, we will provide a short summary with links to the relevant documentation.

General HEMCO settings

Define general simulation parameters in the [Settings](#) section of `HEMCO_Config.rc`. This includes data paths, global diagnostic options, and verbose output options.

```
#####
### BEGIN SECTION SETTINGS
#####

ROOT:                /path/to/hemco/data/dir
METDIR:              /path/to/hemco/met/dir
GCAPSCENARIO:        not_used
GCAPVERTRES:         47
Logfile:              *
DiagnFile:            HEMCO_Diagn.rc
DiagnPrefix:          ./OutputDir/HEMCO_diagnostics
DiagnFreq:            00000000 010000
Wildcard:             *
Separator:            /
Unit tolerance:       1
Negative values:       0
Only unitless scale factors: false
Verbose:              false
VerboseOnCores:        root      # Accepted values: root all

### END SECTION SETTINGS ###
```

Extension switches

Turn individual emissions inventories on/off in the [Extension Switches](#) section of `HEMCO_Config.rc`. Emission inventories are specified as either [Base Emissions](#) (i.e. read from files on disk) or [Extensions](#) (i.e. computed using meteorological inputs).

```
#####
### BEGIN SECTION EXTENSION SWITCHES
#####
# ExtNr  ExtName                on/off  Species  Years avail.
0        Base                   : on      *
# ----- MAIN SWITCHES -----
--> EMISSIONS                   :         true
--> METEOROLOGY                  :         true
--> CHEMISTRY_INPUT              :         true
```

(continues on next page)

(continued from previous page)

```
# ----- RESTART FIELDS -----
--> GC_RESTART      :      true
--> HEMCO_RESTART   :      true
# ----- NESTED GRID FIELDS -----
--> GC_BCs          :      false
# ----- REGIONAL INVENTORIES -----
--> APEI            :      false   # 1989-2014
--> NEI2016_MONMEAN :      false   # 2002-2020
--> DICE_Africa     :      false   # 2013
# ----- GLOBAL INVENTORIES -----
--> CEDSv2          :      true    # 1750-2019
--> CEDS_GBDMAPS    :      false   # 1970-2017
--> CEDS_GBDMAPS_byFuelType:      false   # 1970-2017

... etc ...

# -----
100   Custom          : off      -
101   SeaFlux         : on       DMS/ACET/ALD2/MENO3/ETNO3/MOH
102   ParaNOx         : on       NO/NO2/O3/HNO3
--> LUT data format   :          nc
--> LUT source dir     :          $ROOT/PARANOX/v2015-02
103   LightNOx        : on       NO
--> CDF table         :          $ROOT/LIGHTNOX/v2014-07/light_dist.ott2010.dat
104   SoilNOx         : on       NO
--> Use fertilizer NOx :          true

... etc ...

### END SECTION EXTENSION SWITCHES ###
```

Base emissions

Note: You do not have to edit this section if you just wish to run GEOS-Chem Classic with its default emissions configuration.

Specify how emissions and other data sets will be read from disk in the [Base Emissions](#) section of HEMCO_Config.rc.

```
#####
### BEGIN SECTION BASE EMISSIONS
#####

# ExtNrName sourceFile sourceVar sourceTime C/R/E SrcDim SrcUnit Species ScalIDs Cat_
↳ Hier

(( (EMISSIONS

#=====
# --- APEI (Canada) ---
#=====

(( (APEI
0 APEI_NO $ROOT/APEI/v2016-11/APEI.0.1x0.1.nc NOx 1989-2014/1/1/0 RF xy kg/m2/s NO _
↳ 25/1002/115 1 30
```

(continues on next page)

(continued from previous page)

```

0 APEI_CO      $ROOT/APEI/v2016-11/APEI.0.1x0.1.nc CO  1989-2014/1/1/0 RF xy kg/m2/s CO  _
→ 26/52/1002      1 30
0 APEI_SOAP -          -          -          -          -          -          -          -
→ SOAP 26/52/1002/280 1 30
0 APEI_SO2     $ROOT/APEI/v2016-11/APEI.0.1x0.1.nc SOx 1989-2014/1/1/0 RF xy kg/m2/s SO2 _
→ 60/1002          1 30
0 APEI_SO4     -          -          -          -          -          -          -          -
→ 60/65/1002      1 30
0 APEI_pFe     -
... etc ...

### END SECTION BASE EMISSIONS ###

```

Scale factors

Define scale factors for emissions inventories and other data sets in the [Scale Factors](#) section of `HEMCO_Config.rc`.

```

#=====
# --- Scale factors used for species conversions ---
#=====

# Units carbon to species conversions
# Factor = # carbon atoms * MW carbon) / MW species
40 CtoACET MATH:58.09/(3.0*12.0) - - - xy unitless 1
41 CtoALD2 MATH:44.06/(2.0*12.0) - - - xy unitless 1
42 CtoALK4 MATH:58.12/(4.3*12.0) - - - xy unitless 1

... etc ...
# VOC speciations
(( (RCP_3PD.or.RCP_45.or.RCP_60.or.RCP_85
50 KET2MEK      0.25 - - - xy unitless 1
51 KET2ACET     0.75 - - - xy unitless 1
))) RCP_3PD.or.RCP_45.or.RCP_60.or.RCP_85

... etc ...

### END SECTION SCALE FACTORS ###

```

Masks

Define masks for emissions and other data sets in the [Masks](#) section of `HEMCO_Config.rc`

```

#####
### BEGIN SECTION MASKS
#####

# ScalID Name sourceFile sourceVar sourceTime C/R/E SrcDim SrcUnit Oper Lon1/Lat1/
→ Lon2/Lat2

(( (EMISSIONS

#=====
# Country/region masks

```

(continues on next page)

(continued from previous page)

```

=====
(( (APEI
1002 CANADA_MASK $ROOT/MASKS/v2018-09/Canada_mask.geos.1x1.nc          MASK
↪ 2000/1/1/0 C xy 1 1 -141/40/-52/85
)) )APEI

(( (NEI2016_MONMEAN
1007 CONUS_MASK $ROOT/MASKS/v2018-09/CONUS_Mask.01x01.nc          MASK
↪ 2000/1/1/0 C xy 1 1 -140/20/-50/60
)) )NEI2016_MONMEAN

... etc ...

)) )EMISSIONS

### END SECTION MASKS ###

### END OF HEMCO INPUT FILE ###

```

9.1.3 HEMCO_Diagn.rc

In your run directory, you will find a copy of the **HEMCO diagnostic configuration file** (named `HEMCO_Diagn.rc`) corresponding to the *HEMCO_Config.rc* file. You will only need to edit this file if you wish to change the default diagnostic output configuration.

A snippet of the `HEMCO_Diagn.rc` for the *fullchem* simulation is shown below:

```

#####
####  ALD2 emissions                                     ####
#####
EmisALD2_Total      ALD2   -1   -1   -1   3   kg/m2/s  ALD2_emission_flux_from_all_
↪ sectors
EmisALD2_Anthro     ALD2    0    1   -1   3   kg/m2/s  ALD2_emission_flux_from_
↪ anthropogenic
EmisALD2_BioBurn     ALD2  111   -1   -1   2   kg/m2/s  ALD2_emission_flux_from_
↪ biomass_burning
EmisALD2_Biogenic    ALD2    0    4   -1   2   kg/m2/s  ALD2_emission_flux_from_
↪ biogenic_sources
EmisALD2_Ocean       ALD2  101   -1   -1   2   kg/m2/s  ALD2_emission_flux_from_ocean
EmisALD2_PlantDecay  ALD2    0    3   -1   2   kg/m2/s  ALD2_emission_flux_from_
↪ decaying_plants
EmisALD2_Ship        ALD2    0   10   -1   2   kg/m2/s  ALD2_emission_flux_from_ships

```

Columns:

1. netCDF variable name for the requested diagnostic quantity
2. Species name
3. Extension number (-1 means sum over all extensions)
4. Category (-1 means sum over all categories)
5. Hierarchy (-1 means sum over all hierarchies)
6. Dimension of data (1: scalar, 2: lon-lat, 3: lon-lat-lev)
7. Units

8. Value for the long_name netCDF variable attribute

The prefix (e.g. OutputDir/HEMCO_diagnostics) for HEMCO diagnostics output files are specified in the Settings section of the HEMCO_Config.rc file.

9.1.4 HISTORY.rc

You can specify which GEOS-Chem Classic diagnostic outputs you would like to archive with the HISTORY.rc configuration file.

Sample HISTORY.rc diagnostic input file

A simplified HISTORY.rc file is shown below.

```
#=====
# EXPID allows you to specify the beginning of the file path corresponding
# to each diagnostic collection. For example:
#
#   EXPID: ./GEOSChem
#       Will create netCDF files whose names begin "GEOSChem",
#       in this run directory.
#
#   EXPID: ./OutputDir/GEOSChem
#       Will create netCDF files whose names begin with "GEOSChem"
#       in the OutputDir sub-folder of this run directory.
#
#=====
EXPID:  ./OutputDir/GEOSChem
#=====
# %%%% COLLECTION NAME DECLARATIONS %%%%
#
# To disable a collection, place a "#" character in front of its name
#
# NOTE: These are the "default" collections for GEOS-Chem.
# But you can create your own custom diagnostic collections as well.
#=====
COLLECTIONS:  'SpeciesConc' ,
              'SpeciesConcSubset' ,
              'ConcAfterChem' ,
::
#=====
# %%%% THE SpeciesConc COLLECTION %%%%
#
# GEOS-Chem species concentrations (default = advected species)
#
# Available for all simulations
#=====
SpeciesConc.template:      '%y4%m2%d2_%h2%n2z.nc4' ,
SpeciesConc.frequency:     00000000 060000 ,
SpeciesConc.duration:      00000001 000000 ,
SpeciesConc mode:          'instantaneous' ,
SpeciesConc.fields:         'SpeciesConcVV_?ADV?'
                          'SpeciesConcMND_?ADV?'
::
#=====
# %%%% THE SpeciesConcSubset COLLECTION %%%%
```

(continues on next page)

(continued from previous page)

```

#
# Same as the SpeciesConc collection, but will subset data in the horizontal
# and vertical dimensions so that the netCDF diagnostic files will cover
# a smaller region of the globe. This can save disk space and memory.
#
# NOTE: This capability will be available in GEOS-Chem "Classic" 12.5.0
# and later versions.
#
# Available for all simulations
#=====
SpeciesConcSubset.template:      '%y4%m2%d2_%h2%n2z.nc4',
SpeciesConcSubset.frequency:    060000,
SpeciesConcSubset.duration:     00000001 000000,
SpeciesConcSubset.mode:         'instantaneous',
SpeciesConcSubset.LON_RANGE:    -40.0 60.0,
SpeciesConcSubset.LAT_RANGE:    -10.0 50.0,
SpeciesConcSubset.levels:       1 2 3 4 5,
SpeciesConcSubset.fields:       'SpeciesConcVV_?ADV?',
                                'SpeciesConcMND_?ADV?',
::
#=====
# %%% THE ConcAfterChem COLLECTION %%%
#
# Concentrations of OH, HO2, O1D, O3P immediately after exiting the KPP solver
# or OH after the CH4 specialty-simulation chemistry routine.
#
# OH:      Available for all full-chemistry simulations and CH4 specialty sim
# HO2:     Available for all full-chemistry simulations
# O1D, O3P: Available for full-chemistry simulations using UCX mechanism
#=====
ConcAfterChem.template:         '%y4%m2%d2_%h2%n2z.nc4',
ConcAfterChem.frequency:        00000100 000000,
ConcAfterChem.duration:         00000100 000000,
ConcAfterChem.mode:             'time-averaged',
ConcAfterChem.fields:           'OHconcAfterChem',
                                'HO2concAfterChem',
                                'O1DconcAfterChem',
                                'O3PconcAfterChem',
::

```

In this HISTORY.rc file, we are requesting three collections (SpeciesConc, SpeciesConcSubset, and ConcAfterChem). Each collection represents a set of netCDF files that will contain the same diagnostic fields.

Legend

COLLECTIONS:

List of **diagnostic collections** in the HISTORY.rc file.

To turn off a collection, place a comment character (#) before its name. For example:

```

COLLECTIONS:  #'SpeciesConc',
               'SpeciesConcSubset',
               'ConcAfterChem',

```

turns off the SpeciesConc collection.

<collection-name>.template

Determines the date and time format of the netCDF file names that will be created for diagnostic collection <collection-name>.

The string `%y4%m2%d2_%h2%n2z.nc4` will print `YYYYMMDD_hhmmz.nc4` to the end of each netCDF file-name, where:

- `YYYYMMDD` is the date in year/month/day format
- `hhmm` is the time in hour:minutes format.
- `z` denotes “Zulu”, which is an abbreviation for UTC time.
- `.nc4` denotes that the data file is in the netCDF-4 format.

<collection-name>.frequency

Determines how often the diagnostic fields belonging to collection <collection-name> collection will be written to a netCDF file. For example:

- `010000` schedules diagnostic archival each hour.
- `00000100 000000` schedules diagnostic output each month.
- `00000001 000000` (or `240000`) schedules diagnostic output each day.
- ... etc. ...

<collection-name>.duration

Determines how often a new netCDF file will be created for collection <collection-name>. For example:

- `010000` creates a new netCDF each hour.
- `00000100 000000` creates a new netCDF file each month. month.
- `00000001 000000` (or `240000`) creates a new netCDF file each day.

<collection-name>.mode

Determines the averaging method for collection <collection-name>. Accepted values are:

instantaneous

Data will be archived as instantaneous “snapshots” at the frequency specified in *<collection-name>.frequency*.

time-averaged

Data will be time-averaged with the frequency specified in *<collection-name>.frequency*.

<collection-name>.fields

A list of the diagnostic fields that will be included in collection <collection-name>.

A single underscore `_` denotes a **species-based diagnostic field**. To request output for a single species, list the species name immediately after the underscore, such as:

```
SpeciesConc.fields:  'SpeciesConcVV_NO'
                    'SpeciesConcVV_O3'
                    'SpeciesConcVV_CO'
                    ... etc ...
::
```

You may also use a **wildcard** such as `?ADV?`, which requests all advected species:

```
SpeciesConc.fields:  'SpeciesConcVV_?ADV?'
                    ... etc ...
::
```

The complete wildcard list is shown below. Wildcards are case-insensitive.

- ?ADV?: Only the advected species
- ?AER?: Only the aerosol species
- ?ALL?: All GEOS-Chem species
- ?DRYALT?: Only the dry-deposited species whose concentrations we wish to archive at a given altitude above the surface. (In practice these are only O₃ and HNO₃.)
- ?DRY?: Only the dry-deposited species
- ?FIX?: Only the inactive (aka “fixed”) species in the KPP chemical mechanism
- ?GAS?: Only the gas-phase species
- ?HYG?: Only aerosols that undergo hygroscopic growth (sulfate, BC, OC, SALA, SALC)
- ?LOS?: Only chemical loss species or families
- ?KPP?: Only the KPP species
- ?PHO?: Only the photolyzed species
- ?VAR?: Only the active (aka “variable”) species in the KPP chemical mechanism
- ?WET?: Only the wet-deposited species
- ?PRD?: Only chemical production species or families
- ?DUSTBIN?: Only the dust bin number
- ?PHOTOBIN? Number of a given wavelength bin for photolysis

To include fields from the `State_Chm` object in collection `<collection-name>`, precede the field name with `Chem_`:

```
'Chem_phCloud',
... etc ...
```

To include fields from the `State_Met` object in collection `<collection-name>`, precede the field name with `Met_`:

```
'Met_T',
'Met_PS',
'Met_SPHU',
... etc ...
```

Both `Chem_` and `Met_` specifiers are case-insensitive.

<collection-name>.LON_RANGE

Optional. Restrict data fields of collection `<collection-name>` to the range `min_lon`, `max_lon`.

<collection-name>.LAT_RANGE

Optional. Restrict data fields of collection `<collection-name>` to the range `min_lat`, `max_lat`.

<collection-name>.levels

Optional. Restrict data fields of collection `<collection-name>` to the specified levels (e.g., 1, 2, 3, 4, 5 or 1–5).

::

Signifies the end of the definition section for collection `<collection-name>`.

All of the above-mentioned files are included in your *GEOS-Chem Classic run directory*.

Please see our *Customize simulations with research options* Supplemental Guide to learn how you can customize your simulation by activating alternate science options in your simulations.

9.2 Less-commonly updated configuration files

If you need to add or delete species, or to change the default photolysis and/or chemistry mechanism settings in your simulation, you'll need to edit these configuration files:

9.2.1 species_database.yml

Note: You will only need to edit `species_database.yml` if you are adding new species to a GEOS-Chem simulation.

The *GEOS-Chem Species Database* is a **YAML** file that contains a listing of metadata for each species used by GEOS-Chem. The Species Database is included in your run directory as file `species_database.yml`, a snippet of which is shown below.

```
# GEOS-Chem Species Database
# Core species only (neglecting microphysics)
# NOTE: Anchors must be defined before any variables that reference them.
A3O2:
  Formula: CH3CH2CH2OO
  FullName: Primary peroxy radical from C3H8
  Is_Gas: true
  MW_g: 75.10
ACET:
  DD_F0: 1.0
  DD_Hstar: 1.0e+5
  Formula: CH3C(O)CH3
  FullName: Acetone
  Henry_CR: 5500.0
  Henry_K0: 2.74e+1
  Is_Advected: true
  Is_DryDep: true
  Is_Gas: true
  Is_Photolysis: true
  MW_g: 58.09
... etc ...

AERI:
  DD_DvzAerSnow: 0.03
  DD_DvzMinVal: [0.01, 0.01]
  DD_F0: 0.0
  DD_Hstar: 0.0
  Formula: I
  FullName: Iodine on aerosol
  Is_Advected: true
  Is_Aerosol: true
  Is_DryDep: true
  Is_WetDep: true
```

(continues on next page)

(continued from previous page)

```
MW_g: 126.90
WD_AerScavEff: 1.0
WD_KcScaleFac: [1.0, 0.5, 1.0]
WD_RainoutEff: [1.0, 0.0, 1.0]
WD_RainoutEff_Luo: [0.4, 0.0, 1.0]

... etc ...
```

Important: Species NO (nitrogen oxide) must be listed in `species_database.yml` as 'NO' :. This will avoid YAML readers mis-intepreting this as no (meaning false).

Each species name begins in the first column of the file, followed by a :. Underneath the species name follows an indented block of *species properties* in Property: Value format.

Some properties listed above are only applicable to gas-phase species, and others to aerosol species. But at the very least, each species should have the following properties defined:

- Formula
- FullName
- MW_g
- Either Is_Gas or Is_Aerosol

For more information about species properties, please see [View GEOS-Chem species properties](#) in the Supplemental Guides section.

9.2.2 Photolysis and chemistry configuration files

Note: You won't need to edit these configuration files unless you are changing the photolysis and/or chemistry mechanisms used in your GEOS-Chem Classic simulation.

Photolysis configuration files

These are found in the `ExtData/CHEM_INPUTS/CLOUD-J/` directory structure. Cloud-J is used by default in GEOS-Chem compute photolysis rates. If using legacy FAST-JX instead then configuration files are located in the `ExtData/CHEM_INPUTS/FAST-JX/` directory. See the configuration file `geoschem_config.yml` for which subdirectory within these folders you are configured to use. See the README in the data directory for information about these files.

Chemical mechanism configuration files

GEOS-Chem Classic simulations use source code generated by [The Kinetic PreProcessor](#). If you need to update the default chemistry mechanism, you will need to do the following steps:

1. Modify the relevant KPP configuration files (described below);
2. Run KPP to generate updated source code for GEOS-Chem Classic;
3. *Compile GEOS-Chem Classic* to create a new executable;
4. Start your GEOS-Chem Classic simulation.

Chemical mechanism configuration files are located in these folders:

KPP/fullchem

Contains configuration files for the default “full-chemistry” mechanism (NO_x + O_x + aerosols + Br + Cl + I).

- `fullchem.kpp`: Main configuration file for the **fullchem** mechanism.
- `fullchem.eqn`: List of species and reactions for the **fullchem** mechanism.

KPP/carbon

Contains configuration files for the carbon gases mechanism (CH₄-CO₂-CO-OCS):

- `carbon.kpp`: Main configuration file for the **carbon** mechanism.
- `carbon.eqn`: List of species and reactions for the **carbon** mechanism.

KPP/custom

Contains configuration files that you can edit if you need to create a custom mechanism. We recommend that you create the custom in this folder and leave `KPP/fullchem` and `KPP/Hg` untouched.

- `custom.kpp`: Copy of `fullchem.kpp`
- `custom.eqn`: Copy of `fullchem.eqn`.

KPP/Hg

Contains configuration files for the mercury chemistry mechanism:

- `Hg.kpp`: Main configuration file for the **Hg** mechanism.
- `Hg.eqn`: List of species and reactions for the **Hg** mechanism.

Please see the following references for more information about KPP:

1. The KPP user manual (kpp.readthedocs.io)
2. Supplemental Guide: *Update chemical mechanisms with KPP*

DOWNLOAD INPUT DATA

In the following chapters, you will learn how to download input data for your GEOS-Chem simulation:

Note: If you are located at an institution with several GEOS-Chem users, the input data for GEOS-Chem Classic may have already been downloaded to a shared data directory on your system. If this is the case for you, feel free to skip ahead to the *Run your simulation* chapter.

10.1 Input data for GEOS-Chem Classic

GEOS-Chem Classic reads (via [HEMCO](#)) several data files from disk during a simulation. These can be grouped into the following categories:

1. *Initial conditions input data* (aka *Restart files*)
2. *Chemistry input data*
3. *Emissions input data*
4. *Meteorology input data*

10.1.1 Data portals

Input data files for GEOS-Chem can be downloaded from one of the following portals:

WashU

The primary data portal for GEOS-Chem, geoschemdata.wustl.edu

Note: The WashU data portal may be unavailable at times due to regularly-scheduled maintenance periods. Please check the [WashU IT status page](#) for more information.

Amazon

GEOS-Chem data on the Amazon cloud, [s3://gcgrid](https://s3.amazonaws.com/gcgrid)

Rochester

Portal for the GCAP 2.0 meteorological data files, atmos.earth.rochester.edu

10.1.2 Initial conditions input data

Initial conditions include:

- Initial species concentrations (aka *Restart files*) used to start a GEOS-Chem simulation.

Download method	From portals
<i>Dry run simulation</i>	<i>WashU Amazon Rochester</i>
Run <i>bashdatacatalog</i> on the InitialConditions.csv file ¹	<i>WashU</i>
Direct data download (FTP or wget)	<i>WashU Amazon Rochester</i>
Globus, use endpoint GEOS-Chem data (WashU)	<i>WashU</i>

¹ We provide InitialConditions.csv files (for each GEOS-Chem version since 13.0.0) at our [input-data-catalogs](#) Github repository.

10.1.3 Chemistry input data

Chemistry input data includes:

- Quantum yields and cross sections for photolysis using either Cloud-J or legacy FAST-JX
- Climatology data for **Linoz**
- Boundary conditions for **UCX** stratospheric chemistry routines

Download method	From portals
<i>Dry run simulation</i>	<i>WashU Amazon Rochester</i>
Run <i>bashdatacatalog</i> on the ChemistryInputs.csv file ²	<i>WashU</i>
Direct data download (FTP or wget)	<i>WashU Amazon Rochester</i>
Globus, use endpoint GEOS-Chem data (WashU)	<i>WashU</i>

² We provide ChemistryInputs.csv files (for each GEOS-Chem version since 13.0.0) at our [input-data-catalogs](#) Github repository.

10.1.4 Emissions input data

Emissions input data includes the following data:

- Emissions inventories
- Input data for HEMCO Extensions
- Input data for GEOS-Chem specialty simulations
- Scale factors
- Mask definitions
- Surface boundary conditions
- Leaf area indices
- Land cover map

Download method	From portals
<i>Dry run simulation</i>	<i>WashU Amazon Rochester</i>
Run <i>bashdatacatalog</i> on the <code>EmissionsInputs.csv</code> file ³	<i>WashU</i>
Direct data download (FTP or wget)	<i>WashU Amazon Rochester</i>
Globus, use endpoint GEOS-Chem data (WashU)	<i>WashU</i>

³ We provide `EmissionsInputs.csv` files (for each GEOS-Chem version since 13.0.0) at our [input-data-catalogs](#) Github repository.

10.1.5 Meteorology input data

As described previously, GEOS-Chem Classic be driven by the following meteorology products:

1. MERRA-2
2. GEOS-FP
3. GCAP 2.0

Download method	From portals
<i>Dry run simulation</i>	<i>WashU Amazon Rochester</i>
Run <i>bashdatacatalog</i> on the <code>MeteorologyInputs.csv</code> file ⁴	<i>WashU</i>
Direct data download (FTP or wget)	<i>WashU Amazon Rochester</i>
Globus, use endpoint GEOS-Chem data (WashU)	<i>WashU</i>

⁴ We provide a `MeteorologyInputs.csv` file at our [input-data-catalogs](#) Github repository.

10.2 Restart files

In the following chapters, you will learn about **restart files** and how they are used.

10.2.1 What is a restart file?

Restart files contain the initial conditions (cf. *Initial conditions input data*) for a GEOS-Chem simulation. GEOS-Chem simulations need two restart files.

GEOSChem.Restart.YYYYMMDD_hhmmz.nc4

Format: netCDF

Description: The *GEOS-Chem Classic restart file*. Contains species concentrations that are read at simulation startup.

GEOS-Chem writes a restart file at the end of each simulation. This allows a long simulation to be split into several individual run stages. For example, the restart file that was created at 00:00 UTC on August 1, 2019 is named: `GEOSChem.Restart.20190801_0000z.nc4`. The `z` character indicates “Zulu” time (aka UTC).

GEOS-Chem restart files are created in the *Restarts/* folder of your *GEOS-Chem run directory* directory.

HEMCO_restart.YYYYMMDDhhmm.nc

Format: netCDF

Description: The *HEMCO restart file*. HEMCO archives certain quantities (mostly pertaining to soil NO_x and biogenic emissions) in order to facilitate long GEOS-Chem simulations with several run stages.

HEMCO restart files are created in the *Restarts/* folder of your *GEOS-Chem run directory* directory.

When you run a GEOS-Chem simulation, it will write new GEOS-Chem restart files at the intervals you specify in *HISTORY.rc*. New HEMCO restart files are written with frequency configured in *HEMCO_Config.rc*.

Viewing and manipulating restart files

Please see the following sections of our Supplemental Guide: *Work with netCDF files* for more information on how you can view and manipulate data in restart files:

1. *Useful tools*
2. *Regrid netCDF files*
3. *Add a new variable to a netCDF file*
4. *Crop netCDF files*
5. *Chunk and deflate a netCDF file to improve I/O*

10.2.2 GEOS-Chem restart files

How are restart files read into GEOS-Chem?

GEOS-Chem restart files are read via *HEMCO*. The entries listed below have been added to *HEMCO_Config.rc* (and may vary slightly for different simulation types). These fields are obtained from HEMCO and copied to the appropriate *State_Chm* and *State_Met* fields in routine *Get_GC_Restart* (located in *GeosCore/hcoi_gc_main_mod.F90*).

```
#=====
# --- GEOS-Chem restart file ---
#=====
((GC_RESTART
* SPC_                ./Restarts/GEOSChem.Restart.$YYYY$MM$DD_$HH$MNz.nc4 SpeciesRst_?ALL?_
→ $YYYY/$MM/$DD/$HH EFYO xyz 1 * - 1 1
* DELPDRY            ./Restarts/GEOSChem.Restart.$YYYY$MM$DD_$HH$MNz.nc4 Met_DELPDRY
→ $YYYY/$MM/$DD/$HH EY   xyz 1 * - 1 1
* KPP_HVALUE         ./Restarts/GEOSChem.Restart.$YYYY$MM$DD_$HH$MNz.nc4 Chem_KPPHvalue
→ $YYYY/$MM/$DD/$HH EY   xyz 1 * - 1 1
* WETDEP_N           ./Restarts/GEOSChem.Restart.$YYYY$MM$DD_$HH$MNz.nc4 Chem_
→ WetDepNitrogen $YYYY/$MM/$DD/$HH EY   xy 1 * - 1 1
* DRYDEP_N           ./Restarts/GEOSChem.Restart.$YYYY$MM$DD_$HH$MNz.nc4 Chem_
→ DryDepNitrogen $YYYY/$MM/$DD/$HH EY   xy 1 * - 1 1
* SO2_AFTERCHEM      ./Restarts/GEOSChem.Restart.$YYYY$MM$DD_$HH$MNz.nc4 Chem_
→ SO2AfterChem   $YYYY/$MM/$DD/$HH EY   xyz 1 * - 1 1
* H2O2_AFTERCHEM     ./Restarts/GEOSChem.Restart.$YYYY$MM$DD_$HH$MNz.nc4 Chem_
→ H2O2AfterChem  $YYYY/$MM/$DD/$HH EY   xyz 1 * - 1 1
* AEROH2O_SNA        ./Restarts/GEOSChem.Restart.$YYYY$MM$DD_$HH$MNz.nc4 Chem_AeroH2OSNA
→ $YYYY/$MM/$DD/$HH EY   xyz 1 * - 1 1
* ORVCSSEQ           ./Restarts/GEOSChem.Restart.$YYYY$MM$DD_$HH$MNz.nc4 Chem_ORVCSSEQ
→ $YYYY/$MM/$DD/$HH EY   xyz 1 * - 1 1
* JOH                ./Restarts/GEOSChem.Restart.$YYYY$MM$DD_$HH$MNz.nc4 Chem_JOH
→ $YYYY/$MM/$DD/$HH EY   xy 1 * - 1 1
* JNO2               ./Restarts/GEOSChem.Restart.$YYYY$MM$DD_$HH$MNz.nc4 Chem_JNO2
→ $YYYY/$MM/$DD/$HH EY   xy 1 * - 1 1
```

(continues on next page)

(continued from previous page)

```
* STATE_PSC      ./Restarts/GEOSChem.Restart.$YYYY$MM$DD_$HH$MNz.nc4 Chem_StatePSC
→ $YYYY/$MM/$DD/$HH EY    xyz count * - 1 1
)) GC_RESTART
```

GEOS-Chem species (the SPC_ entry) use HEMCO time cycle flag EFYO by default. Other restart file fields use the time cycle flag EY. These are explained below.

E

Exact: Stops with an error if the date of the simulation is different than the file timestamp.

F

Forced: Stops with an error if the file isn't found.

Y

Simulation Year: Only reads the data for the simulation year but not for other years.

O

Once: Does not keep cycling in time but only reads the file once.

When reading the **species concentrations** (time cycle flag: EFYO) from the restart file, HEMCO will cause your simulation to stop with an error if:

1. The restart file is missing, or;
2. Any species is not found in the restart file, or;
3. The date in the restart file (which is usually 20190101 or 20190701, depending on your simulation) differs from the start date listed in [geoschem_config.yml](#).

When reading **other restart file fields** (time cycle flag: EY). HEMCO will

1. The restart file is missing, or
2. The date in the restart file (which is usually 20190101 or 20190701, depending on your simulation) differs from the start date listed in [geoschem_config.yml](#).

Attention: If you wish to spin up a GEOS-Chem simulation with a restart file that has (1) missing species or (2) a timestamp that does not match the start date in [geoschem_config.yml](#), simply change the time cycle flag from

```
* SPC_ ... $YYYY/$MM/$DD/$HH EFYO xyz 1 * - 1 1
```

to

```
* SPC_ ... $YYYY/$MM/$DD/$HH CYS xyz 1 * - 1 1
```

This will direct HEMCO to read the closest date available (C), to use the simulation year (Y), and to skip any species (S) not found in the restart file.

Skipped species will be assigned the initial concentration (units: mol mol^{-1} w/r/t dry air) specified by its [BackgroundVV](#) entry in [species_database.yml](#). If the species does not have a [BackgroundVV](#) value specified, then its initial concentration will be set to 1.0×10^{-20} instead.

How can I determine the date of a restart file?

To determine the date of a netCDF restart file, you may use **ncdump**. For example:

```
ncdump -v time -t GEOSChem.Restart.YYYYMMDD_hhmmz.nc4
```

The **-t** option will return the time value in human-readable date-time strings rather than numerical values in unit such as "hours since 1985-1-1 00:00:0.0".

Where can I get a restart file for my simulation?

GEOS-Chem Classic *run directories* are configured to use sample GEOS-Chem restart files in **netCDF** format. These files are available for download at: http://geoschemdata.wustl.edu/ExtData/GEOSCHEM_RESTARTS/.

Tip: We recommend that you download restart files to your disk space with either a *dry-run simulation* or with the *bashdatacatalog*. This will ensure that the proper files will be downloaded.

If you have the `ExtData/GEOSCHEM_RESTARTS` folder in your GEOS-Chem data paths, then a sample restart file will be copied to your run directory when you *generate a new GEOS-Chem classic run directory*.

Monthly GEOS-Chem restart files from the GEOS-Chem 14.0.0 10-year benchmark may be found at http://ftp.as.harvard.edu/gcgrid/geos-chem/10yr_benchmarks/14.0.0/GCCClassic/restarts.

Attention: The sample restart files do not reflect the actual atmospheric state and should only be used to “spin up” the model. In other words, they should be used as initial values in an initialization simulation to generate more accurate initial conditions for your production runs.

For how long should I spin up before starting a production simulation?

Doing a 6-month year spin up is usually sufficient for full-chemistry simulations. We recommend ten years for ozone, carbon dioxide, and methane simulations, and four years for radon-lead-beryllium simulations. If you are in doubt about how long your spin up should be for your simulation, we recommend contacting the *GEOS-Chem Working Group* that specializes in your area of research.

You may spin up the model starting at any year for which there is met data, but you should always start your simulations at the month and day corresponding to the restart file to more accurately capture seasonal variation. If you want to start your production run at a specific date, we recommend doing a spin up for the appropriate number of years plus the number of days needed to reach your ultimate start date. For example, if you want to do a production simulation starting on 2019/12/01, you could spin up the model for one year using the initial GEOS-FP restart file dated 2019/07/01 and then use the new restart file to spin up the model for five additional months, from 2019/07/01 to 2019/12/01.

See also this discussion on our Github page for further guidance: <https://github.com/geoschem/geos-chem/discussions/911>.

How do I check my initial conditions?

To ensure you are using the expected initial conditions for your simulation, please check the GEOS-Chem log file. You should see something like:

```
HEMCO: Opening ./Restarts/GEOSChem.Restart.20190701_0000z.nc4
- Found all CN      met fields for 2011/01/01 00:00
- Found all A1      met fields for 2019/07/01 00:30
- Found all A3cld   met fields for 2019/07/01 01:30
- Found all A3dyn   met fields for 2019/07/01 01:30
- Found all A3mstC  met fields for 2019/07/01 01:30
- Found all A3mstE  met fields for 2019/07/01 01:30
- Found all I3      met fields for 2019/07/01 00:00
Initialize DELP_DRY from restart file
- Found all I3      met fields for 2019/07/01 03:00
=====
R E S T A R T   F I L E   I N P U T
Min and Max of each species in restart file [mol/mol]:
Species  1,      ACET: Min = 1.000458833E-22  Max = 6.680149323E-09
Species  2,      ACTA: Min = 6.574137699E-23  Max = 6.108235029E-10
Species  3,      AERI: Min = 4.122849756E-16   Max = 1.213838925E-11
Species  4,      ALD2: Min = 4.186668786E-23   Max = 4.571487633E-09
...
```

If a species is not found in the restart file, you may see something like:

```
Species 178,      pFe: Use background = 9.999999683E-21
```

How are GEOS-Chem restart files written?

GEOS-Chem restart files are now saved via the History component. A **Restart collection** has been defined in [HISTORY.rc](#) and fields saved out to the restart file can be modified in that file.

For more information, please see our documentation about the [Restart collection in GEOS-Chem History diagnostics](#). This documentation is currently on the GEOS-Chem wiki, but will be ported to ReadTheDocs in the near future.

10.2.3 HEMCO restart files

In this chapter, you will learn more about HEMCO restart files.

Do I need a HEMCO restart file for my initial spin-up run?

Using a HEMCO restart file for your initial spin up run is optional. The HEMCO restart file contains fields for initializing variables required for Soil NO_x emissions, MEGAN biogenic emissions, and the UCX chemistry quantities. The HEMCO restart file that comes with a run directory may only be used for the date and time indicated in the filename. HEMCO will automatically recognize when a restart file is not available for the date and time required, and in that case HEMCO will use default values to initialize those fields. You can also force HEMCO to use the default initialization values by setting `HEMCO_RESTART` to false in `HEMCO_Config.rc`.

For more information

Please see the [HEMCO diagnostics](http://hemco.readthedocs.io) (at hemco.readthedocs.io) for more information about restart files and other diagnostic output from HEMCO.

10.3 Download data with a dry-run simulation

Tip: If you are located at an institution with many other GEOS-Chem users, then the *necessary input data* might have already been downloaded and stored in a common directory on your system. Ask your sysadmin or IT support staff.

Another way to download and manage GEOS-Chem input data is with the *bashdatacatalog* tool.

A “dry-run” is a **GEOS-Chem Classic** simulation that steps through time, but does not perform computations or read data files from disk. Instead, a dry-run simulation prints a list of all data files that a regular GEOS-Chem simulation would have read. The dry-run output also denotes whether each data file was found on disk, or if it is missing. This output can be fed to a script which will download the missing data files to your computer system.

You may generate dry-run output for any of the GEOS-Chem Classic simulation types (*fullchem*, *CH4*, *TransportTracers*, etc.)

In the following chapters, you will learn how to you can download data using the output from a dry-run simulation:

10.3.1 Execute a dry-run simulation

Follow the steps below to perform a GEOS-Chem Classic dry-run simulation:

Tip: Also be sure to watch our video tutorial [Using the updated dry-run capability in GEOS-Chem 13.2.1 and later versions](#) at our [GEOS-Chem Youtube Channel](#), which will guide you through these steps.

Complete preliminary setup

Make sure that you have done the following steps;

1. *Downloaded GEOS-Chem Classic source code*
2. *Compiled the source code*
3. *Configured your simulation*

Then doublecheck these settings in the following *configuration files*:

geoschem_config.yml

1. *start_date*: Set the start date and time for your simulation.
2. *end_date*: Set the end date and time for your simulation.
3. *met_field*: Check if the meteorology setting (*GEOS-FP*, *MERRA2*, *GCAP2*) is correct for your simulation.
4. *root_data_dir*: Make sure that the path to ExtData is correct.

HISTORY.rc

1. Set the frequency and duration for the *HISTORY diagnostic* collections to be consistent with the settings in *geoschem_config.yml*.

HEMCO_Config.rc

1. Check the [Settings section](#) to make sure that diagnostic frequency `DiagnFreq:` is set to the interval that you wish (e.g. Monthly, Daily, YYYYMMDD hhmmss, etc).
2. Check the [Extension Settings section](#), to make sure all of the required emissions inventories and data sets for your simulation have been switched on.

Tip: You can reduce the amount of data that needs to be downloaded for your simulation by turning off inventories that you don't need.

Run the executable with the `--dryrun` flag

Run the GEOS-Chem Classic executable file at the command line with the `--dryrun` command-line argument as shown below:

```
$ ./gcclassic --dryrun | tee log.dryrun
```

The `tee` command will send the output of the dryrun to the screen as well as to a file named `log.dryrun`.

The `log.dryrun` file will look somewhat like a regular GEOS-Chem log file but will also contain a list of data files and whether each file was found on disk or not. This information will be used by the `download_data.py` script in the next step.

You may use whatever name you like for the dry-run output log file (but we prefer `log.dryrun`). You will need this file to download data (*see the next chapter*).

10.3.2 Download data from dry-run output

Once you have successfully executed a GEOS-Chem dry-run, you can use the output from the dry-run (contained in the `log.dryrun` file) to download the data files that GEOS-Chem will need to perform the corresponding “production” simulation. You may download from one of several locations, which are described in the following sections.

Important: Before you use the `download_data.py` script, make sure to initialize a Mamba or Conda environment with the relevant command shown below:

```
$ mamba activate ENV-NAME    # If using Mamba
$ conda activate ENV-NAME    # If using Conda
```

Here `ENV-NAME` is the name of your environment.

Also make sure that you have installed the PyYAML module to your conda environment. PyYAML will allow the `download_data.py` script to read certain configurable settings from a YAML file in your run directory.

The Python environment for GCPy has all of the proper packages that you need to download data from a dry-run simulation. For more information, please see gcpy.readthedocs.io.

Choose a data portal

You can download input data from one of the following locations:

The `geoschemdata.wustl.edu` site (aka WashU)

If you are using GEOS-Chem on your institutional computer cluster, we recommend that you **download data from the WashU (Washington University in St. Louis) site** (<http://geoschemdata.wustl.edu>). This site, which is maintained by Randall Martin's group at WashU, is the main data site for GEOS-Chem.

Tip: We have also set up a [Globus](#) endpoint named **GEOS-Chem data (WashU)** on the WashU site. If you need to download many years of data, it may be faster to use Globus (particularly if your home institution supports it).

The `s3://gcgrid` bucket (aka Amazon)

If you are running GEOS-Chem Classic on the Amazon Web Services cloud, you can quickly **download the necessary data for your GEOS-Chem simulation from the `:file:`s3://gcgrid`` bucket** to the Elastic Block Storage (EBS) volume attached to your cloud instance.

Navigate to your GEOS-Chem Classic run directory and type:

```
$ ./download_data.py log.dryrun amazon
```

This will start the data download process using the `aws s3 cp` commands, which should execute much more quickly than if you were to download the data from another location. It will also produce a **log of unique data files**.

Important: Copying data from `s3://gcgrid` to the EBS volume of an Amazon EC2 cloud instance is always free. But if you download data from `s3://gcgrid` to your own computer system, you will incur an egress fee. **PROCEED WITH CAUTION!**

The `atmos.earth.rochester.edu` site (aka Rochester)

The U. Rochester site (which is maintained by Lee Murray's research there) contains the GCAP 2.0 met field data. This met field data is useful if you wish to perform simulations stretching back into the preindustrial period, or running into the future.

To download data from the Rochester site, type:

```
$ ./download_data.py log.dryrun rochester
```


Run the download_data.py script on the dryrun log file

Navigate to your GEOS-Chem run directory where you executed the dry-run and type:

```
$ ./download_data.py log.dryrun washu
```

The download_data.py Python program is included in the *GEOS-Chem run directory* that you created. This Python program creates and executes a temporary bash script containing the appropriate `wget` commands to download the data files. (We have found that this is the fastest method.)

The download_data.py program will also generate a **log of unique data files** (i.e. with all duplicate listings removed), which looks similar to this:

```
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!!! LIST OF (UNIQUE) FILES REQUIRED FOR THE SIMULATION
!!! Start Date      : 20160701 000000
!!! End Date        : 20160701 010000
!!! Simulation       : standard
!!! Meteorology      : GEOSFP
!!! Grid Resolution  : 4.0x5.0
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
./GEOSChem.Restart.20160701_0000z.nc4 --> /n/holylnfs/EXTERNAL_REPOS/GEOS-CHEM/gcgrid/
↪data/ExtData/GEOSCHEM_RESTARTS/v2018-11/initial_GEOSChem_rst.4x5_standard.nc
./HEMCO_Config.rc
./HEMCO_Diagn.rc
./HEMCO_restart.201607010000.nc
./HISTORY.rc
./input.geos
/n/holylnfs/EXTERNAL_REPOS/GEOS-CHEM/gcgrid/data/ExtData/CHEM_INPUTS/FAST_JX/v2019-10/
↪FJX_j2j.dat
/n/holylnfs/EXTERNAL_REPOS/GEOS-CHEM/gcgrid/data/ExtData/CHEM_INPUTS/FAST_JX/v2019-10/
↪FJX_spec.dat
/n/holylnfs/EXTERNAL_REPOS/GEOS-CHEM/gcgrid/data/ExtData/CHEM_INPUTS/FAST_JX/v2019-10/
↪dust.dat
/n/holylnfs/EXTERNAL_REPOS/GEOS-CHEM/gcgrid/data/ExtData/CHEM_INPUTS/FAST_JX/v2019-10/
↪h2so4.dat
/n/holylnfs/EXTERNAL_REPOS/GEOS-CHEM/gcgrid/data/ExtData/CHEM_INPUTS/FAST_JX/v2019-10/
↪jv_spec_mie.dat
... etc ...
```

This name of this “unique” log file will be the same as the log file with dryrun output, with `.unique` appended. In our above example, we passed `log.dryrun` to `download_data.py`, so the “unique” log file will be named `log.dryrun.unique`. This “unique” log file can be very useful for documentation purposes.

Skip download, but create log of unique files

If you wish to only produce the *log of unique data files without downloading any data, then type the following command from within your GEOS-Chem run directory:

```
$ ./download_data.py log.dryrun --skip-download
```

or for short:

```
$ ./download_data.py log.dryrun --skip
```

This can be useful if you already have the necessary data downloaded to your system but wish to create the log of unique files for documentation purposes (such as for benchmark simulations, etc.)

RUN YOUR SIMULATION

In the following chapters, you will learn how to run your GEOS-Chem Classic simulation on your computer system.

11.1 Create a run script

We recommend that you create a **run script** for your GEOS-Chem simulation. This is a bash script containing the commands to run GEOS-Chem.

A sample GEOS-Chem run script is provided for you in the GEOS-Chem Classic *run directory*. You can edit this script as necessary for your own computational system.

Navigate to your run directory. Then copy the `runScriptSamples/geoschem.run` sample run script into the run directory:

```
cp ./runScriptSamples/geoschem.run .
```

The `geoschem.run` script looks like this:

```
#!/bin/bash

#SBATCH -c 8
#SBATCH -N 1
#SBATCH -t 0-12:00
#SBATCH -p MYQUEUE
#SBATCH --mem=15000
#SBATCH --mail-type=END

#####
### Sample GEOS-Chem run script for SLURM
### You can increase the number of cores with -c and memory with --mem,
### particularly if you are running at very fine resolution (e.g. nested-grid)
#####

# Set the proper # of threads for OpenMP
# SLURM_CPUS_PER_TASK ensures this matches the number you set with -c above
export OMP_NUM_THREADS=$SLURM_CPUS_PER_TASK

# Set the stacksize memory to the highest possible limit
ulimit -s unlimited
export OMP_STACKSIZE=500m

# Run GEOS-Chem. The "time" command will return CPU and wall times.
# Stdout and stderr will be directed to the "GC.log" log file
```

(continues on next page)

(continued from previous page)

```
# (you can change the log file name below if you wish)
srun -c $OMP_NUM_THREADS time -p ./gcclassic > GC.log 2>&1

# Exit normally
exit 0
```

The sample run script contains commands for the **SLURM** scheduler, which is used on many HPC systems.

Note: If your computer system uses a different scheduler (such as LSF or PBS), then you can replace the SLURM-specific commands with commands for your scheduler. Ask your IT staff for more information.

Important commands in the run script are listed below:

#SBATCH -c 8
Tells SLURM to request 8 computational cores.

#SBATCH -N 1
Tells SLURM to request 1 computational node.

Important: GEOS-Chem Classic uses **OpenMP**, which is a shared-memory parallelization model. Using OpenMP limits GEOS-Chem Classic to one computational node.

#SBATCH -t 0-12:00
Tells SLURM to request 12 hours of computational time. The format is D-hh:mm or (days-hours:minutes).

#SBATCH -p MYQUEUE
Tells SLURM to run GEOS-Chem Classic in the computational partition named MYQUEUE. Ask your IT staff for a list of the available partitions on your system.

#SBATCH --mem=15000
Tells SLURM to reserve 15000 MB (15 GB) of memory for the simulation.

#SBATCH --mail-type=END
Tells SLURM to send an email upon completion (successful or unsuccessful) of the simulation.

export OMP_NUM_THREADS=\$SLURM_CPUS_PER_TASK
Specifies how many computational cores that GEOS-Chem Classic should use. The environment variable SLURM_CPUS_PER_TASK will fill in the number of cores requested (in this example, we used #SBATCH -c 8, which requests 8 cores).

ulimit -s unlimited
Tells the bash shell to remove any restrictions on stack memory. This is the place in GEOS-Chem's memory where temporary variables (including *PRIVATE variables for OpenMP parallel loops*) get created.

export OMP_STACKSIZE=500m
Tells the GEOS-Chem executable to use as much memory as it needs for allocating *PRIVATE variables in OpenMP parallel loops*.

srun -c \$OMP_NUM_THREADS
Tells SLURM to run the GEOS-Chem Classic executable using the number of cores specified in *OMP_NUM_THREADS*.

time -p ./gcclassic > GC.log 2>&1
Executes the GEOS-Chem Classic executable and pipes the output (both stdout and stderr streams) to a file named GC.log.

The `time -p` command will print the amount of time (both CPU time and wall time) that the simulation took to complete to the end of `GC.log`.

11.2 Complete this pre-run checklist

Now that you have created a *run script* for GEOS-Chem Classic, take a moment to make sure that you have completed all required setup steps before running your simulation.

11.2.1 First-time setup

1. Make sure that your computational environment meets all of the *hardware* and *software* requirements for GEOS-Chem Classic.

11.2.2 Each-time setup

1. Make sure that you have *properly configured your login environment* (i.e. load necessary software modules after login, etc.)
2. Create a *GEOS-Chem Classic run directory*, and make sure that it is correct for the simulation you wish to perform.

Attention: The initial *restart file that is included with your run directory* does not reflect the actual atmospheric state and should only be used to “spin-up” the model. We recommend a spin-up period of 6 months to 1 year (depending on the type of simulation you are using).

3. *Edit configuration files* to specify the runtime behavior of GEOS-Chem Classic..

Attention: Prior to running with *GEOS-FP* met fields, please be aware of several caveats regarding that data stream. (cf. [The GEOS-FP wiki page](#)).

4. *Configure and build* the source code into an executable file.
5. Copy a sample *GEOS-Chem Classic run script* to your run directory and edit it for the particulars of your simulation and computer system.
6. Make sure that your run script contains the proper settings for *OpenMP parallelization*.
7. Be aware of *ways in which you can speed up your GEOS-Chem Classic simulations*.

11.3 Submit your run script to a scheduler

Many shared computer systems use a **scheduler** to determine in which order submitted jobs will run.

If your computer system uses the **SLURM scheduler**, then you can use the following command to submit your GEOS-Chem Classic *run script* to a computational queue:

```
sbatch geoschem.run
```

The SLURM scheduler will then decide when your job starts based on parameters such as current load on the system and past cluster usage (sometimes known as **fairshare**). If there is high demand on the cluster, your job may remain in **pending state** for a few hours (or sometimes days!) before it starts.

If your computer system uses a different scheduler (e.g. LSF, PBS, etc.) then ask your sysadmin or IT staff about the commands that are needed to submit jobs.

11.4 Or run the script from the command line

If your computer system does not use a scheduler, or if you are logged into an Amazon Web Services (AWS) cloud instance, then you can run GEOS-Chem Classic directly from the terminal command line.

Here is a sample run script for interactive use (`geoschem-int.sh`). It is similar to the *run script shown previously*, with a few edits:

```
#!/bin/bash

#####
### Sample GEOS-Chem run script for interactive use
#####

# Set the proper # of threads for OpenMP
export OMP_NUM_THREADS=8

# Max out stack memory available to GEOS-Chem
ulimit -s unlimited
export OMP_STACKSIZE=500m

# Run GEOS-Chem. The "time" command will return CPU and wall times.
# Stdout and stderr will be directed to the "GC.log" log file
# (you can change the log file name below if you wish)
time -p ./gcclassic > GC.log 2>&1

# Exit normally
exit 0
```

The modifications entail:

1. Removing the SLURM-specific commands (i.e. `#SBATCH`, `$SLURM_CPUS__PER_TASK`, and `srun`).
2. Manually specifying the number of cores that you wish GEOS-Chem to use (`export $OMP_NUM_THREADS=8`).

Note: If you are logged into an AWS cloud instance, you can add

```
export OMP_NUM_THREADS=`ncpus`
```

to the run script. This will automatically set `OMP_NUM_THREADS` to the available number of cores.

To run GEOS-Chem interactively, type:

```
$ ./geoschem.run > GC.log 2>&1 &
```

This will run the job in the background. To monitor the progress of the job you can type:

```
$ tail -f GC.log
```

which will show the contents of the log file as they are being written.

Another way to view output from GEOS-Chem in real time is to use the `tee` command. This will print output to the screen and also send the same output to a log file. Type:

```
$ ./geoschem.run | tee GC.log
```

11.5 Verify a successful simulation

There are several ways to verify that your GEOS-Chem Classic run was successful:

1. The following output can be found at the end of the log file:

```
*****      E N D      O F      G E O S  -- C H E M      *****
```

2. NetCDF files (e.g. `OutputDir/GEOSChem*.nc4` and `OutputDir/HEMCO*.nc`) are present.
3. *Restart files* (e.g. `GEOSChem.Restart.YYYYMMDD_hhmmz.nc4` and `HEMCO_restart.YYYYMMDDhh.nc`) for ending date `YYYYMMDD hhmm` are present.
4. Your scheduler log file (e.g. `slurm-xxxxxx.out`, where `xxxxxx` is the job id) is free of errors.

If your run stopped with an error, please see the following resources:

- [Understand what error messages mean](#)
- [Debug GEOS-Chem and HEMCO errors](#)
- [Submitting GEOS-Chem support requests](#)

11.6 Minimize differences in multi-stage runs

If you need to split up a very long simulation (e.g. 1 model year or more) into multiple stages, keep these guidelines in mind:

1. Make sure `GC_RESTART` and `HEMCO_RESTART` options are set to `true`: in [HEMCO_Config.rc](#).
2. To ensure your *restart files* are read and species concentrations are properly initialized, check your GEOS-Chem log file for the following output:

```
=====
R E S T A R T   F I L E   I N P U T
Min and Max of each species in restart file [mol/mol]:``
Species   1,      NO: Min = 1.000000003E-30   Max = 1.560991691E-08
Species   2,      O3: Min = 3.135925075E-09   Max = 9.816152669E-06
Species   3,      PAN: Min = 3.435056848E-25   Max = 1.222619450E-09
...
=====
```

Actual values may differ. If you see `Use background = ...` for most or all species, that suggests your restart file was not found. To avoid using the wrong restart file make sure to use time cycle flag `EY` in `HEMCO_Config.rc` (cf. [How are restart files read into GEOS-Chem?](#)).

11.7 Speed up a slow simulation

GEOS-Chem Classic performance is continuously monitored by the [GEOS-Chem Support Team](#) by means of benchmark simulations and ad-hoc timing tests. It has been shown that running GEOS-Chem with [recommended timesteps](#) from Philip *et al.* [\[\[Philip et al., 2016\]\]](#), can increase run times by approximately a factor of 2. To speed up GEOS-Chem Classic simulations, users may choose to use any of the following options.

11.7.1 Use coarser timesteps

As [discussed previously](#), the default timesteps for GEOS-Chem Classic are 600 seconds for dynamics, and 1200 seconds for chemistry and emissions. You can experiment with using coarser timesteps (such as 1800 seconds for dynamics and 3600 seconds for emissions & chemistry).

Attention: For nested-grid simulations, you might not be able to use coarser timesteps, or else the Courant limit in transport will be violated.

11.7.2 Turn off unwanted diagnostics

Several diagnostics are turned on by default in *the HISTORY.rc* configuration file. The more diagnostics that are turned on, the more I/O operations need to be done, resulting in longer simulation execution times. Disabling diagnostics that you do not wish to archive can result in a faster simulation.

11.7.3 Disable debugging options

If you previously configured GEOS-Chem with the : `CMAKE_BUILD_TYPE` option set to Debug, then several run-time debugging checks will be activated. These include:

- Checking for array-out-of-bounds errors
- Checking for floating-point math exceptions (e.g. div-by-zero)
- Disabling compiler optimizations

These options can be useful in detecting errors in your GEOS-Chem Classic simulation, but result in a much slower simulation. If you plan on running a long Classic simulation, make sure that you [configure and build GEOS-Chem Classic](#) so that `CMAKE_BUILD_TYPE` is set to Release.

VIEW OUTPUT FILES

In this chapter, you will learn more about the **output files** that are generated by GEOS-Chem Classic simulations.

12.1 Log files

Log files redirect the output of Fortran `PRINT*` or `WRITE` statements to a file. You can check the log files for an “echo-back” of simulation options, as well as error messages.

12.1.1 GEOS-Chem and HEMCO log file

File name: `GC.log` (or similar)

Contains an “echo-back” of input options that were specified in *geoschem_config.yml* and *HISTORY.rc*, as well as information about what is happening at each GEOS-Chem timestep. If your GEOS-Chem Classic simulation dies with an error, a detailed error message will be printed in this log file.

In GEOS-Chem 14.1.0 and later versions, information about emissions, met fields, and other relevant data that are read from disk and processed by *HEMCO* is now sent to this log file (instead of to `HEMCO.log`).

12.1.2 GEOS-Chem log file with dry-run output

File name: `log.dryrun` (or similar)

Contains the full path names of all input files (configuration files, meteorology files, emissions files) that are read by GEOS-Chem. This will allow users to download only those files that their GEOS-Chem simulation requires, thus speeding up the data downloading process.

For more information, please see the *dry run* chapter.

12.1.3 GEOS-Chem species metadata log

File name: `OutputDir/geoschem_species_metadata.yml`

Contains metadata (taken from the *GEOS-Chem species database*) in YAML format for only those species that are used in the simulation. This facilitates coupling GEOS-Chem to other Earth System Models.

12.1.4 Timers log file

File name: `gcclassic_timers.json` (in JSON format).

The timers log file is created when you set `use_gcclassic_timers: true` in *the Simulation Settings section of `geoschem_config.yml`*. It contains “wall-clock” times that measure how long each component of GEOS-Chem took to execute. This information is used by the GEOS-Chem benchmarking scripts that execute on the Amazon cloud computing platform.

12.1.5 Scheduler log file

File name: Specific to each scheduler.

If you used a batch scheduler such as SLURM, PBS, LSF, etc. to submit your GEOS-Chem Classic simulation, then output from the Unix stdout and/or stderr streams may be printed to this file. This file may contain important error messages.

12.2 History diagnostics output

GEOS-Chem History diagnostics are comprised of several **diagnostic collections**. Each diagnostic collection contains a series of **diagnostic fields** that may be archived from a GEOS-Chem Classic simulation.

In the *HISTORY.rc* configuration file (which is located in your *GEOS-Chem Classic run directory*, you will find a list of **default diagnostic collections**. These are collections that have been predefined for you. You may edit the *HISTORY.rc* configuration file to select which diagnostic collections you wish to archive from your GEOS-Chem Classic simulation. You may also define your own custom diagnostic collectinons.

The filenames listed below correspond to the default diagnostic collections in the *HISTORY.rc* file.

Table 1: GEOS-Chem History diagnostics output files

History output file	Diagnostic collection	Used in simulations
GEOSChem.AdvFluxVert.YYYYMMDD_hhhmmz.nc4	AdvFluxVert	<i>fullchem</i>
GEOSChem.AerosolMass.YYYYMMDD_hhhmmz.nc4	AerosolMass	<i>fullchem aerosol</i>
GEOSChem.Aerosols.YYYYMMDD_hhhmmz.nc4	Aerosols	<i>fullchem aerosol</i>
GEOSChem.BoundaryConditions.YYYYMMDD_hhhmmz.nc4	BoundaryConditions	Nested-grid simulations
GEOSChem.CH4.YYYYMMDD_hhhmmz.nc4	CH4	<i>CH4</i>
GEOSChem.CloudConvFlux.YYYYMMDD_hhhmmz.nc4	CloudConvFlux	All simulations
GEOSChem.ConcAboveSfc.YYYYMMDD_hhhmmz.nc4	ConcAboveSfc	<i>fullchem</i>
GEOSChem.ConcAfterChem.YYYYMMDD_hhhmmz.nc4	ConcAfterChem	<i>fullchem</i>
GEOSChem.DryDep.YYYYMMDD_hhhmmz.nc4	DryDep	All simulations with dry-depositing species
GEOSChem.JValues.YYYYMMDD_hhhmmz.nc4	JValues	<i>fullchem Hg</i>
GEOSChem.KppDiags.YYYYMMDD_hhhmmz.nc4	KppDiags	<i>fullchem Hg</i>
GEOSChem.LevelEdgeDiags.YYYYMMDD_hhhmmz.nc4	LevelEdgeDiags	All simulations
GEOSChem.MercuryChem.YYYYMMDD_hhhmmz.nc4	MercuryChem	<i>Hg</i>
GEOSChem.MercuryEmis.YYYYMMDD_hhhmmz.nc4	MercuryEmis	<i>Hg</i>
GEOSChem.MercuryOcean.YYYYMMDD_hhhmmz.nc4	MercuryOcean	<i>Hg</i>
GEOSChem.POPs.YYYYMMDD_hhhmmz.nc4	POPs	<i>POPs</i>
GEOSChem.Metrics.YYYYMMDD_hhhmmz.nc4	Metrics	<i>fullchem CH4</i>
GEOSChem.ProdLoss.YYYYMMDD_hhhmmz.nc4	ProdLoss	<i>fullchem aerosol tagCO tagO3</i>
GEOSChem.RadioNuclide.YYYYMMDD_hhhmmz.nc4	RadioNuclide	<i>TransportTracers</i>
GEOSChem.Restart.YYYYMMDD_hhhmmz.nc4	Restart	All simulations
GEOSChem.RxnRates.YYYYMMDD_hhhmmz.nc4	RxnRates	<i>fullchem CH4 Hg</i>
GEOSChem.SatDiagn.YYYYMMDD_hhhmmz.nc4	SatDiagn	All simulations
GEOSChem.SpeciesConc.YYYYMMDD_hhhmmz.nc4	SpeciesConc	All simulations
GEOSChem.StateChm.YYYYMMDD_hhhmmz.nc4	StateChm	All simulations
GEOSChem.StateMet.YYYYMMDD_hhhmmz.nc4	StateMet	All simulations
GEOSChem.WetLossConv.YYYYMMDD_hhhmmz.nc4	WetLossConv	All simulations with wet-depositing species
GEOSChem.WetLossLS.YYYYMMDD_hhhmmz.nc4	WetLossLS	All simulations with wet-depositing species

12.3 Other diagnostic output files

12.3.1 HEMCO diagnostic output

HEMCO diagnostics generate *netCDF-format* files in the `OutputDir/` subdirectory of your *GEOS-Chem run directory*. You may change this filepath by editing the *HEMCO_Config.rc* configuration file.

HEMCO diagnostic files use the naming convention `HEMCO_diagnostics.YYYYMMDDhhmm.nc`, where `YYYYMMDD` and `hhmm` refer to the model date and time at which each file was created.

For more information, please see our HEMCO user manual at hemco.readthedocs.io.

12.3.2 Planeflight diagnostic output

The *GEOS-Chem plane-following diagnostic* generates text files in the top-level of your *GEOS-Chem Classic run directory*. You may change this filepath by editing the *planeflight* section of *geoschem_config.yml*.

Planeflight diagnostic files use the naming convention `plane.log.YYYYMMDDhhmm`, where `YYYYMMDD` refers to the model date at which each diagnostic file is created.

12.3.3 ObsPack diagnostic output

The *GEOS-Chem ObsPack diagnostic* generates *netCDF-format* files in the top-level of your *GEOS-Chem Classic run directory*. You may change this filepath by editing the *ObsPack* section of *geoschem_config.yml*.

ObsPack diagnostic files use the naming convention `GEOS-Chem.ObsPack.YYYYMMDD_hhmmz.nc4`, where `YYYYMMDD` and `hhmm` refers to the model date at which each diagnostic file is created, and `z` refers to UTC (aka Zulu time).

GEOS-Chem Classic and HEMCO also create *restart files*, which have been discussed previously.

DIAGNOSTICS REFERENCE

In the following chapters, you will learn about the types of diagnostic outputs you can generate with GEOS-Chem Classic.

13.1 GEOS-Chem History diagnostics

At present, our [Guide to GEOS-Chem History diagnostics](#) is still located on the GEOS-Chem wiki. We will be migrating this information over to ReadTheDocs very soon.

13.2 HEMCO diagnostics

Information about diagnostic output from HEMCO (the Harmonized Emissions Component) may be found on our [HEMCO ReadTheDocs site](#).

13.3 Planeflight diagnostic

On this page we provide information about the GEOS-Chem planeflight diagnostic, which allows you to save certain diagnostic quantities along flight tracks or at the position of ground observations. This can be more efficient in terms of storage than saving out 3-D data files via the [GEOS-Chem History diagnostics](#).

Attention: Several diagnostic quantities were disabled when the SMVGEAR chemistry solver was replaced with the FlexChem implementation of **KPP** (cf: [Update chemical mechanisms with KPP](#)). Therefore, you may find that functionality is not currently working. We look to GEOS-Chem community members to help us maintain the planeflight diagnostic.

13.3.1 The Planeflight.dat.YYYYMMDD configuration file

The `Planeflight.dat.YYYYMMDD` files allow you to specify the diagnostic quantities (species, reaction rates, met fields) that you want to print out for a specific longitude, latitude, altitude, and time. A sample `Planeflight.dat.YYYYMMDD` file is given below. Of course if you have lots of flight track data points, your file will be much longer.

If the plane flight following diagnostic is switched on, then it will look for a new `Planeflight.dat.YYYYMMDD` file each day. If a `Planeflight.dat.YYYYMMDD:`` file is found for a given day, then GEOS-Chem will save out diagnostic quantities along the flight track(s) to the `plane.log.YYYYMMDD` file.

Format

```

Planeflight.dat -- Input file for planeflight diagnostic
GCST
July 2018
-----
9      <-- # of variables to be output (listed below)
-----
TRA_001
TRA_002
TRA_003
GMAO_TEMP
GMAO_ABSH
GMAO_RELH
GMAO_IIEV
GMAO_JJEV
GMAO_LLEV
-----
Now give the times and locations of the flight
-----
Point   Type DD-MM-YYYY HH:MM   LAT    LON  ALT/PRE   OBS
  1     Scrz 30-06-2012 13:53  -46.43  51.85  202.00  1765.030
  2     Scrz 30-06-2012 13:53  -46.43  51.85  202.00  1765.060
  3     Sush 30-06-2012 16:25  -54.85 -68.31   32.00  1764.750
  4     Sush 30-06-2012 16:25  -54.85 -68.31   32.00  1765.610
  5     S11b 30-06-2012 17:13   54.95 -112.45  588.00  1891.200
  6     S11b 30-06-2012 17:13   54.95 -112.45  588.00  1891.310
99999   END  00-00-0000  00:00    0.00    0.00    0.00    0.000

```

The data in this text file can be read and plotted using GAMAP routines [CTM_READPLANEFLIGHT](#) and [PLANE_PLOT](#).

Requesting diagnostic quantities

The first part of the Planeflight.dat.YYYYMMDD file allows you to request several diagnostic quantities that you would like to be archived along the plane's flight track. These are listed in the table below.

You must make sure that you have specified the number of requested quantities properly, or you will get an input error.

Important: Several planeflight diagnostic quantities had to be disabled when the SMVGear chemical solver was replaced by the FlexChem implementation of the KPP chemical solver. Therefore, you may find that not all of the planeflight diagnostic quantities listed below are still functional. Please report any issues to the GEOS-Chem Support Team by [opening a new Github issue](#).

Table 1: Planeflight diagnostic archivable quantities

Quantity	Description	Units
TRA_nnn	Species concentration (nnn = species index)	v/v dry
OH, HO ₂ , etc.	Species concentration	molec/cm ³
RO ₂	Concentration of RO ₂ family	v/v dry
AN	Concentration of AN family	v/v dry
NO _y	Concentration of NO _y family	v/v dry
GMAO_TEMP	Temperature	K

continues on next page

Table 1 – continued from previous page

Quantity	Description	Units
GMAO_ABSH	Absolute humidity	unitless
GMAO_SURF	Aerosol surface area	cm ² /cm ³
GMAO_PSFC	Surface pressure	hPa
GMAO_UWND	Zonal winds	m/s
GMAO_VWND	Meridional winds	m/s
GMAO_IIEV	Longitude index	unitless
GMAO_JJEV	Latitude index	unitless
GMAO_LLEV	Level index	unitless
GMAO_RELH	Relative humidity	%
GMAO_PSLV	Sea level pressure	hPa
GMAO_AVGW	Water vapor mixing ratio	v/v
GMAO_THTA	Potential temperature	K
GMAO_PRES	Pressure at center of grid box	hPa
GMAO_ICEnn	SEAICEnn fields	unitless
AODC_SULF	Column AOD, sulfate	unitless
AODC_BLKc	Column AOD, black carbon	unitless
AODC_ORGC	Column AOD, organic carbon	unitless
AODC_SALA	Column AOD, fine sea salt	unitless
AODC_SALC	Column AOD, coarse sea salt	unitless
AODC_DUST	Column AOD, dust	unitless
AODB_SULF	Column AOD, sulfate (below aircraft)	unitless
AODB_BLKc	Column AOD, black carbon (below aircraft)	unitless
AODB_ORGC	Column AOD, organic carbon (below aircraft)	unitless
AODB_SALA	Column AOD, fine sea salt (below aircraft)	unitless
AODB_SALC	Column AOD, coarse sea salt (below aircraft)	unitless
AODB_DUST	Column AOD, dust (below the aircraft)	unitless
TMS_nnn	Nucleation rates (TOMAS)	
HG2_FRACG	Frac of Hg(II) in gas phase	unitless
HG2_FRACP	Frac Hg(II) in particle phase	unitless
ISOR_HPLUS	ISORROPIA H ⁺	M
ISOR_PH	ISORROPIA pH	unitless
ISOR_AH2O	ISORROPIA aerosol water	ug/m ³ air
ISOR_HSO4	ISORROPIA bifulfate	M
TIME_LT	Local time	hours
AQAER_RAD	Aqueous aerosol radius	cm
AQAER_SURF	Aqueous aerosol surface area	cm ² /cm ³
PROD_XXXX	Production rates (needs updating)	molec/cm ³ /s
REA_nnn	Reaction rates (Needs updating)	molec/cm ³ /s

Specifying the flight track

The next section of the `Planeflight.dat.YYYYMMDD` file is where you will specify the points that make up the flight track.

Quantity	Description
POINT	A sequential index of flight track points.
TYPE	Identifier for the plane (or station)
DD	Day of the observation
MM	Month of the observation
YYYY	Year of the observation
HH	Hour of the observation (UTC)
MM	Minute of the observation (UTC)
LAT	Latitude (deg), range -90 to +90
LON	Longitude (deg), range -180 to +180
ALT/PRE	Altitude [m] or Pressure [hPa] of the observation
OBS	Value of the observation (if known), used to compare to model output

Important: The `TYPE` column can be used to specify the aircraft type and flight number to distinguish between multiple plane flight tracks.

The planeflight diagnostic will automatically set `L=1` if it does not recognize `TYPE`. When using a new flight track, make sure to add your `TYPE` to [this IF statement](#) if you do not wish to use `L=1` for that type value.

The plane.log.YYYYMMDD output file

The `plane.log.YYYYMMDD` file contains output from the planeflight diagnostic.

Format

POINT	TYPE	YYYYMMDD	HHMM	LAT	LON	PRESS	OBS	T-IND	P-I	I-IND	J-
→IND	TRA_001	GMAO_TEMP	...								
1	Scrz	20120630	1353	-46.43	51.85	981.74	1765.030	000061277	002	00047	→
→00012	1.785E-006	2.780E+002	...								
2	Scrz	20120630	1353	-46.43	51.85	981.74	1765.060	000061277	002	00047	→
→00012	1.785E-006	2.780E+002	...								
3	Sush	20120630	1625	-54.85	-68.31	949.77	1764.750	000061281	002	00023	→
→00010	1.784E-006	2.746E+002	...								
4	Sush	20120630	1625	-54.85	-68.31	949.77	1765.610	000061281	002	00023	→
→00010	1.784E-006	2.746E+002	...								
5	S11b	20120630	1713	54.95	-112.45	876.13	1891.200	000061283	005	00015	→
→00037	1.906E-006	2.942E+002	...								
6	S11b	20120630	1713	54.95	-112.45	876.13	1891.310	000061283	005	00015	→
→00037	1.906E-006	2.942E+002	...								

Columns

Table 2: Cp;

Column	Description
POINT	Flight track data point number (for reference)
TYPE	Aircraft/flight number ID or ground station ID
YYYYMMDD	Year, month, and day (UTC) for each flight track point
HHMM	Hour and minute (UTC) for each flight track point
LAT	Latitude (-90 to 90 degrees) for each flight track point
LON	Longitude (-180 to 180 degrees) for each flight track point
PRESS	Pressure in hPa for each flight track point
OBS	Observation value from the flight campaign
T-IND	Time index
P-IND	GEOS-Chem level index
I-IND	GEOS-Chem longitude index
J-IND	GEOS-Chem latitude index
TRA_001 etc.	Diagnostic quantities requested in <code>Planeflight.dat.YYMMDD</code>

13.4 ObsPack diagnostic

On this page we provide information about the ObsPack diagnostic in GEOS-Chem, which is intended to sample GEOS-Chem data at specified coordinates and times (e.g. corresponding to surface or flight track measurements). This feature was written by Andy Jacobson of NOAA and Andrew Schuh of Colorado State University and implemented in GEOS-Chem 12.2.0.

13.4.1 Specifying ObsPack diagnostic options

The **ObsPack Menu** section of `input.geos` is where you define the following settings:

1. Turning ObsPack on or off
2. Specifying which GEOS-Chem species you would like to archive.
 - At present, you can archive individual species, or all advected species.
3. Specifying the names of input files
 - These are the files from which coordinate data will be read)
4. Specifying the names of output files
 - These are the files that will contain GEOS-Chem data sampled by the ObsPack diagnostic.

13.4.2 Input file format

The ObsPack diagnostic reads input information (such as coordinates, sampling method, and observation ID's) from netCDF files having the format shown below. You will need to prepare an input file for each YYYY/MM/DD date on which you would like to obtain ObsPack diagnostic output from GEOS-Chem. (The ObsPack diagnostic will skip over days on which it cannot find an input file.)

ObsPack input files can be downloaded from NOAA (see <https://www.esrl.noaa.gov/gmd/ccgg/obspack/>).

Attention: Starting in ObsPack v6, `time_components` indicates the start-time of the sampling interval, not the center time. For the center time, we need to read the `time` variable. The `time` variable represents the center of the averaging window in all ObsPack data versions.

Obspack file metadata

Here is the metadata from an ObsPack data file. We have only displayed the variables that the ObsPack module needs to read.

```
netcdf obspack_co2_1_GLOBALVIEWplus_v6.0_2020-09-11.20190408 {
dimensions:
    obs = UNLIMITED ; // (3111 currently)
    calendar_components = 6 ;
    string_of_200chars = 200 ;
    string_of_500chars = 500 ;
variables:
    int obs(obs) ;
        obs:long_name = "obs" ;
        obs:_Storage = "chunked" ;
        obs:_ChunkSizes = 1024 ;
        obs:_Endianness = "little" ;
    int time(obs) ;
        time:units = "Seconds since 1970-01-01 00:00:00 UTC" ;
        time:_FillValue = -999999999 ;
        time:long_name = "Seconds since 1970-01-01 00:00:00 UTC" ;
        time:_Storage = "chunked" ;
        time:_ChunkSizes = 778 ;
        time:_DeflateLevel = 5 ;
        time:_Endianness = "little" ;
    ...
    float latitude(obs) ;
        latitude:units = "degrees_north" ;
        latitude:_FillValue = -1.e+34f ;
        latitude:long_name = "Sample latitude" ;
        latitude:_Storage = "chunked" ;
        latitude:_ChunkSizes = 778 ;
        latitude:_DeflateLevel = 5 ;
        latitude:_Endianness = "little" ;
    float longitude(obs) ;
        longitude:units = "degrees_east" ;
        longitude:_FillValue = -1.e+34f ;
        longitude:long_name = "Sample longitude" ;
        longitude:_Storage = "chunked" ;
        longitude:_ChunkSizes = 778 ;
        longitude:_DeflateLevel = 5 ;
        longitude:_Endianness = "little" ;
```

(continues on next page)

(continued from previous page)

```

float altitude(obs) ;
    altitude:units = "meters" ;
    altitude:_FillValue = -1.e+34f ;
    altitude:long_name = "sample altitude in meters above sea level" ;
    altitude:comment = "Altitude is surface elevation plus sample intake_
↪height in meters above sea level." ;
    altitude:_Storage = "chunked" ;
    altitude:_ChunkSizes = 778 ;
    altitude:_DeflateLevel = 5 ;

...
char obspack_id(obs, string_of_200chars) ;
    obspack_id:long_name = "Unique ObsPack observation id" ;
    obspack_id:comment = "Unique observation id string that includes obs_
↪id, dataset_id and obspack_num." ;
    obspack_id:_Storage = "chunked" ;
    obspack_id:_ChunkSizes = 1, 200 ;
    obspack_id:_DeflateLevel = 5 ;

...
int CT_sampling_strategy(obs) ;
    CT_sampling_strategy:_FillValue = -9 ;
    CT_sampling_strategy:long_name = "model sampling strategy" ;
    CT_sampling_strategy:values = "How to sample model. 1=4-hour avg; 2=1-
↪hour avg; 3=90-min avg; 4=instantaneous" ;
    CT_sampling_strategy:_Storage = "chunked" ;
    CT_sampling_strategy:_ChunkSizes = 778 ;
    CT_sampling_strategy:_DeflateLevel = 5 ;
    CT_sampling_strategy:_Endianness = "little"

... omitting global attributes etc. ...

```

Notes

1. The ObsPack ID string should be 200 characters long.
2. If you have coordinate data in another format (e.g. a text-based *Planeflight.dat* file) then you'll need to create a netCDF file using the format shown above, or else ObsPack will not be able to read it.

13.4.3 Output file format

The ObsPack diagnostic will produce a file called `GEOSChem.ObsPack.YYYYMMDD_hhmmz.nc4` for each day where an *input file* has been specified. (You can change the output file name in the ObsPack Menu in `input.geos`.)

Below is shown an ObsPack output file for the [GEOS-Chem methane simulation](#). If you are using the ObsPack diagnostic with other GEOS-Chem simulations, your output files will look similar to this, except for the species names.

```

netcdf GEOSChem.ObsPack.20180926_0000z.nc4 {
dimensions:
    obs = UNLIMITED ; // (662 currently)
    species = 1 ;
    char_len_obs = 200 ;
variables:
    char obspack_id(obs, char_len_obs) ;
        obspack_id:long_name = "obspack_id" ;

```

(continues on next page)

(continued from previous page)

```

        obspack_id:units = "1" ;
    int nsamples(obs) ;
        nsamples:long_name = "no. of model samples" ;
        nsamples:units = "1" ;
        nsamples:comment = "Number of discrete model samples in average" ;
    int averaging_interval(obs) ;
        averaging_interval:long_name = "Amount of model time over which this_
↪observation is averaged" ;
        averaging_interval:units = "seconds" ;
    int averaging_interval_start(obs) ;
        averaging_interval_start:long_name = "Start of averaging interval" ;
        averaging_interval_start:units = "seconds since 1970-01-01 00:00:00_
↪UTC" ;
        averaging_interval_start:calendar = "standard" ;
    int averaging_interval_end(obs) ;
        averaging_interval_end:long_name = "End of averaging interval" ;
        averaging_interval_end:units = "seconds since 1970-01-01 00:00:00 UTC
↪" ;
        averaging_interval_end:calendar = "standard" ;
    float lon(obs) ;
        lon:long_name = "longitude" ;
        lon:units = "degrees_east" ;
    float lat(obs) ;
        lat:long_name = "latitude" ;
        lat:units = "degrees_north" ;
    float u(obs) ;
        u:long_name = "Zonal component of wind" ;
        u:units = "m s^-1" ;
    float v(obs) ;
        v:long_name = "Meridional component of wind" ;
        v:units = "m s^-1" ;
    float blh(obs) ;
        blh:long_name = "Boundary layer height" ;
        blh:units = "m" ;
    float q(obs) ;
        q:long_name = "mass_fraction_of_water_inair" ;
        q:units = "kg water (kg air)^-1" ;
    float pressure(obs) ;
        pressure:long_name = "pressure" ;
        pressure:units = "Pa" ;
    float temperature(obs) ;
        temperature:long_name = "temperature" ;
        temperature:units = "K" ;
    float CH4(obs) ;
        CH4:long_name = "Methane" ;
        CH4:units = "mol mol^-1" ;
        CH4:_FillValue = -1.e+34f ;

// global attributes:
    :history = "GEOS-Chem simulation at 2019/01/11 14:54" ;
    :conventions = "CF-1.4" ;
    :references = "www.geos-chem.org; wiki.geos-chem.org" ;
    :model_start_date = "2018/09/26 00:00:00 UTC" ;
    :model_end_date = "2018/09/27 00:00:00 UTC" ;
}

```

You can use several different types of netCDF-reading software to read and plot data from Obspack diagnostic output files.

We recommend using either Python scripts or Jupyter notebooks.

13.4.4 Known issues

Unit conversions are currently done for all species

In routine `ObsPack_Sample` (located in module `ObsPack/obspack_mod.F90`), the following algorithm is used:

```
! Ensure that units of species are "v/v dry", which is dry=
! air mole fraction. Capture the InUnit value, this is=
! what the units are prior to this call. After we sample=
! the species, we'll call this again requesting that the=
! species are converted back to the InUnit values.=

... THEN DO THE DATA SAMPLING .....
... i.e. determine which GEOS-Chem grid boxes to include in the averaging ...

! Return State_Chm%SPECIES to whatever units they had
! coming into this routine
call Convert_Spc_Units( am_I_root, Input_Opt, State_Met, &
```

The routine `Convert_Spc_Units` performs unit conversions for all of the species in the `State_Chm%Species` array, regardless of whether they are being archived with `ObsPack` or not. This can lead to a bottleneck in performance, as `ObsPack_Sample` is called on every GEOS-Chem “heartbeat” timestep.

What would be more efficient would be to do the unit conversion only for those species that are being archived by `ObsPack`. A typical full-chemistry simulation includes about 200 species. But if we are only using `ObsPack` to archive 10 of these species, GEOS-Chem would execute much faster if we were doing unit conversions for only the 10 archived species instead of all 200 species.

This issue is currently unresolved.

GEOS-CHEM CLASSIC FOLDER TREE

The tables below list the folders in which various components of GEOS-Chem and HEMCO reside.

14.1 GEOS-Chem folder tree

GCClassic/src/GEOS-Chem

Root folder for the GEOS-Chem “science codebase”.

GCClassic/src/GEOS-Chem/GTMM

Contains the GTMM (Global Terrestrial Mercury Model) source code. (NOTE: This option has fallen into disuse.)

GCClassic/src/GEOS-Chem/GeosCore

Contains most GEOS-Chem modules & routines

GCClassic/src/GEOS-Chem/GeosRad

Contains the RRTMG radiative transfer model source code.

GCClassic/src/GEOS-Chem/GeosUtil

Contains GEOS-Chem utility modules & routines (for error handling, string handling, etc.)

GCClassic/src/GEOS-Chem/Headers

Contains modules for with derived-type definitions for state objects, fixed parameter settings, etc.

GCClassic/src/GEOS-Chem/History

Contains modules & routines for archiving GEOS-Chem diagnostics to netCDF-format output.

GCClassic/src/GEOS-Chem/ISORROPIA

Contains the ISORROPIA II source code, which is used for aerosol thermodynamical equilibrium computations.

GCClassic/src/GEOS-Chem/KPP

Main folder for chemical mechanisms built with [KPP-for-GEOS-Chem](#).

GCClassic/src/GEOS-Chem/NcdfUtil

Contains modules & routines for netCDF file I/O.

GCClassic/src/GEOS-Chem/ObsPack

Contains modules & routines for generating GEOS-Chem diagnostic output at the same locations of NOAA ObsPack observational stations.

GCClassic/src/GEOS-Chem/PKUCPL

Contains the coupler code for the PKU 2-way nesting algorithm. (This option has fallen into disuse.)

`GEOS-Chem/Interfaces/GCClassic

Contains the GCClassic driver program (main.F90).

14.2 HEMCO folder tree

GCClassic/src/HEMCO/src/Core

Contains modules for reading, storing, and updating data.

GCClassic/src/HEMCO/src/Extensions

Contains modules for calculating emissions that depend on meteorological variables or parameterizations.

GCClassic/src/HEMCO/src/Interfaces

Contains modules and routines for linking HEMCO to GEOS-Chem Classic and other external models.

GCClassic/src/HEMCO/src/shared

Contains various modules with utility routines (such as for netCDF I/O, regridding, string handling, etc.)

SAMPLE GEOS-CHEM RUN SCRIPTS

Here are some sample run scripts that you can adapt for your own purposes.

15.1 For clusters using the Slurm scheduler

Here is a sample GEOS-Chem run script for computational clusters that use the SLURM scheduler to control jobs:

15.1.1 Run script for Slurm

Save this code to a file named `geoschem.run.slurm`:

```
#!/bin/bash

#SBATCH -c 24
#SBATCH -N 1
#SBATCH -t 0-12:00
#SBATCH -p my-queue_name
#SBATCH --mem=30000
#SBATCH --mail-type=END

#####
### Sample GEOS-Chem run script for SLURM
### You can increase the number of cores with -c and memory with --mem,
### particularly if you are running at very fine resolution (e.g. nested-grid)
#####

# Load your bash-shell customizations
source ~/.bashrc

# Load software modules
# (this example is for GNU 10.2.0 compilers)
source ~/gccclassic.gnu10.env

# Set the proper # of threads for OpenMP
# SLURM_CPUS_PER_TASK ensures this matches the number you set with -c above
export OMP_NUM_THREADS=$SLURM_CPUS_PER_TASK

# NOTE: If the environment file does not max out the available
# stack memory for GEOS-Chem, you may uncomment these lines here:
#ulimit -s unlimited
#export OMP_STACKSIZE=500m
```

(continues on next page)

(continued from previous page)

```
# Run GEOS_Chem. The "time" command will return CPU and wall times.
# Stdout and stderr will be directed to the "GC.log" log file
# (you can change the log file name below if you wish)
srun -c $OMP_NUM_THREADS time -p ./gcclassic >> GC.log
```

Important: If you forget to define `OMP_NUM_THREADS` in your run script, then **GEOS-Chem Classic** will execute using one core. This can cause your simulations to take much longer than is necessary!

Then make `geoschem.run.slurm` executable:

```
$ chmod 755 geoschem.run.slurm
```

For more information about how Slurm is set up on your particular cluster, ask your sysadmin.

15.1.2 Submitting jobs with Slurm

To schedule a **GEOS-Chem Classic** job with Slurm, use this command:

```
$ sbatch geoschem.run.slurm
```

15.2 For Amazon Web Services EC2 cloud instances

When you log into an Amazon Web Services EC2 instance, you will receive an entire node with as many **vCPUs** as you have requested. A vCPU is equivalent to the number of computational cores. Most cloud instances have twice as many vCPUs as physical CPUs (i.e. each CPU chip has 2 cores).

Tip: To find out how many vCPUs are available in your instance, you can use then **nproc** command.

15.2.1 Run script for Amazon EC2

Save the code below to a file named `geoschem.run.aws`:

```
#!/bin/bash

#####
### Sample GEOS-Chem run script for Amazon Web Services EC2 instances
#####

# Load your bash-shell customizations
source ~/.bashrc

### NOTE: We do not have to load an environment file
### because all libraries are contained in the Amazon
### Machine Image (AMI) used to initialize the instance.

# In an AWS cloud instance, you own the entire node, so there is no need
```

(continues on next page)

(continued from previous page)

```
# to use a scheduler like SLURM. You can just use the `nproc` command
# to specify the number of cores that GEOS-Chem should use.
export OMP_NUM_THREADS=$(nproc)

# NOTE: If your ~/.bashrc file does not max out the available
# stack memory for GEOS-Chem, you may uncomment these lines here:
#ulimit -s unlimited
#export OMP_STACKSIZE=500m

# Run GEOS_Chem. The "time" command will return CPU and wall times.
# Stdout and stderr will be directed to the "GC.log" log file
# (you can change the log file name below if you wish)
time -p ./gcclassic >> GC.log 2>&1
```

And then make the `geoschem.run.aws` file executable:

```
$ chmod 755 geoschem.run.aws
```

15.2.2 Running jobs on AWS

When you are on an AWS EC2 instance, you own the entire node, so it is not necessary to use a scheduler. You can run your GEOS-Chem job in with this command:

```
$ ./geoschem.run.aws &
```

This will run your job in the background and send all output (i.e. program output and error output) to `log`.

LOAD SOFTWARE INTO YOUR ENVIRONMENT

This supplemental guide describes the how to load the required software dependencies for **GEOS-Chem** and **HEMCO** into your computational environment.

16.1 On the Amazon Web Services Cloud

All of the required software dependencies for **GEOS-Chem** and **HEMCO** will be included in the Amazon Machine Image (AMI) that you use to initialize your Amazon Elastic Cloud Compute (EC2) instance. For more information, please see our [our GEOS-Chem cloud computing tutorial](#).

16.2 On a shared computer cluster

If you plan to use **GEOS-Chem** or **HEMCO** on a shared computational cluster (e.g. at a university or research institution), then there is a good chance that your IT staff will have already installed several of the required software dependencies.

Depending on your system's setup, there are a few different ways in which you can activate these software packages in your computational environment, as shown below.

16.2.1 1. Check if libraries are available as modules

Many high-performance computing (HPC) clusters use a module manager such as [Lmod](#) or [environment-modules](#) to load software packages and libraries. A module manager allows you to load different compilers and libraries with simple commands.

One downside of using a module manager is that you are locked into using only those compiler and software versions that have already been installed on your system by your sysadmin or IT support staff. But in general, module managers succeed in ensuring that only well-tested compiler/software combinations are made available to users.

Tip: Ask your sysadmin or IT support staff for the software loading instructions specific to your system.

1a. Module load example

The commands shown below are specific to the Harvard Cannon cluster, but serve to demonstrate the process. Note that the names of software packages being loaded may contain version numbers and/or ID strings that serve to differentiate one build from another.

```
$ module load gcc/10.2.0-fasrc01      # gcc / g++ / gfortran
$ module load openmpi/4.1.0-fasrc01   # MPI
$ module load netcdf-c/4.8.0-fasrc01   # netcdf-c
$ module load netcdf-fortran/4.5.3-fasrc01 # netcdf-fortran
$ module load flex/2.6.4-fasrc01      # Flex lexer (needed for KPP)
$ module load cmake/3.25.2-fasrc01    # CMake (needed to compile)
```

Note that it is often not necessary to load all modules. For example, loading **netcdf-fortran** will also cause its dependencies (such as **netcdf-c**, **hdf5**, etc.) to also be loaded into your environment.

Here is a summary of what the above commands do:

module purge

Removes all previously loaded modules

module load git/...

Loads Git (version control system)

module load gcc/...

Loads the GNU Compiler Collection (suite of C, C++, and Fortran compilers)

module load openmpi/...

Loads the OpenMPI library (a dependency of netCDF)

module load netcdf/..

Loads the netCDF library

Important: Depending on how the netCDF libraries have been installed on your system, you might also need to load the netCDF-Fortran library separately, e.g.:

```
module load netcdf-fortran/...
```

module load perl/...

Loads Perl (scripting language)

module load cmake/...

Loads Cmake (needed to compile GEOS-Chem)

module load flex/...

Loads the Flex lexer (needed for [The Kinetic PreProcessor](#)).

16.2.2 2. Check if Spack-built libraries are available

If your system doesn't have a module manager installed, check to see if the required libraries for **GEOS-Chem** and **HEMCO** were *built with the Spack package manager*. Type

```
$ spack find
```

to locate any Spack-built software libraries on your system. If there Spack-built libraries are found, you may present, you may load them into your computational environment with **spack load** commands such as:

```
$ spack load gcc@10.2.0
$ spack load netcdf-c%gcc@10.2.0
$ spack load netcdf-fortran%gcc@10.2.0
... etc ...
```

When loading a Spack-built library, you can specify its version number. For example, **spack load gcc@10.2.0** tells Spack to load the GNU Compiler Collection version 10.2.0.

You may also specify a library by the compiler it was built with. For example, **spack load netcdf-fortran%gcc@10.2.0** tells Spack to load the version of netCDF-Fortran that was built with GNU Compiler Collection version 10.2.0.

These specification methods are often necessary to select a given library in case there are several available builds to choose from.

We recommend that you place **spack load** commands into an [environment file](#).

If a [Spack environment](#) has been installed on your system, type:

```
spack env activate -p ENVIRONMENT-NAME
```

to load all of the libraries in the environment together.

To deactivate the environment, type:

```
spack deactivate
```

16.2.3 3. Check if libraries have been manually installed

If your computer system does not use a module manager and does not use Spack, check for a manual library installation. Very often, common software libraries are installed into standard locations (such as the `/usr/lib` or `/usr/local/lib` system folders). Ask your sysadmin for more information.

Once you know the location of the compiler and netCDF libraries, you can set the proper environment variables for GEOS-Chem and HEMCO.

16.2.4 4. If there are none of these, install them with Spack

If your system has none of the required software packages that **GEOS-Chem** and **HEMCO** need, then we recommend that you *use Spack to build the libraries yourself*. Spack makes the process easy and will make sure that all software dependences are resolved.

Once you have installed the libraries with Spack, you can load the libraries into your computational environment *as described above*.

BUILD REQUIRED SOFTWARE WITH SPACK

This page has instructions for building **dependencies** for [GEOS-Chem Classic](#), [GCHP](#), and [HEMCO](#). These are the **software libraries** that are needed to compile and execute these programs.

Before proceeding, please also check if the dependencies for GEOS-Chem, GCHP, and HEMCO are already found on your computational cluster or cloud environment. If this is the case, you may use the pre-installed versions of these software libraries and won't have to install your own versions.

For more information about software dependencies, see:

- [GEOS-Chem Classic software requirements](#)
- [GCHP software requirements](#)
- [HEMCO software requirements](#)

17.1 Introduction

In the sections below, we will show you how to **build a single software environment containing all software dependencies for GEOS-Chem Classic, GCHP, and HEMCO**. This will be especially of use for those users working on a computational cluster where these dependencies have not yet been installed.

We will be using the [Spack](#) package manager to download and build all required software dependencies for GEOS-Chem Classic, GCHP and HEMCO.

Note: Spack is not the only way to build the dependencies. It is possible to download and compile the source code for each library manually. Spack automates this process, thus it is the recommended method.

You will be using this workflow:

1. *Install Spack and do first-time setup*
2. *Clone a copy of GCClassic, GCHP, or HEMCO*
3. *Install the recommended compiler*
4. *Build GEOS-Chem dependencies and useful tools*
5. *Add spack load commands to your environment file*
6. *Clean up*

17.2 Install Spack and do first-time setup

Decide where you want to install Spack (aka the **Spack root directory**). A few details you should consider are:

- The Spack root directory will be ~5-10 GB. Keep in mind that some computational clusters restrict the size of your home directory (aka `${HOME}`) to a few GB).
- This Spack root directory cannot be moved. Instead, you will have to reinstall Spack to a different directory location (and rebuild all software packages).
- The Spack root directory should be placed in a shared drive if several users need to access it.

Once you have chosen an location for the Spack root directory, you may continue with the Spack download and setup process.

Important: Execute all commands in this tutorial from the same directory. This is typically one directory level higher than the Spack root directory.

For example, if you install Spack as a subdirectory of `${HOME}`, then you will issue all commands from `${HOME}`.

Use the commands listed below to install Spack and perform first-time setup. You can copy-paste these commands, but lookout for lines marked with a `#` (modifiable) ... comment as they might require modification.

```
$ cd ${HOME}                                # (modifiable) cd to the install location_
↪you chose

$ git clone -c feature.manyFiles=true https://github.com/spack/spack.git # download_
↪Spack

$ source spack/share/spack/setup-env.sh    # Load Spack

$ spack external find                      # Tell Spack to look for existing software

$ spack compiler find                     # Tell Spack to look for existing compilers
```

Note: If you should encounter this error:

```
$ spack external find
==> Error: 'name'
```

then Spack could not find any external software on your system.

Spack searches for executables that are located within your search path (i.e. the list of directories contained in your `$PATH` environment variable), but not within software modules. Because of this, you might have to *load a software package into your environment* before Spack can detect it. Ask your sysadmin or IT staff for more information about your system's specific setup.

After the first-time setup has been completed, an environment variable named `SPACK_ROOT`, will be created in your Unix/Linux environment. This contains to the absolute path of the Spack root directory. Use this command to view the value of `SPACK_ROOT`:

```
$ echo ${SPACK_ROOT}
/path/to/home/spack    # Path to Spack root, assumes installation to a subdir of $
↪{HOME}
```

17.3 Clone a copy of GCClassic, GCHP, or HEMCO

The **GCClassic**, **GCHP**, and **HEMCO** repositories each contain a `spack/` subdirectory with customized Spack configuration files `modules.yaml` and `packages.yaml`. We have updated these YAML files with the proper settings in order to ensure a smooth software build process with Spack.

First, define the `model`, `scope_dir`, and `scope_args` environment variables as shown below.

```
$ model=GCClassic           # Use this if you will be working with GEOS-Chem_
↪Classic
$ model=GCHP                # Use this if you will be working with GCHP
$ model=HEMCO               # Use this if you will be working with HEMCO_
↪standalone

$ scope_dir="${model}/spack"  # Folder where customized YAML files are stored

$ scope_args="-C ${scope_dir}" # Tell spack to for custom YAML files in scope_dir
```

You will use these environment variables in the steps below.

When you have completed this step, download the source code for your preferred model (e.g. GEOS-Chem Classic, GCHP, or HEMCO standalone):

```
$ git clone --recurse-submodules https://github.com/geoschem/${model}.git
```

17.4 Install the recommended compiler

Next, install the recommended compiler, **gcc** (aka the GNU Compiler Collection). Use the `scope_args` environment variable that you defined in the *previous step*.

```
$ spack ${scope_args} install gcc      # Install GNU Compiler Collection
```

Note: Requested version numbers for software packages (including the compiler) are listed in the `${scope_dir}/packages.yaml` file. We have selected software package versions that have been proven to work together. You should not have to change any of the settings in `${scope_dir}/packages.yaml`.

As of this writing, the default compiler is **gcc 10.2.0** (includes C, C++, and Fortran compilers). We will upgrade to newer compiler and software package versions as necessary.

The compiler installation should take several minutes (or longer if you have a slow internet connection).

Register the compiler with Spack after it has been installed. This will allow Spack to use this compiler to build other software packages. Use this command:

```
$ spack compiler add $(spack location -i gcc)  # Register GNU Compiler Collection
```

You will then see output similar to this:

```
==> Added 1 new compiler to /path/to/home/.spack/linux/compilers.yaml
gcc@X.Y.Z
==> Compilers are defined in the following files:
    /path/to/home/.spack/linux/compilers.yaml
```

where

- `/path/to/home` indicates the absolute path of your home directory (aka `${HOME}`)
- `X.Y.Z` indicates the version of the GCC compiler that you just built with Spack.

Tip: Use this command to view the list of compilers that have been registered with Spack:

```
$ spack compiler list
```

Use this command to view the installation location for a Spackguide-built software package:

```
$ spack location -i <package-name>
```

17.5 Build GEOS-Chem dependencies and useful tools

Once the compiler has been built and registered, you may proceed to building the software dependencies for GEOS-Chem Classic, GCHP, and HEMCO.

The Spack installation commands that you will use take the form:

```
$ spack ${scope_args} install <package-name>%gcc^openmpi
```

where

- `${scope_args}` is the environment variable that *you defined above*;
- `<package-name>` is a placeholder for the name of the software package that you wish to install;
- `%gcc` tells Spack that it should use the GNU Compiler Collection version that you just built;
- `^openmpi` tells Spack to use OpenMPI when building software packages. You may omit this setting for packages that do not require it.

Spack will download and build `<package-name>` plus all of its dependencies that have not already been installed.

Note: Use this command to find out what other packages will be built along with `<package-name>`:

```
$ spack spec <package-name>
```

This step is not required, but may be useful for informational purposes.

Use the following commands to build dependencies for GEOS-Chem Classic, GCHP, and HEMCO, as well as some useful tools for working with GEOS-Chem data:

1. Build the **esmf** (Earth System Model Framework), **hdf5**, **netcdf-c**, **netcdf-fortran**, and **openmpi** packages:

```
$ spack ${scope_args} install esmf%gcc^openmpi
```

The above command will build all of the above-mentioned packages in a single step.

Note: GEOS-Chem Classic does not require **esmf**. However, we recommend that you build ESMF anyway so that it will already be installed in case you decide to use GCHP in the future.

2. Build the **cdo** (Climate Data Operators) and **nco** (netCDF operators) packages. These are command-line tools for editing and manipulating data contained in netCDF files.

```
$ spack ${scope_args} install cdo%gcc^openmpi
$ spack ${scope_args} install nco%gcc^openmpi
```

3. Build the **ncview** package, which is a quick-and-dirty netCDF file viewer.

```
$ spack ${scope_args} install ncview%gcc^openmpi
```

4. Build the **flex** (Fast Lexical Analyzer) package. This is a dependency of the **Kinetic PreProcessor (KPP)**, with which you can update GEOS-Chem chemical mechanisms.

```
$ spack ${scope_args} install flex%gcc
```

Note: The **flex** package does not use OpenMPI. Therefore, we can omit `^openmpi` from the above command.

At any time, you may see a list of installed packages by using this command:

```
$ spack find
```

17.6 Add spack load commands to your environment file

We recommend “sourcing” the `load_script` that you created in the [previous section](#) from within an **environment file**. This is a file that not only loads the required modules but also defines settings that you need to run GEOS-Chem Classic, GCHP, or the HEMCO standalone.

Please see the following links for sample environment files.

- [Sample GEOS-Chem Classic environment file](#)
- [Sample GCHP environment file](#)
- [Sample HEMCO environment file](#)

Copy and paste the code below into a file named `${model}.env` (using the `${model}` environment variable that you defined above). Then replace any existing `module load` commands with the following code:

```
#####
# Load Spackguide-built modules
#####

# Setup Spack if it hasn't already been done
# ${SPACK_ROOT} will be blank if the setup-env.sh script hasn't been called.
# (modifiable) Replace "/path/to/spack" with the path to your Spack root directory
if [[ "x${SPACK_ROOT}" == "x" ]]; fi
    source /path/to/spack/source/spack/setup-env.sh
fi
```

(continues on next page)

(continued from previous page)

```

# Load esmf, hdf5, netcdf-c, netcdf-fortran, openmpi
spack load esmf%gcc^openmpi

# Load netCDF packages (cdo, nco, ncview)
spack load cdo%gcc^openmpi
spack load nco%gcc^openmpi
spack load ncview

# Load flex
spack load flex

#=====
# Set environment variables for compilers
#=====
export CC=gcc
export CXX=g++
export FC=gfortran
export F77=gfortran

#=====
# Set environment variables for Spack-built modules
#=====

# openmpi (needed for GCHP)
export MPI_ROOT=$(spack-location -i openmpi%gcc)

# esmf (needed for GCHP)
export ESMF_DIR=$(spack location -i esmf%gcc^openmpi)
export ESMF_LIB=${ESMF_DIR}/lib
export ESMF_COMPILER=gfortran
export ESMF_COMM=openmpi
export ESMF_INSTALL_PREFIX=${ESMF_DIR}/INSTALL_gfortran10_openmpi4

# netcdf-c
export NETCDF_HOME=$(spack location -i netcdf-c%gcc^openmpi)
export NETCDF_LIB=$NETCDF_HOME/lib

# netcdf-fortran
export NETCDF_FORTRAN_HOME=$(spack location -i netcdf-fortran%gcc^openmpi)
export NETCDF_FORTRAN_LIB=$NETCDF_FORTRAN_HOME/lib

# flex
export FLEX_HOME=$(spack location -i flex%gcc^openmpi)
export FLEX_LIB=$NETCDF_FORTRAN_HOME/lib
export KPP_FLEX_LIB_DIR=${FLEX_LIB}      # OPTIONAL: Needed for KPP

```

To apply these settings into your login environment, type

```
source ${model}.env # One of GCClassic.env, GCHP.env, HEMCO.env
```

To test if the modules have been loaded properly, type:

```
$ nf-config --help # netcdf-fortran configuration utility
```

If you see a screen similar to this, you know that the modules have been installed properly.

```
Usage: nf-config [OPTION]
```

Available values for OPTION include:

```
--help      display this help message and exit
--all       display all options
--cc        C compiler
--fc        Fortran compiler
--cflags    pre-processor and compiler flags
--fflags    flags needed to compile a Fortran program
--has-dap   whether OPeNDAP is enabled in this build
--has-nc2   whether NetCDF-2 API is enabled
--has-nc4   whether NetCDF-4/HDF-5 is enabled in this build
--has-f90   whether Fortran 90 API is enabled in this build
--has-f03   whether Fortran 2003 API is enabled in this build
--flibs     libraries needed to link a Fortran program
--prefix    Install prefix
--includedir Include directory
--version   Library version
```

17.7 Clean up

At this point, you can remove the `${model}` directory as it is not needed. (Unless you would like to keep it to build the executable for your research with GEOS-Chem Classic, GCHP, or HEMCO.)

The `spack` directory needs to remain. *As mentioned above*, this directory cannot be moved.

You can clean up any Spack temporary build stage information with:

```
$ spack clean -m
==> Removing cached information on repositories
```

That's it!

CUSTOMIZE SIMULATIONS WITH RESEARCH OPTIONS

Most of the time you will want to use the “out-of-the-box” settings in your GEOS-Chem simulations, as these are the recommended settings that have been evaluated with benchmark simulations. But depending on your research needs, you may wish to use alternate simulation options. In this Guide we will show you how you can select these **research options** by editing the various GEOS-Chem and HEMCO configuration files.

18.1 Aerosols

18.1.1 Aerosol microphysics

GEOS-Chem incorporates two different aerosol microphysics schemes: APM (Yu and Luo [[Yu and Luo 2009]]) and TOMAS (Trivitayanurak *et al.* [[Trivitayanurak et al., 2008]]) as compile-time options for the full-chemistry simulation. Both APM and TOMAS are deactivated by default due to the extra computational overhead that these microphysics schemes require.

Follow the steps below to activate either APM or TOMAS microphysics in your full-chemistry simulation.

APM

1. Create a run directory for the Full Chemistry simulation with APM as the extra simulation option.
2. Navigate to the `build` folder within the run directory.
3. Then type the following:

```
$ cmake .. -DAPM=y
$ make -j
$ make install
```

TOMAS

1. Create a run directory for the Full Chemistry simulation with TOMAS as the extra simulation option.
2. Navigate to the `build` folder within the run directory.
3. Then type the following:

```
$ cmake .. -DTOMAS=y -DTOMAS_BINS=15 -DBPCH_DIAG=y
$ make -j
$ make install
```

This will create a GEOS-Chem executable for the TOMAS15 (15 size bins) simulation. To generate an executable for the TOMAS40 (40 size-bins) simulation, replace `-DTOMAS_BINS=15` with `-DTOMAS_BINS=40` in the `cmake` step above.

18.2 Chemistry

18.2.1 Adaptive Rosenbrock solver with mechanism auto-reduction

In Lin *et al.* [[Lin et al., 2023]], the authors introduce an [adaptive Rosenbrock solver with on-the-fly mechanism reduction](#) in The Kinetic PreProcessor (KPP) version 3.0.0 and later. While this adaptive solver is available for all GEOS-Chem simulations that use the `fullchem` simulation, it is disabled by default.

To activate the adaptive Rosenbrock solver with mechanism auto-reduction, edit the line of `geoschem_config.yml` indicated below:

```
chemistry:
  activate: true
  # ... Previous sub-sections omitted
  autoreduce_solver:
    activate: false # <== true activates the adaptive Rosenbrock solver
    use_target_threshold:
      activate: true
      oh_tuning_factor: 0.00005
      no2_tuning_factor: 0.0001
    use_absolute_threshold:
      scale_by_pressure: true
      absolute_threshold: 100.0
    keep_halogens_active: false
    append_in_internal_timestep: false
```

Please see the Lin *et al.* [[Lin et al., 2023]] reference for a detailed explanation of the other adaptive Rosenbrock solver options.

18.2.2 Alternate chemistry mechanisms

GEOS-Chem is compiled “out-of-the-box” with KPP-generated solver code for the `fullchem` mechanism. But you must manually specify the mechanism name at configuration time for the following instances:

Carbon mechanism

Follow these steps to build an executable with the `carbon` mechanism:

1. Create a run directory for the Carbon simulation
2. Navigate to the `build` folder within the run directory.
3. Then type the following:

```
$ cmake .. -DMECH=carbon
$ make -j
$ make install
```

Custom full-chemistry mechanism

We recommend that you use the `custom` mechanism instead of directly modifying the `fullchem` mechanism. The `custom` mechanism is a copy of `fullchem`, but the KPP solver code will be generated in the `KPP/custom` folder instead of in `KPP/fullchem`. This lets you keep the `fullchem` folder untouched.

Follow these steps:

1. Create a run directory for the full-chemistry simulation (whichever configuration you need).
2. Navigate to the `build` folder within the run directory.
3. Then type the following:

```
$ cmake .. -DMECH=custom
$ make -j
$ make install
```

Hg mechanism

Follow these steps to build an executable with the Hg (mercury) mechanism:

1. Create a run directory for the Hg simulation.
2. Navigate to the `build` folder within the run directory.
3. Then type the following:

```
$ cmake .. -DMECH=Hg
$ make -j
$ make install
```

18.2.3 HO₂ heterogeneous chemistry reaction probability

You may update the value of γ_{HO_2} (reaction probability for uptake of HO₂ in heterogeneous chemistry) used in your simulations. Edit the line of `geoschem_config.yml` indicated below:

```
chemistry:
  activate: true
  # ... Preceding sections omitted ...
  gamma_HO2: 0.2 # <=== add new value here
```

18.2.4 TransportTracers

In GEOS-Chem 14.2.0 and later versions, species belonging to the TransportTracers simulation (radionuclides and passive species) now have their properties defined in the `species_database.yml` file. For example:

```
CH3I:
  Background_VV: 1.0e-20
  Formula: CH3I
  FullName: Methyl iodide
  Henry_CR: 3.6e+3
  Henry_K0: 0.20265
  Is_Advected: true
  Is_Gas: true
```

(continues on next page)

(continued from previous page)

```

Is_Photolysis: true
Is_Tracer: true
Snk_Horiz: all
Snk_Mode: efolding
Snk_Period: 5
Snk_Vert: all
Src_Add: true
Src_Mode: HEMCO
MW_g: 141.94

```

where:

- `Is_Tracer: true` indicates a TransportTracer species
- `Snk_*` define species sink properties
- `Src_*` define species source properties
- `Units:` specifies the default units for species (added mainly for age of air species at this time which are in days)

For TransportTracers species that have a source term in HEMCO, there will be corresponding entries in `HEMCO_Config.rc`:

```

--> OCEAN_CH3I          :      true

# ... etc ...

=====
# CH3I emitted over the oceans at rate of 1 molec/cm2/s
=====
(((OCEAN_CH3I
0 SRC_2D_CH3I 1.0 - - - xy molec/cm2/s CH3I 1000 1 1
)))OCEAN_CH3I

```

Sources and sinks for TransportTracers are now applied in the new source code module `GeosCore/tracer_mod.F90`.

Note: Sources and sinks for radionuclide species (Rn, Pb, Be isotopes) are currently not applied in `GeosCore/tracer_mod.F90` (but may be in the future). Emissions for radionuclide species are computed by the HEMCO GC-Rn-Pb-Be extension and chemistry is done in `GeosCore/RnPbBe_mod.F90`.

TransportTracer properties for radionuclide species have been added to `species_database.yml` but are currently commented out.

18.3 Diagnostics

18.3.1 GEOS-Chem and HEMCO diagnostics

Please see our [Diagnostics reference](#) chapter for an overview of how to archive diagnostics from GEOS-Chem and HEMCO.

18.3.2 RRTMG radiative transfer diagnostics

You can use the RRTMG radiative transfer model to archive radiative forcing fluxes to the `GeosRad` History diagnostic collection. RRTMG is implemented as a compile-time option due to the extra computational overhead that it incurs.

To activate RRTMG, follow these steps:

1. Create a run directory for the Full Chemistry simulation, with extra option RRTMG.
2. Navigate to the `build` folder within the run directory.
3. Then type the following:

```
$ cmake .. -DRRTMG=y
$ make -j
$ make install
```

Then also make sure to request the radiative forcing flux diagnostics that you wish to archive in the `HISTORY.rc` file.

18.4 Emissions

18.4.1 Offline vs. online emissions

Emission inventories sometimes include dynamic source types and nonlinear scale factors that have functional dependencies on local environmental variables such as wind speed or temperature, which are best calculated online during execution of the model. HEMCO includes a suite of additional modules (aka [HEMCO extensions](#)) that perform **online emissions** calculations for a variety of sources.

Some types of emissions are highly sensitive to meteorological variables such as wind speed and temperature. Because the meteorological inputs are regridded from their native resolution to the GEOS-Chem or HEMCO simulation grid, emissions computed with fine-resolution meteorology can significantly differ from emissions computed with coarse-resolution meteorology. This can make it difficult to compare the output of GEOS-Chem and HEMCO simulations that use different horizontal resolutions.

In order to provide more consistency in the computed emissions, we now make available for download **offline emissions**. These offline emissions are pre-computed with HEMCO standalone simulations using meteorological inputs at native horizontal resolutions possible. When these emissions are regridded within GEOS-Chem and HEMCO, the total mass emitted will be conserved regardless of the horizontal resolution of the simulation grid.

You should use offline emissions:

- For all GCHP simulations
- For full-chemistry simulations (except benchmark)

You should use online emissions:

- For benchmark simulations
- If you wish to assess the impact of changing/updating the meteorological inputs on emissions.

You may toggle offline emissions on (`true`) or off (`false`) in this section of `HEMCO_Config.rc`:

```
# ----- OFFLINE EMISSIONS -----
# To use online emissions instead set the offline emissions to 'false' and the
# corresponding HEMCO extension to 'on':
# OFFLINE_DUST           - DustDead or DustGinoux
```

(continues on next page)

(continued from previous page)

```

# OFFLINE_BIOGENICVOC - MEGAN
# OFFLINE_SEASALT      - SeaSalt
# OFFLINE_SOILNOX      - SoilNOx
#
# NOTE: When switching between offline and online emissions, make sure to also
# update ExtNr and Cat in HEMCO_Diagn.rc to properly save out emissions for
# any affected species.
#-----
--> OFFLINE_DUST          :      true      # 1980-2019
--> OFFLINE_BIOGENICVOC   :      true      # 1980-2020
--> OFFLINE_SEASALT       :      true      # 1980-2019
--> CalcBrSeasalt         :      true
--> OFFLINE_SOILNOX       :      true      # 1980-2020

```

As stated in the comments, if you switch between offline and online emissions, you will need to activate the corresponding HEMCO extension:

Table 1: Offline emissions and corresponding HEMCO extensions

Offline base emission	Extension #	Corresponding HEMCO extension	Extension #
OFFLINE_DUST	0	DustDead	105
OFFLINE_BIOGENICVOC	0	MEGAN	108
OFFLINE_SEASALT	0	SeaSalt	107
OFFLINE_SOILNOX	0	SoilNOx	104

Example: Disabling offline dust emissions

1. Change the OFFLINE_DUST setting from true to false in HEMCO_Config.rc:

```
--> OFFLINE_DUST          :      false      # 1980-2019
```

2. Change the DustDead extension setting from off to on in HEMCO_Config.rc:

```
105      DustDead          : on      DST1/DST2/DST3/DST4
```

3. Change the extension number for all dust emission diagnostics from 0 (the extension number for base emissions) to 105 (the extension number for DustDead) in HEMCO_Diagn.rc.

```

#####
#### Dust emissions #####
#####
EmisDST1_Total    DST1   -1   -1   -1   2   kg/m2/s  DST1_emission_flux_from_
↳all_sectors
EmisDST1_Anthro   DST1   105    1   -1   2   kg/m2/s  DST1_emission_flux_from_
↳anthropogenic
EmisDST1_Natural  DST1   105    3   -1   2   kg/m2/s  DST1_emission_flux_from_
↳natural_sources
EmisDST2_Natural  DST2   105    3   -1   2   kg/m2/s  DST2_emission_flux_from_
↳natural_sources
EmisDST3_Natural  DST3   105    3   -1   2   kg/m2/s  DST3_emission_flux_from_
↳natural_sources
EmisDST4_Natural  DST4   105    3   -1   2   kg/m2/s  DST4_emission_flux_from_
↳natural_sources

```

To enable online emissions again, do the inverse of the steps listed above.

18.4.2 Sea salt debromination

In Zhu *et al.* [2018], the authors present a mechanistic description of sea salt aerosol debromination. This option was originally enabled by in GEOS-Chem 13.4.0, but was then changed to be an option (disabled by default) due to the impact it had on ozone concentrations.

Further chemistry updates to GEOS-Chem have allowed us to re-activate sea-salt debromination as the default option in GEOS-Chem 14.2.0 and later versions. If you wish to disable sea salt debromination in your simulations, edit the line in `HEMCO_Config.rc` indicated below.

```
107      SeaSalt                : on   SALA/SALC/SALACL/SALCCL/SALAAL/SALCAL/BrSALA/
↪BrSALC/MOPO/MOPI
      # ... Preceding options omitted ...
      --> Model sea salt Br-    :      true    # <== false deactivates sea salt_
↪debromination
      --> Br- mass ratio       :      2.11e-3
```

18.5 Photolysis

18.5.1 Particulate nitrate photolysis

A study by Shah *et al.* [2023] showed that particulate nitrate photolysis increases GEOS-Chem modeled ozone concentrations by up to 5 ppbv in the free troposphere in northern extratropical regions. This helps to correct a low bias with respect to observations.

Particulate nitrate photolysis is turned on by default in GEOS-Chem 14.2.0 and later versions. You may disable this option by editing the line in `geoschem_config.yml` indicated below:

```
photolysis:
  activate: true
  # .. preceding sub-sections omitted ...
  photolyze_nitrate_aerosol:
    activate: true # <=== false deactivates nitrate photolysis
    NITs_Jscale_JHNO3: 100.0
    NIT_Jscale_JHNO2: 100.0
    percent_channel_A_HONO: 66.667
    percent_channel_B_NO2: 33.333
```

You can also edit the other nitrate photolysis parameters by changing the appropriate lines above. See the Shah *et al.* [2023] reference for more information.

18.6 Wet deposition

18.6.1 Luo et al 2020 wetdep parameterization

In Luo *et al.* [[Luo *et al.*, 2020]], the authors introduced an updated wet deposition parameterization, which is now incorporated into GEOS-Chem as a compile-time option. Follow these steps to activate the Luo *et al.* 2020 wetdep scheme in your GEOS-Chem simulations.

1. Create a run directory for the type of simulation that you wish to use.
 - CAVEAT: Make sure your simulation uses at least one species that can be wet-scavenged.
2. Navigate to the `build` folder within the run directory.

3. Then type the following:

```
$ cmake .. -DLUO_WETDEP=y  
$ make -j  
$ make install
```


UNDERSTAND WHAT ERROR MESSAGES MEAN

In this Guide we provide information about the different types of errors that your GEOS-Chem simulation might encounter.

Important: Know the difference between warnings and errors.

Warnings are non-fatal informational messages. Usually you do not have to take any action when encountering a warning. Nevertheless, you should always try to investigate why the warning was generated in the first place.

Errors are fatal and will halt GEOS-Chem compilation or execution. Looking at the error message will give you some clues as to why the error occurred.

We strongly encourage that you try to debug the issue using the info both in this Guide and in our [Debug GEOS-Chem and HEMCO errors](#) Guide. Please see our [Support Guidelines](#) for more information.

19.1 Where does error output get printed?

GEOS-Chem Classic, GCHP, and HEMCO, like all Linux-based programs, send output to two streams: **stdout** and **stderr**.

Most output will go to the **stdout** stream, which takes I/O from the Fortran `WRITE` and `PRINT` commands. If you run e.g. GEOS-Chem Classic by just typing the executable name at the Unix prompt:

```
$ ./gcclassic
```

then the **stdout** stream will be printed to the terminal window. You can also redirect the **stdout** stream to a log file with the redirect command:

```
$ ./gcclassic > GC.log 2>&1
```

The **2>&1** tells the bash script to append the **stderr** stream (noted by 2) to the **stdout** stream (noted by 1). This will make sure that any error output also shows up in the log file.

You can also use the Linux **tee** command, which will send output both to a log file as well as to the terminal window:

```
$ ./gcclassic | tee GC.log 2>&1
```

Note: Please note the following:

1. We have combined HEMCO and GEOS-Chem informational printouts as of GEOS-Chem 14.2.0 and HEMCO 3.7.0. In previous versions, HEMCO informational printouts would have been sent to a separate `HEMCO.log` file.

2. We have disabled most GEOS-Chem and HEMCO informational printouts by default, starting in GEOS-Chem 14.2.0 and HEMCO 3.7.0. These printouts may be restored (e.g. for debugging) by enabling verbose output in both `geoschem_config.yml` and `HEMCO_Config.rc`.
 3. GCHP sends output to several log files as well as to the stdout and stderr streams. Please see gchp.readthedocs.io for more information.
-

19.2 Compile-time errors

In this section we discuss some compilation warnings that you may encounter when building GEOS-Chem.

19.2.1 Cannot open include file netcdf.inc

```
error #5102: Cannot open include file 'netcdf.inc'
```

Problem: The `netcdf-fortran` library cannot be found.

Solution: Make sure that *all software dependencies have been installed and loaded into your Linux environment*.

19.2.2 KPP error: Cannot find -lfl

```
/usr/bin/ld: cannot find -lfl
error: ld returned exit 1 status
```

Problem:: The `Kinetic PreProcessor (KPP)` cannot find the `flex` library, which is one of its dependencies.

Solution: Make sure that *all software dependencies have been installed and loaded into your Linux environment*.

19.2.3 GNU Fortran internal compiler error

```
f951: internal compiler error: in ____ at ____
```

Problem: Compilation halted due to a compiler issue. These types of errors can indicate:

1. An undiagnosed bug in the compiler itself.
2. The inability of the compiler to parse source code adhering to the most recent Fortran language standard.

Solution: Try switching to a newer compiler:

- For GCHP: Use GNU Compiler Collection 9.3 and later.
- For GEOS-Chem Classic and HEMCO: Use GNU Compiler Collection 7.0 and later

19.3 Run-time errors

19.3.1 Floating invalid or floating-point exception error

```
forrtl: error (65): floating invalid      # Error message from Intel Fortran Compiler
Floating point exception (core dumped)  # Error message from GNU Fortran compiler
```

Problem: An illegal floating-point math operation has occurred. This error can be generated if one of the following conditions has been encountered:

1. Division by zero
2. Underflow or overflow
3. Square root of a negative number
4. Logarithm of a negative number
5. Negative or Positive Infinity
6. Undefined value(s) used in an equation

Solution: Re-configure GEOS-Chem (or the HEMCO standalone) with the `-DCMAKE_RELEASE_TYPE=Debug` Cmake option. This will build in additional error checking that should alert you to where the error is occurring. Once you find the location of the error, you can take the appropriate steps, such as making sure that the denominator of an expression never goes to zero, etc.

19.3.2 Forced exit from Rosenbrock

```
Forced exit from Rosenbrock due to the following error:
--> Step size too small: T + 10*H = T or H < Roundoff
T=   3044.21151383269      and H=  1.281206877135470E-012
### INTEGRATE RETURNED ERROR AT:           40           68           1

Forced exit from Rosenbrock due to the following error:
--> Step size too small: T + 10*H = T or H < Roundoff
T=   3044.21151383269      and H=  1.281206877135470E-012
### INTEGRATE FAILED TWICE ###

#####
### KPP DEBUG OUTPUT
### Species concentrations at problem box           40           68           1
#####
... printout of species concentrations ...

#####
### KPP DEBUG OUTPUT
### Species concentrations at problem box           40           68           1
#####
... printout of reaction rates ...
```

Problem: The KPP Rosenbrock integrator could not converge to a solution at a particular grid box. This can happen when:

1. The absolute (ATOL) and/or relative (RTOL) *error tolerances* need to be refined.
2. A particular species has numerically underflowed or overflowed.

3. A division by zero occurred in the reaction rate computations.
4. A species has been set to a very low value in another operation (e.g. wet scavenging), thus causing the non-convergence.
5. The initial conditions of the simulation may be non-physical.
6. A data file (meteorology or emissions) may be corrupted.

If the non-convergence only happens once, then GEOS-Chem will revert to prior concentrations and reset the saved KPP internal timestep (H_{new}) to zero before calling the Rosenbrock integrator again. In many instances, this is sufficient for the chemistry to converge to a solution.

In the case that the Rosenbrock integrator fails to converge to a solution twice in a row, all of the concentrations and reaction rates at the grid box will be printed to *stdout* and the simulation will terminate.

Solution: Look at the error printout. You will likely notice species concentrations or reaction rates that are extremely high or low compared to the others. This will give you a clue as to where in GEOS-Chem the error may have occurred.

Try performing some short test simulations, turning each operation (e.g. transport, PBL mixing, convection, etc.) off one at a time. This should isolate the location of the error. Make sure to turn on verbose output in both `geoschem_config.yml` and `HEMCO_Config.rc`; this will send additional printout to the *stdout* stream. The clue to finding the error may become obvious by looking at this output.

Check your restart file to make sure that the initial concentrations make sense. For certain simulations, using initial conditions from a simulation that has been sufficiently spun-up makes a difference.

Use a netCDF file viewer like **ncview** to open the meteorology files on the day that the error occurred. If a file does not open properly, it is probably corrupted. If you suspect that the file may have been corrupted during download, then download the file again from its original source. If this still does not fix the error, then the file may have been corrupted at its source. Please open a new Github issue to alert the GEOS-Chem Support Team.

More about KPP error tolerances

The error tolerances are set in the following locations:

1. **fullchem** mechanism: In routine `Do_FlexChem` (located in `in GeosCore/fullchem_mod.F90`).
2. **Hg** mechanism: In routine `ChemMercury` (located in `in GeosCore/mercury_mod.F90`).

For example, in the fullchem mechanism, ATOL and RTOL are defined as:

```
!%%%% CONVERGENCE CRITERIA %%%%  
  
! Absolute tolerance  
ATOL      = 1e-2_dp  
  
! Relative tolerance  
! Changed to 0.5e-3 to avoid integrate errors by halogen chemistry  
! -- Becky Alexander & Bob Yantosca (24 Jan 2023)  
RTOL      = 0.5e-3_dp
```

Convergence errors can occur because the system arrives to a state too far from the truth to be able to converge. By tightening (i.e. decreasing) the tolerances, you ensure that the system stays closer to the truth at every time step. Then, the problematic time steps will start the chemistry with a system closer to the true state, enabling the chemistry to converge.

CAVEAT: If the first time step of chemistry cannot converge, tightening the tolerances wouldn't work but loosening the tolerance would. So you might have to experiment a little bit in order to find the proper settings for ATOL and RTOL for your specific mechanism.

19.3.3 HEMCO Error: Cannot find field

```
HEMCO Error: Cannot find field _____. Please check the name in the config file.
```

Problem: A GEOS-Chem Classic or HEMCO standalone simulation halts because HEMCO cannot find a certain input field.

Solution: Most of the time, this error indicates that a species is missing from the [GEOS-Chem restart file](#). By default, the GEOS-Chem restart file (entry `SPC_` in `HEMCO_Config.rc`) uses time cycle flag `EFY0`. This setting tells HEMCO to halt if a species does not have an initial condition field contained in the GEOS-Chem restart file. Changing this time cycle flag to `CYS` will allow the simulation to proceed. In this case, species will be given a default background initial concentration, and the simulation will be allowed to proceed.

19.3.4 HEMCO Error: Cannot find file for current simulation time

```
HEMCO ERROR: Cannot find file for current simulation time:
./Restarts/GEOSChem.Restart.17120701_0000z.nc4 - Cannot get field SPC_NO.
Please check file name and time (incl. time range flag) in the config. file
```

Problem: HEMCO tried to read data from a file but could not find the time slice requested in `HEMCO_Config.rc`.

Solution: Make sure that the file is at the path specified in `HEMCO_Config.rc`. HEMCO will try to look back in time starting with the current year and going all the way back to the year 1712 or 1713. So if you see 1712 or 1713 in the error message, that is a tip-off that the file is missing.

19.3.5 HEMCO Run Error

```
=====
GEOS-CHEM ERROR: HCO_RUN

HEMCO ERROR: Please check the HEMCO log file for error messages!

STOP at HCOI_GC_RUN (hcoi_gc_main_mod.F90)
=====
```

Problem: A GEOS-Chem simulation stopped in the `HCOI_GC_RUN` routine with an error message similar to that shown above.

Solution: Look at the output that was written to the *stdout* and *stderr* streams. Error messages containing `HCO` originate in HEMCO.

19.3.6 HEMCO time stamps may be wrong

```
HEMCO WARNING: ncdf reference year is prior to 1901 - time stamps may be wrong!
--> LOCATION: GET_TIMEIDX (hco_read_std_mod.F90)
```

Problem: HEMCO reads the files but gives zero emissions and shows the error listed above.

Solution: Do the following:

1. Reset the reference datetime in the netCDF file so that it is after 1901.
2. Make sure that the `time:calendar` string is either `standard` or `gregorian`. GEOS-Chem Classic, GCHP, and HEMCO can only read data placed on calendars with leap years.

GCST member [Lizzie Lundgren](#) writes:

This HEMCO error occurs if the reference time for the netCDF file time dimension is prior to 1901. If you do `ncdump -c filename` you will be able to see the metadata for the time dimension as well as the time variable values. The time units should include the reference date.

You can get around this issue by changing the reference time within the file. You can do this with `cdo` (Climate Data Operators) using the `setreftime` command.

Here is a bash script example by GCST member [Melissa Sulprizio](#) that updates the calendar and reference time for all files ending in `*.nc` within a directory. This script was made for a user who ran into this issue. In that case the first file was for Jan 1, 1950, so that was made the new reference time. I would recommend doing the same for your dataset so that the first time variable value would be 0. This script also compresses the file which we recommend doing.

```
#!/bin/bash

for file in *.nc; do
    echo "Processing $file"

    # Make sure the calendar is "standard" and not e.g. 360 days
    cdo setcalendar,standard $file tmp.nc
    mv tmp.nc $file

    # Set file reference time to 1950-01-01 at 0z
    cdo setreftime,1950-01-01,0 $file tmp.nc
    mv tmp.nc $file

    # Compress the file
    nccopy -d1 -c "time/1" $file tmp.nc
    mv tmp.nc $file
done
```

After you update the file you can then again do `ncdump -c filename` to check the time dimension. For the case above it looks like this after processing.

```
double time(time) ;
    time:standard_name = "time" ;
    time:long_name = "time" ;
    time:bounds = "time_bnds" ;
    time:units = "days since 1950-01-01 00:00:00" ;
    time:calendar = "standard" ;
    . . .

time = 0, 31, 59, 90, 120, 151, 181, 212, 243, 273, 304, 334, 365, 396, 424,
      455, 485, 516, 546, 577, 608, 638, 669, 699, 730, 761, 790, 821, 851, ``
      882, 912, 943, 974, 1004, 1035, 1065, 1096, 1127, 1155, 1186, 1216,
↪1247 . . .
```

19.3.7 Negative tracer found in WETDEP

```

WETDEP: ERROR at 40 67 1 for species 2 in area WASHOUT: at surface
LS      : T
PDOWN   : 0.0000000000000000
QQ      : 0.0000000000000000
ALPHA   : 0.0000000000000000
ALPHA2  : 0.0000000000000000
RAINFrac : 0.0000000000000000
WASHFRAC : 0.0000000000000000
MASS_WASH : 0.0000000000000000
MASS_NOWASH : 0.0000000000000000
WETLOSS  : NaN
GAINED   : 0.0000000000000000
LOST     : 0.0000000000000000
DSpc(NW,:) : NaN 6.0358243778561746E-013 6.
↪5871997362336500E-013 7.2710915872550685E-013 8.0185772698102585E-013 8.
↪7883682997147595E-013 9.6396466805517407E-013 1.0574719517340253E-012 1.
↪1617302070198606E-012 1.2976219851862141E-012 1.4347568254382824E-012 1.
↪5772212240871896E-012 1.7071657565802178E-012 1.8443377617027378E-012 1.
↪9982208320328261E-012 2.1567932874822908E-012 2.2591568422224307E-012 2.
↪2208301198704935E-012 1.8475974519883714E-012 1.7716069173018996E-013 1.
↪7714395985520433E-013 1.7633649101242403E-013 1.6668529114369137E-013 1.
↪3548045738669223E-013 5.1061710020314286E-014 0.0000000000000000 0.
↪0000000000000000 0.0000000000000000 0.0000000000000000 0.
↪0000000000000000 0.0000000000000000 0.0000000000000000 0.
↪0000000000000000 0.0000000000000000 0.0000000000000000 0.
↪0000000000000000 0.0000000000000000 0.0000000000000000 0.
↪0000000000000000 0.0000000000000000 0.0000000000000000 0.
↪0000000000000000 0.0000000000000000 0.0000000000000000 0.
↪0000000000000000 0.0000000000000000 0.0000000000000000 0.
↪0000000000000000 0.0000000000000000 0.0000000000000000 0.
Spc(I,J,:N) : NaN 3.5108056785061143E-009 3.
↪8363969256742307E-009 3.6615166033026556E-009 3.6780394914242783E-009 4.
↪1462343168230006E-009 4.7319942271993657E-009 5.1961472823088513E-009 5.
↪4030830279477525E-009 5.5736845790195336E-009 5.7139596145766606E-009 5.
↪8629212873139874E-009 7.9742789235773213E-009 1.0334311421916619E-008 1.
↪0816150360971255E-008 1.1168715310744298E-008 1.1534959217017146E-008 1.
↪1809950282570185E-008 1.7969626885629474E-008 1.7430760762446019E-008 1.
↪7477810715818748E-008 1.7967321756900857E-008 1.8683742574601477E-008 1.
↪9309929368816065E-008 2.0262386892450682E-008 2.0489969814921647E-008 1.
↪9961590106306151E-008 2.2859284477873924E-008 1.3161046290246557E-008 6.
↪5857053651000387E-009 2.7535806161296159E-009 1.2708780077337107E-009 3.
↪6557775667039418E-010 6.1984105316417057E-011 2.6665694620973736E-011 8.
↪7599157145440813E-012 4.8009375158768866E-012 1.0086435318729046E-012 1.
↪3493529625353547E-013 1.6403790023674963E-014 2.7417226109948757E-015 4.
↪2031825835582592E-014 2.3778709382809943E-013 8.3223532851684382E-013 4.
↪5695049346098890E-012 6.9911523125704209E-012 2.5076669266356582E-012
=====
GEOS-Chem ERROR: Error encountered in wet deposition!
-> at SAFETY (in module GeosCore/wetscav_mod.F90)
=====
GEOS-Chem ERROR: Error encountered in "Safety"!
-> at Do_Washout_at_Sfc (in module GeosCore/wetscav_mod.F90)
=====

```

(continues on next page)

(continued from previous page)

```
=====
GEOS-Chem ERROR:
-> at WetDep (in module GeosCore/wetscav_mod.F90)
=====

=====
GEOS-Chem ERROR: Error encountered in "Wetdep"!
-> at Do_WetDep (in module GeosCore/wetscav_mod.F90)
=====

=====
GEOS-CHEM ERROR: Error encountered in "Do_WetDep"!
STOP at -> at GEOS-Chem (in GeosCore/main.F90)
=====

- CLEANUP: deallocating arrays now...
```

Problem: A GEOS-Chem simulation has encountered either negative or NaN (not-a-number) concentrations in the wet deposition module. This can indicate the following:

1. The wet deposition routines have removed too much soluble species from within a grid box.
2. Another operation (e.g. transport, convection, etc.) has removed too much soluble species from within a grid box.
3. A corrupted or incorrect meteorological input has caused too much rainout or washout to occur within a grid box (which leads to conditions 1 and/or 2 above).
4. An *array-out-of-bounds error* has corrupted a variable that is used in wet deposition.
5. For nested-grid simulations, the transport timestep may be too large, thus resulting in grid boxes with zero or negative concentrations.

Solution: Re-configure GEOS-Chem and/or HEMCO with the `-DCMAKE_RELEASE_TYPE=Debug` CMake option. This adds in additional error checks that may help you find where the error occurs.

Also try adding some `PRINT*` statements before and after the call to `DO_WETDEP` to check the concentrations entering and leaving the wetdep module. That might give you an idea of where the concentrations are going negative.

19.3.8 Permission denied error

```
geoschem.run: Permission denied
```

Problem: The script `geoschem.run` is not executable.

Solution: Change the permission of the script with:

```
$ chmod 755 geoschem.run
```


19.3.9 Excessive fall velocity error

```
GEOS-CHEM ERROR: Excessive fall velocity?
STOP at CALC_FALLVEL, UCX_mod
```

Problem: The fall velocity (in stratospheric chemistry routine `Calc_FallVel` in module `GeosCore/ucx_mod.F90`) exceeds 10 m/s. This error will most often occur in GEOS-Chem Classic nested-grid simulations.

Solution: Reduce the default timestep settings in `geoschem_config.yml`. You may need to use 300 seconds (transport) and 600 seconds (chemistry) or even smaller depending on the horizontal resolution of your simulation.

19.4 File I/O errors

19.4.1 List-directed I/O syntax error

```
# Error message from GNU Fortran
At line NNNN of file filename.F90
Fortran runtime error: Bad real number|integer number|character in item X of list_
↪input

# Error message from Intel Fortran
forrtl: severe (59): list-directed I/O syntax error, unit -5, file Internal List-
↪Directed Read
```

Problem: This error indicates that the wrong type of data was read from a text file. This can happen when:

1. Numeric input is expected but character input was read from disk (or vice-versa);
2. A **READ** statement in your code has been omitted or deleted.

Solution: Check configuration files (`geoschem_config.yml`, `HEMCO_Config.rc`, `HEMCO_Diagn.rc`, etc.) for syntax errors and omissions that could be causing this error.

19.4.2 Nf_Def_Var: can not define variable

```
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!

Nf_Def_var: can not define variable: ____

!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!

Code stopped from DO_ERR_OUT (in module NcdfUtil/m_do_err_out.F90)

This is an error that was encountered in one of the netCDF I/O modules,
which indicates an error in writing to or reading from a netCDF file!

!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
```

Problem: GEOS-Chem or HEMCO could not write a variable to a netCDF file. This error may be caused by:

1. The netCDF file is write-protected and cannot be overwritten.
2. The path to the netCDF file is incorrect (e.g. directory does not exist).
3. The netCDF file already contains a variable with the same name.

Solution: Try the following:

1. If GEOS-Chem or HEMCO will be overwriting any existing netCDF files (which can often happen during testing & development), make sure that the file and containing directory are not write-protected.
2. Make sure that the path where you intend to write the netCDF file exists.
3. Check your `HISTORY.rc` and `HEMCO_Diagn.rc` diagnostic configuration files to make sure that you are not writing more than one diagnostic variable with the same name.

19.4.3 NetCDF: HDF Error

```
NetCDF: HDF error
```

Problem: The netCDF library routines in GEOS-Chem or HEMCO cannot read a netCDF file. The error is occurring in the HDF5 library (upon which netCDF depends). This may indicate a corrupted or incomplete netCDF file.

Solution: Try re-downloading the file from the [WashU data portal](#). Downloading a fresh copy of the file is often sufficient to fix this type of issue. If the error persists, please open a new GitHub issue to alert the GEOS-Chem Support team, as the corruption may have occurred at the original source of the data.

19.5 Segmentation faults and similar errors

```
SIGSEGV, segmentation fault occurred
```

Problem: GEOS-Chem or HEMCO tried to access an [invalid memory location](#).

Solution: See the sections below for ways to debug segmentation fault errors.

19.5.1 Array-out-of-bounds error

```
Subscript #N of the array THISARRAY has value X which is less than the lower bound of
↳Y

or

Subscript #N of the array THISARRAY has value A which is greater than the upper bound
↳of B
```

Problem: An array index variable refers to an element that lies outside of the array boundaries.

Solution: Reconfigure GEOS-Chem with the following options:

```
$ cd /path/to/build                # Your GEOS-Chem or HEMCO build directory
$ cmake . -DCMAKE_BUILD_TYPE=Debug
```

This will enable several debugging options, including checking for array operations indices that going out of bounds. You will get an error message similar to those shown above.

Use the **grep** command to search for all instances of the array (in this example, `THISARRAY`) in each source code folder:

```
grep -i THISARRAY *.F90    # -i means ignore uppercase/lowercase distinction
```

This should let you quickly locate the issue. Depending on the compiler that is used, you might also get a routine name and line number from the error output.

19.5.2 Segmentation fault encountered after TPCORE initialization

```
NASA-GSFC Tracer Transport Module successfully initialized
```

Problem: A GEOS-Chem simulation dies right after you see this text.

Note: Starting in GEOS-Chem Classic 14.1.0, the text above will only be printed if you have activated verbose output in the `geoschem_config.yml` configuration file.

Solution: Increase the amount of stack memory available to GEOS-Chem and HEMCO. [Please follow this link](#) for detailed instructions.

19.5.3 Invalid memory access

```
severe (174): SIGSEGV, segmentation fault occurred
This message indicates that the program attempted an invalid memory reference.
Check the program for possible errors.
```

Problem: GEOS-Chem or HEMCO code tried to read data from an invalid memory location. This can happen when data is being read from a file into an array, but the array is too small to hold all the data.

Solution: Use a debugger (like **gdb**) to try to diagnose the situation. Also try increasing the dimensions of the array that you suspect might be too small.

19.5.4 Stack overflow

```
severe (174): SIGSEGV, possible program stack overflow occurred
Program requirements exceed current stacksize resource limit.
```

Problem: GEOS-Chem and/or HEMCO is using more **stack memory** than is currently available to the system. Stack memory is a reserved portion of the memory structure where short-lived variables are stored, such as:

1. Variables that are local to a given subroutine
2. Variables that are NOT globally saved
3. Variables that are NOT declared as an `ALLOCATABLE` array
4. Variables that are NOT declared as a `POINTER` variable or array
5. Variables that are included in an `!$OMP PRIVATE` or `!$OMP THREADPRIVATE`

Solution: Max out the amount of stack memory that is available to GEOS-Chem and HEMCO. [See this section](#) for instructions.

19.6 Less common errors

The errors listed below, which occur infrequently, are related to invalid memory operations. These can especially occur with `POINTER`-based variables.

19.6.1 Bus Error

Problem: GEOS-Chem or HEMCO is trying to reference memory that cannot possibly be there. The website Stack-Overflow.com has a [definition of bus error and how it differs from a segmentation fault](#).

Solution: A bus error may occur when you call a subroutine with too many arguments. Check subroutine definitions and subroutine calls to make sure the correct number of arguments are passed.

19.6.2 Double free or corruption

```
*** glibc detected *** PROGRAM_NAME: double free or corruption (out): ____ ***
```

Problem: The following error is not common, but can occur under some circumstances. Usually this means one of the following has occurred:

1. You are deallocating the same variable more than once.
2. You are deallocating a variable that wasn't allocated, or that has already been deallocated.

Please see [this link](#) for more details.

Solution: Try setting all deleted pointers to `NULL()`.

You can also use a debugger like **gdb**, which will show you a backtrace from your crash. This will contain information about in which routine and line number the code crashed, and what other routines were called before the crash happened.

Remember these three basic rules when working with `POINTER`-based variables:

1. Set pointer to `NULL` after free.
2. Check for `NULL` before freeing.
3. Initialize pointer to `NULL` in the start.

Using these rules helps to prevent this type of error.

Also note, you may see this error when a software library required by GEOS-Chem and/or HEMCO is not (e.g. **netcdf** or **netcdf-fortran** has not been installed. GEOS-Chem and/or HEMCO may be making calls to the missing library, which results in the error. If this is the case, the solution would be to [install all required libraries](#).

19.6.3 Dwarf subprogram entry error

```
Dwarf subprogram entry L_ROUTINE-NAME__LINE-NUMBER__par_loop2_2_576 has high_pc < low_
↳pc.
This warning will not be repeated for other occurrences.
```

Problem: GEOS-Chem or HEMCO code tried to use a `POINTER`-based variable that is **unassociated** (i.e. not pointing to any other variable or memory) from within an OpenMP parallel loop.

This error can happen when a `POINTER`-based variable is set to `NULL()` where it is declared:

```
TYPE(Species), POINTER :: ThisSpc => NULL()
```

The above declaration causes use pointer variable `ThisSpc` to be implicitly declared with the `SAVE` attribute. This causes a segmentation fault, because all pointers used within an OpenMP parallel region must be associated and nullified on the same thread.

Solution: Make sure that any `POINTER`-based variables (such as `ThisSpc` in this example) point to their target and are nullified within the same OpenMP parallel loop.

```
TYPE(Species), POINTER :: ThisSpc    ! Do not set to NULL() here!!!

... etc ...

!$OMP PARALLEL DO(
!$OMP DEFAULT( SHARED ) &
!$OMP PRIVATE( I, J, L, N, ThisSpc, ... )
DO N = 1, nSpecies
DO L = 1, NZ
DO J = 1, NY
DO I = 1, NX

    ... etc ...

    ! Point to species database entry
    ThisSpc => State_Chm%Species(N)%Info

    ... etc ...

    ! Free pointer at end of loop
    ThisSpc => NULL()

ENDDO
ENDDO
ENDDO
ENDDO
```

Note that you must also add `POINTER`-based variables (such as `ThisSpc`) to the `!$OMP PRIVATE` clause for the parallel loop.

For more information about this type of error, [please see this article](#).

19.6.4 Free: invalid size

```
Error in PROGRAM_NAME free(): invalid size: 0x00000000 0662e090
```

Problem: This error is not common. It can happen when:

1. You are trying to free a pointer that wasn't allocated.
2. You are trying to delete an object that wasn't created.
3. You may be trying to nullify or deallocate an object more than once.
4. You may be overflowing a buffer.
5. You may be writing to memory that you shouldn't be writing to.

Solution: Any number of programming errors can cause this problem. You need to use a debugger (such as `gdb`), get a backtrace, and see what your program is doing when the error occurs. If that fails and you determine you have

corrupted the memory at some previous point in time, you may be in for some painful debugging (it may not be too painful if the project is small enough that you can tackle it piece by piece).

See [this post on StackOverflow](#) for more information.

19.6.5 Munmap_chunk: invalid pointer

```
** glibc detected *** PROGRAM_NAME: munmap_chunk(): invalid pointer:
↳0x0000000059aac30 ***
```

Problem: This is not a common error, but can happen if you deallocate or nullify a `POINTER`-based variable that has already been deallocated or modified.

Solution: Use a debugger (like `gdb`) to see where in GEOS-Chem or HEMCO the error occurs. You will likely have to remove a duplicate `DEALLOCATE` or `=> NULL()` statement. See [this link](#) for more information.

19.6.6 Out of memory asking for NNNNN

```
Fatal compilation error: Out of memory asking for 36864.
```

Problem: This error may be caused by the `datasize` limit not being maxed out in your Linux login environment. For more informatin, see [this link](#) for more information.

Solution: Use this command to check the status of the `datasize` limit:

```
$ ulimit -d
unlimited
```

If the result of this command is not `unlimited`, then set it to `unlimited` with this command:

```
$ ulimit -d unlimited
```

Note: The two most important limits for GEOS-Chem and HEMCO are `datasize` and `stacksize`. These should both be set to `unlimited`.

DEBUG GEOS-CHEM AND HEMCO ERRORS

If your **GEOS-Chem** or **HEMCO** simulation dies unexpectedly with an error or takes much longer to execute than it should, the most important thing is to try to isolate the source of the error or bottleneck right away. Below are some debugging tips that you can use.

20.1 Check if a solution has been posted to Github

We have migrated support requests from the [GEOS-Chem wiki](#) to **Github issues**. A quick search of Github issues (both open and closed) might reveal the answer to your question or provide a solution to your problem.

You should also feel free to open a new issue at one of these Github links:

- [GEOS-Chem Classic new issues page](#)
- [GCHP new issues page](#)
- [HEMCO new issues page](#)

If you are new to Github, we recommend viewing our Github tutorial videos at [our GEOS-Chem Youtube site](#).

20.2 Check if your computational environment is configured properly

Many **GEOS-Chem** and **HEMCO** errors occur due to improper configuration settings (i.e. missing libraries, incorrectly-specified environment variables, etc.) in your computational environment. Take a moment and refer back to these manual pages (on ReadTheDocs) for information on configuring your environment:

- [GEOS-Chem Classic manual](#)
- [GCHP manual](#)
- [HEMCO manual](#)

20.3 Check any code modifications that you have added

If you have made modifications to a “fresh out-of-the-box” **GEOS-Chem** or **HEMCO** version, look over your code edits to search for sources of potential error.

You can also use Git to revert to the last stable version, which is always in the **main** branch.

20.4 Check if your runs exceeded time or memory limits

If you are running **GEOS-Chem** or **HEMCO** on a shared computer system, you will probably have to use a **job scheduler** (such as **SLURM**) to submit your jobs to a computational queue. You should be aware of the run time and memory limits for each of the queues on your system.

If your job uses more memory or run time than the computational queue allows, it can be cancelled by the scheduler. You will usually get an error message printed out to the stderr stream, and maybe also an email stating that the run was terminated. Be sure to check all of the log files created by your jobs for such error messages.

To solve this issue, try submitting your **GEOS-Chem** or **HEMCO** simulations to a queue with larger run-time and memory limits. You can also try splitting up your long simulations into several smaller stages (e.g. monthly) that take less time to run to completion.

20.5 Send debug printout to the log files

If your **GEOS-Chem** simulation stopped with an error, but you cannot tell where, turn on the `debug_printout` option. This is found in the **Simulation Settings** section of `geoschem_config.yml`:

```
#####
# Simulation settings
#####
simulation:
  name: fullchem
  start_date: [20190701, 000000]
  end_date: [20190801, 000000]
  root_data_dir: /path/to/ExtData
  met_field: MERRA2
  species_database_file: ./species_database.yml
  debug_printout: false # <---- set this to true
  use_gcclassic_timers: false
```

This will send additional output to the **GEOS-Chem** log file, which may help you to determine where the simulation stopped.

If your **HEMCO** simulation stopped with an error, turn on debug printout by editing the `Verbose` and `Warnings` settings at the top of the `HEMCO_Config.rc` configuration file:

```
#####
## BEGIN SECTION SETTINGS
#####

ROOT:                               /path/to/ExtData/HEMCO
METDIR:                             MERRA2
GCAP2SCENARIO:                      none
GCAP2VERTRES:                       none
Logfile:                            HEMCO.log
DiagnFile:                          HEMCO_Diagn.rc
DiagnPrefix:                        ./OutputDir/HEMCO_diagnostics
DiagnFreq:                          Monthly
Wildcard:                           *
Separator:                          /
Unit tolerance:                     1
Negative values:                    0
Only unitless scale factors: false
```

(continues on next page)

(continued from previous page)

```

Verbose:          0      # <---- set this to 3
Warnings:         1      # <---- set this to 3

```

Both `Verbose` and `Warnings` settings can have values from 0 to 3. The higher the number, the more information will be printed out to the `HEMCO.log` file. A value of 0 disables debug printout.

Having this extra debug printout in your log file output may provide insight as to where your simulation is halting.

20.6 Look at the traceback output

An **error traceback** will be printed out whenever a **GEOS-Chem** or **HEMCO** simulation halts with an error. This is a list of routines that were called when the error occurred.

An sample error traceback is shown here:

```

forrtl: severe (174): SIGSEGV, segmentation fault occurred

Image                PC                Routine                Line                Source
gcclassic             0000000000C82023   Unknown              Unknown             Unknown
libpthread-2.17.s     00002AACE8015630   Unknown              Unknown             Unknown
gcclassic             000000000095935E   error_mod_mp_erro    437                error_mod.F90
gcclassic             000000000040ABB7   MAIN__               422                main.F90
gcclassic             0000000000406B92   Unknown              Unknown             Unknown
libc-2.17.so          00002AACE8244555   __libc_start_main    Unknown             Unknown
gcclassic             0000000000406AA9   Unknown              Unknown             Unknown

```

The top line with a valid routine name and line number printed is the routine that exited with an error (`error_mod.F90`, line 437). You might also have to look at the other listed files as well to get some more information about the error (e.g. `main.F90`, line 422).

20.7 Identify whether the error happens consistently

If your **GEOS-Chem** or **HEMCO** error always happens at the same model date and time, this could indicate corrupted meteorology or emissions input data files. In this case, you may be able to fix the issue simply by re-downloading the files to your disk space.

If the error happened only once, it could be caused by a network problem or other such transient condition.

20.8 Isolate the error to a particular operation

If you are not sure where a **GEOS-Chem** error is occurring, turn off operations (such as transport, chemistry, dry deposition, etc.) one at a time in the `geoschem_config.yml` configuration file, and rerun your simulation.

Similarly, if you are debugging a **HEMCO** error, turn off different emissions inventories and extensions one at a time in the `HEMCO_Config.rc` file, and rerun your simulation.

Repeating this process should eventually lead you to the source of the error.

20.9 Compile with debugging options

You can compile **GEOS-Chem** or **HEMCO** in debug mode. This will activate several additional error run-time error checks (such as looking for assignments that go outside of array bounds or floating point math errors) that can give you more insight as to where your simulation is dying.

Configure your code for debug mode with the `-DCMAKE_RELEASE_TYPE=Debug` option. From your run directory, type these commands:

```
cd build
cmake ../CodeDir -DCMAKE_RELEASE_TYPE=Debug -DRUNDIR=..
make -j
make -j install
cd ..
```

Attention: Compiling in debug mode will add a significant amount of computational overhead to your simulation. Therefore, we recommend to activate these additional error checks only in short simulations and not in long production runs.

20.10 Use a debugger

You can save yourself a lot of time and hassle by using a debugger such as **gdb** (the GNU debugger). With a debugger you can:

- Examine data when a program stops
- Navigate the stack when a program stops
- Set break points

To run **GEOS-Chem** or **HEMCO** in the **gdb** debugger, you should first *compile in debug mode*. This will turn on the `-g` compiler flag (which tells the compiler to generate symbolic information for debugging) and the `-O0` compiler flag (which shuts off all optimizations). Once the executable has been created, type one of the following commands, which will start **gdb**:

```
$ gdb gcclassic      # for GEOS-Chem Classic
$ gdb gchp           # for GCHP
$ gdb hemco          # for HEMCO standalone
```

At the **gdb** prompt, type one of these commands:

```
(gdb) run                # for GEOS-Chem Classic or GCHP
(gdb) run HEMCO_sa_Config.rc # for HEMCO standalone
```

With **gdb**, you can also go directly to the point of the error without having to re-run **GEOS-Chem** or **HEMCO**. When your **GEOS-Chem** or **HEMCO** simulation dies, it will create a **corefile** such as `core.12345`. The 12345 refers to the process ID assigned to your executable by the operating system; this number is different for each running process on your system.

Typing one of these commands:

```
$ gdb gcclassic core.12345      # for GEOS-Chem Classic
$ gdb gchp core.12345          # for GCHP
$ gdb hemco_standalone core.12345 # for HEMCO standalone
```

will open **gdb** and bring you immediately to the point of the error. If you then type at the (gdb) prompt:

```
(gdb) where
```

You will get a *traceback* listing.

To exit **gdb**, type `quit`.

20.11 Print it out if you are in doubt!

Add `print*` statements to write values of variables in the area of the code where you suspect the error is occurring. Also add the `call flush(6)` statement to flush the output to the screen and/or log file immediately after printing. Maybe you will see something wrong in the output.

You can often detect numerical errors by adding debugging print statements into your source code:

1. Use `MINVAL` and `MAXVAL` functions to get the minimum and maximum values of an array:

```
PRINT*, '### Min, Max: ', MINVAL( ARRAY ), MAXVAL( ARRAY )
CALL FLUSH( 6 )
```

2. Use the `SUM` function to check the sum of an array:

```
PRINT*, '### Sum of X : ', SUM( ARRAY )
CALL FLUSH( 6 )
```

20.12 Use the brute-force method when all else fails

If the bug is difficult to locate, then comment out a large section of code and run your **GEOS-Chem** or **HEMCO** simulation again. If the error does not occur, then uncomment some more code and run again. Repeat the process until you find the location of the error. The brute force method may be tedious, but it will usually lead you to the source of the problem.

20.13 Identify poorly-performing code with a profiler

If you think your **GEOS-Chem** or **HEMCO** simulation is taking too long to run, consider using profiling tools to generate a list of the time that is spent in each routine. This can help you identify badly written and/or poorly-parallelized code. For more information, please see [our Profiling GEOS-Chem wiki page](#).

MANAGE A DATA ARCHIVE WITH BASHDATACATALOG

If you need to download a large amount of input data for **GEOS-Chem** or **HEMCO** (e.g. in support of a large user group at your institution) you may find **bashdatacatalog** helpful.

21.1 What is bashdatacatalog?

The **bashdatacatalog** is a command-line tool (written by [Liam Bindle](#)) that facilitates synchronizing local data collections with a remote data source. With the **bashdatacatalog**, you can run queries on your local data collections to answer questions like “What files am I missing?” or “What files aren’t bitwise identical to remote data?”. Queries can include a date range, in which case collections with temporal assets are filtered-out accordingly. The **bashdatacatalog** can format the results of queries as: a URL download list, a Globus transfer list, an rsync transfer list, or simply a file list.

The **bashdatacatalog** was written to facilitate downloading input data for users of the [GEOS-Chem atmospheric chemistry model](#). The canonical GEOS-Chem input data repository has >1 M files and >100 TB of data, and the input data required for a simulation depends on the model version and simulation parameters such as start and end date.

21.2 Usage instructions

For detailed instructions on using **bashdatacatalog**, please see the [bashdatacatalog wiki on Github](#).

Also see our [input-data-catalogs Github repository](#) for comma-separated input lists of GEOS-Chem data, separated by model version.

WORK WITH NETCDF FILES

On this page we provide some useful information about working with data files in netCDF format.

22.1 Useful tools

There are many free and open-source software packages readily available for visualizing and manipulating netCDF files.

cdo

Climate Data Operators: Highly-optimized command-line tools for manipulating and analyzing netCDF files. Contains features that are especially useful for Earth Science applications.

See: <https://code.zmaw.de/projects/cdo>

GCPy

GEOS-Chem Python toolkit: Python package for visualizing and analyzing GEOS-Chem output. Used for creating the GEOS-Chem benchmark plots. Also contains some useful routines for creating single-panel plots and multi-panel difference plots, as well as file regridding utilities.

See: <https://gcpy.readthedocs.io>

ncdump

Generates a text representation of netCDF data and can be used to quickly view the variables contained in a netCDF file. **ncdump** is installed to the `bin/` folder of your netCDF library distribution.

See: <https://www.unidata.ucar.edu/software/netcdf/workshops/2011/utilities/Ncdump.html>

nco

netCDF operators: Highly-optimized command-line tools for manipulating and analyzing netCDF files.

See: <http://nco.sourceforge.net>

ncview

Visualization package for netCDF files. **Ncview** has limited features, but is great for a quick look at the contents of netCDF files.

See: http://meteora.ucsd.edu/~pierce/ncview_home_page.html

netcdf-scripts

Our repository of useful netCDF utility scripts for GEOS-Chem.

See: <https://github.com/geoschem/netcdf-scripts>

Panoply

Java-based data viewer for netCDF files. This package offers an alternative to **ncview**. From our experience, Panoply works nicely when installed on the desktop, but is slow to respond in the Linux environment.

See: <https://www.giss.nasa.gov/tools/panoply/>

xarray

Python package that lets you read the contents of a netCDF file into a data structure. The data can then be further manipulated or converted to numpy or dask arrays for further processing.

See: <https://xarray.readthedocs.io>

Some of the tools listed above, such as **ncdump** and **ncview** may come pre-installed on your system. Others may need to be installed or loaded (e.g. via the **module load** command). Check with your system administrator or IT staff to see what is available on your system.

22.2 Examine the contents of a netCDF file

An easy way to examine the contents of a netCDF file is to use **ncdump** as follows:

```
$ ncdump -ct GEOSChem.SpeciesConc.20190701_0000z.nc4
```

You will see output similar to this:

```
netcdf GEOSChem.SpeciesConc.20190701_0000z {
dimensions:
    time = UNLIMITED ; // (1 currently)
    lev = 72 ;
    ilev = 73 ;
    lat = 46 ;
    lon = 72 ;
    nb = 2 ;
variables:
    double time(time) ;
        time:long_name = "Time" ;
        time:units = "minutes since 2019-07-01 00:00:00" ;
        time:calendar = "gregorian" ;
        time:axis = "T" ;
    double lev(lev) ;
        lev:long_name = "hybrid level at midpoints ((A/P0)+B)" ;
        lev:units = "level" ;
        lev:axis = "Z" ;
        lev:positive = "up" ;
        lev:standard_name = "atmosphere_hybrid_sigma_pressure_coordinate" ;
        lev:formula_terms = "a: hyam b: hybm p0: P0 ps: PS" ;
    double ilev(ilev) ;
        ilev:long_name = "hybrid level at interfaces ((A/P0)+B)" ;
        ilev:units = "level" ;
        ilev:positive = "up" ;
        ilev:standard_name = "atmosphere_hybrid_sigma_pressure_coordinate" ;
        ilev:formula_terms = "a: hyai b: hybi p0: P0 ps: PS" ;
    double lat_bnds(lat, nb) ;
        lat_bnds:long_name = "Latitude bounds (CF-compliant)" ;
        lat_bnds:units = "degrees_north" ;
    double lat(lat) ;
        lat:long_name = "Latitude" ;
        lat:units = "degrees_north" ;
        lat:axis = "Y" ;
        lat:bounds = "lat_bnds" ;
    double lon_bnds(lon, nb) ;
        lon_bnds:long_name = "Longitude bounds (CF-compliant)" ;
```

(continues on next page)

(continued from previous page)

```

    lon_bnds:units = "degrees_east" ;
double lon(lon) ;
    lon:long_name = "Longitude" ;
    lon:units = "degrees_east" ;
    lon:axis = "X" ;
    lon:bounds = "lon_bnds" ;
double hyam(lev) ;
    hyam:long_name = "hybrid A coefficient at layer midpoints" ;
    hyam:units = "hPa" ;
double hybm(lev) ;
    hybm:long_name = "hybrid B coefficient at layer midpoints" ;
    hybm:units = "1" ;
double hyai(ilev) ;
    hyai:long_name = "hybrid A coefficient at layer interfaces" ;
    hyai:units = "hPa" ;
double hybi(ilev) ;
    hybi:long_name = "hybrid B coefficient at layer interfaces" ;
    hybi:units = "1" ;
double P0 ;
    P0:long_name = "reference pressure" ;
    P0:units = "hPa" ;
float AREA(lat, lon) ;
    AREA:long_name = "Surface area" ;
    AREA:units = "m2" ;
float SpeciesConcVV_RCOOH(time, lev, lat, lon) ;
    SpeciesConc_RCOOH:long_name = "Dry mixing ratio of species RCOOH" ;
    SpeciesConcVV_RCOOH:units = "mol mol-1 dry" ;
    SpeciesConcVV_RCOOH:averaging_method = "time-averaged" ;
float SpeciesConcVV_O2(time, lev, lat, lon) ;
    SpeciesConcVV_O2:long_name = "Dry mixing ratio of species O2" ;
    SpeciesConcVV_O2:units = "mol mol-1 dry" ;
    SpeciesConcVV_O2:averaging_method = "time-averaged" ;
float SpeciesConcVV_N2(time, lev, lat, lon) ;
    SpeciesConcVV_N2:long_name = "Dry mixing ratio of species N2" ;
    SpeciesConcVV_N2:units = "mol mol-1 dry" ;
    SpeciesConcVV_N2:averaging_method = "time-averaged" ;
float SpeciesConcVV_H2(time, lev, lat, lon) ;
    SpeciesConcVV_H2:long_name = "Dry mixing ratio of species H2" ;
    SpeciesConcVV_H2:units = "mol mol-1 dry" ;
    SpeciesConcVV_H2:averaging_method = "time-averaged" ;
float SpeciesConcVV_O(time, lev, lat, lon) ;
    SpeciesConcVV_O:long_name = "Dry mixing ratio of species O" ;
    SpeciesConcVVO:units = "mol mol-1 dry" ;

    ... etc ...

// global attributes:
    :title = "GEOS-Chem diagnostic collection: SpeciesConc" ;
    :history = "" ;
    :format = "not found" ;
    :conventions = "COARDS" ;
    :ProdDateTime = "" ;
    :reference = "www.geos-chem.org; wiki.geos-chem.org" ;
    :contact = "GEOS-Chem Support Team (geos-chem-support@g.harvard.edu)" ;
    :simulation_start_date_and_time = "2019-07-01 00:00:00z" ;
    :simulation_end_date_and_time = "2019-07-01 01:00:00z" ;

data:

```

(continues on next page)

(continued from previous page)

```

time = "2019-07-01 00:30" ;

lev = 0.99250002413, 0.97749990013, 0.962499776, 0.947499955, 0.93250006,
      0.91749991, 0.90249991, 0.88749996, 0.87249996, 0.85750006, 0.842500125,
      0.82750016, 0.8100002, 0.78750002, 0.762499965, 0.737500105, 0.7125001,
      0.6875001, 0.65625015, 0.6187502, 0.58125015, 0.5437501, 0.5062501,
      0.4687501, 0.4312501, 0.3937501, 0.3562501, 0.31279158, 0.26647905,
      0.2265135325, 0.192541016587707, 0.163661504087706, 0.139115, 0.11825,
      0.10051436, 0.085439015, 0.07255786, 0.06149566, 0.05201591, 0.04390966,
      0.03699271, 0.03108891, 0.02604911, 0.021761005, 0.01812435, 0.01505025,
      0.01246015, 0.010284921, 0.008456392, 0.0069183215, 0.005631801,
      0.004561686, 0.003676501, 0.002948321, 0.0023525905, 0.00186788,
      0.00147565, 0.001159975, 0.00090728705, 0.0007059566, 0.0005462926,
      0.0004204236, 0.0003217836, 0.00024493755, 0.000185422, 0.000139599,
      0.00010452401, 7.7672515e-05, 5.679251e-05, 4.0142505e-05, 2.635e-05,
      1.5e-05 ;

ilev = 1, 0.98500004826, 0.969999752, 0.9549998, 0.94000011, 0.92500001,
      0.90999981, 0.89500001, 0.87999991, 0.86500001, 0.85000011, 0.83500014,
      0.82000018, 0.80000022, 0.77499982, 0.75000011, 0.7250001, 0.7000001,
      0.6750001, 0.6375002, 0.6000002, 0.5625001, 0.5250001, 0.4875001,
      0.4500001, 0.4125001, 0.3750001, 0.3375001, 0.28808306, 0.24487504,
      0.208152025, 0.176930008175413, 0.150393, 0.127837, 0.108663, 0.09236572,
      0.07851231, 0.06660341, 0.05638791, 0.04764391, 0.04017541, 0.03381001,
      0.02836781, 0.02373041, 0.0197916, 0.0164571, 0.0136434, 0.0112769,
      0.009292942, 0.007619842, 0.006216801, 0.005046801, 0.004076571,
      0.003276431, 0.002620211, 0.00208497, 0.00165079, 0.00130051, 0.00101944,
      0.0007951341, 0.0006167791, 0.0004758061, 0.0003650411, 0.0002785261,
      0.000211349, 0.000159495, 0.000119703, 8.934502e-05, 6.600001e-05,
      4.758501e-05, 3.27e-05, 2e-05, 1e-05 ;

lat = -89, -86, -82, -78, -74, -70, -66, -62, -58, -54, -50, -46, -42, -38,
      -34, -30, -26, -22, -18, -14, -10, -6, -2, 2, 6, 10, 14, 18, 22, 26, 30,
      34, 38, 42, 46, 50, 54, 58, 62, 66, 70, 74, 78, 82, 86, 89 ;

lon = -180, -175, -170, -165, -160, -155, -150, -145, -140, -135, -130,
      -125, -120, -115, -110, -105, -100, -95, -90, -85, -80, -75, -70, -65,
      -60, -55, -50, -45, -40, -35, -30, -25, -20, -15, -10, -5, 0, 5, 10, 15,
      20, 25, 30, 35, 40, 45, 50, 55, 60, 65, 70, 75, 80, 85, 90, 95, 100, 105,
      110, 115, 120, 125, 130, 135, 140, 145, 150, 155, 160, 165, 170, 175 ;
}

```

You can also use **ncdump** to display the data values for a given variable in the netCDF file. This command will display the values in the `SpeciesRst_O3` variable to the screen:

```
$ ncdump -v SpeciesConc_O3 GEOSChem.SpeciesConc.20190701_0000z.nc4 | less
```

Or you can redirect the output to a file:

```
$ ncdump -v SpeciesConc_O3 GEOSChem.SpeciesConc.20190701_0000z.nc4 > log
```

22.3 Read the contents of a netCDF file

22.3.1 Read data with Python

The easiest way to read a netCDF file is to use the `xarray` Python package.

```
#!/usr/bin/env python

# Imports
import numpy as np
import xarray as xr

# Read a restart file into an xarray Dataset object
ds = xr.open_dataset("GEOSChem.SpeciesConc.20190701_0000z.nc4")

# Print the contents of the DataSet
print(ds)

# Print units of data
print(f"\nUnits of SpeciesRst_03: {ds['SpeciesConc_03'].units}")

# Print the sum, max, and min of the data
# NOTE .values returns a numpy ndarray so that we can use
# other numpy functions like np.sum() on the data
print(f"Sum of SpeciesRst_03: {np.sum(ds['SpeciesConc_03'].values)}")
print(f"Max of SpeciesRst_03: {np.max(ds['SpeciesConc_03'].values)}")
print(f"Min of SpeciesRst_03: {np.min(ds['SpeciesConc_03'].values)}")
```

This above script will print the following output:

```
<xarray.Dataset>
Dimensions:                (ilev: 73, lat: 46, lev: 72, lon: 72, nb: 2, time: 1)
Coordinates:
  * time                    (time) datetime64[ns] 2019-07-01T00:30:00
  * lev                     (lev) float64 0.9925 0.9775 ... 2.635e-05 1.5e-05
  * ilev                    (ilev) float64 1.0 0.985 0.97 ... 3.27e-05 2e-05 1e-05
  * lat                     (lat) float64 -89.0 -86.0 -82.0 ... 82.0 86.0 89.0
  * lon                     (lon) float64 -180.0 -175.0 -170.0 ... 170.0 175.0
Dimensions without coordinates: nb
Data variables: (12/315)
    lat_bnds                (lat, nb) float64 ...
    lon_bnds                (lon, nb) float64 ...
    hyam                    (lev) float64 ...
    hybm                    (lev) float64 ...
    hyai                    (ilev) float64 ...
    hybi                    (ilev) float64 ...
    ...
    SpeciesConc_AONITA      (time, lev, lat, lon) float32 ...
    SpeciesConc_ALK4       (time, lev, lat, lon) float32 ...
    SpeciesConc_ALD2       (time, lev, lat, lon) float32 ...
    SpeciesConc_AERI       (time, lev, lat, lon) float32 ...
    SpeciesConc_ACTA       (time, lev, lat, lon) float32 ...
    SpeciesConc_ACET       (time, lev, lat, lon) float32 ...
Attributes:
    title:                  GEOS-Chem diagnostic collection: Species...
    history:
    format:                  not found
```

(continues on next page)

(continued from previous page)

```

conventions:                COARDS
ProdDateTime:
reference:                   www.geos-chem.org; wiki.geos-chem.org
contact:                     GEOS-Chem Support Team (geos-chem-suppor...
simulation_start_date_and_time: 2019-07-01 00:00:00z
simulation_end_date_and_time:   2019-07-01 01:00:00z

Units of SpeciesRst_O3: mol mol-1 dry
Sum of SpeciesRst_O3: 0.4052325189113617
Max of SpeciesRst_O3: 1.01212954177754e-05
Min of SpeciesRst_O3: 3.758645839013752e-09

```

22.3.2 Read data from multiple files in Python

The xarray package will also let you read data from multiple files into a single Dataset object. This is done with the `open_mfdataset` (open multi-file-dataset) function as shown below:

```

#!/usr/bin/env python

# Imports
import xarray as xr

# Create a list of files to open
filelist = [
    'GEOSChem.SpeciesConc.20160101_0000z.nc4',
    'GEOSChem.SpeciesConc.20160201_0000z.nc4',
    ...
]

# Read a restart file into an xarray Dataset object
ds = xr.open_mfdataset(filelist)

```

22.4 Determining if a netCDF file is COARDS-compliant

All netCDF files used as input to GEOS-Chem and/or HEMCO must adhere to the *COARDS netCDF conventions*. You can use the `isCoards` script (from our [netcdf-scripts repository at GitHub](#)) to determine if a netCDF file adheres to the COARDS conventions.

Run the `isCoards` script at the command line on any netCDF file, and you will receive a report as to which elements of the file do not comply with the COARDS conventions.

```

$ isCoards myfile.nc

=====
Filename: myfile.nc
=====

The following items adhere to the COARDS standard:
-----
-> Dimension "time" adheres to standard usage
-> Dimension "lev" adheres to standard usage
-> Dimension "lat" adheres to standard usage

```

(continues on next page)

(continued from previous page)

```

-> Dimension "lon" adheres to standard usage
-> time(time)
-> time is monotonically increasing
-> time:axis = "T"
-> time:calendar = "gregorian"
-> time:long_name = "Time"
-> time:units = "hours since 1985-1-1 00:00:0.0"
-> lev(lev)
-> lev is monotonically decreasing
-> lev:axis = "Z"
-> lev:positive = "up"
-> lev:long_name = "GEOS-Chem levels"
-> lev:units = "sigma_level"
-> lat(lat)
-> lat is monotonically increasing
-> lat:axis = "Y"
-> lat:long_name = "Latitude"
-> lat:units = "degrees_north"
-> lon(lon)
-> lon is monotonically increasing
-> lon:axis = "X"
-> lon:long_name = "Longitude"
-> lon:units = "degrees_east"
-> OH(time,lev,lat,lon)
-> OH:long_name = "Chemically produced OH"
-> OH:units = "kg/m3"
-> OH:long_name = 1.e+30f
-> OH:missing_value = 1.e+30f
-> conventions: "COARDS"
-> history: "Mon Apr  3 08:26:19 2017"
-> title: "COARDS/netCDF file created by BPCH2COARDS (GAMAP v2-17+)"
-> format: "NetCDF-3"

```

The following items DO NOT ADHERE to the COARDS standard:

```

-----
-> time[0] != 0 (problem for GCHP)

```

The following optional items are RECOMMENDED:

```

-----
-> Consider adding the "references" global attribute

```

22.5 Edit variables and attributes

As discussed *in the preceding section*, you may find that you need to edit your netCDF files for COARDS-compliance. Below are several useful commands for editing netCDF files. Many of these commands utilize the *nco* and *cdo* utilities.

1. Display the header and coordinate variables of a netCDF file, with the time variable displayed in human-readable format. Also show status of file *compression and/or chunking*.

```
$ ncdump -cts file.nc
```

2. *Compress a netCDF file*. This can considerably reduce the file size!

```
# No deflation
$ nccopy -d0 myfile.nc tmp.nc
$ mv tmp.nc myfile.nc

# Minimum deflation (good for most applications)
$ nccopy -d1 myfile.nc tmp.nc
$ mv tmp.nc myfile.nc

# Medium deflation
$ nccopy -d5 myfile.nc tmp.nc
$ mv tmp.nc myfile.nc

# Maximum deflation
$ nccopy -d9 myfile.nc tmp.nc
$ mv tmp.nc myfile.nc
```

3. Change variable name from SpeciesConc_NO to NO:

```
$ ncrename -v SpeciesConc_NO,NO myfile.nc
```

4. Set all missing values to zero:

```
$ cdo setemisstoc,0 myfile.nc tmp.nc
$ mv tmp.nc myfile.nc
```

5. Add/change the long_name attribute of the vertical coordinate (lev) to GEOS-Chem levels. This will ensure that **HEMCO** recognizes the vertical levels of the input file as GEOS-Chem model levels.

```
$ ncatted -a long_name,lev,o,c,"GEOS-Chem levels" myfile.nc
```

6. Add/change the axis and positive attributes of the vertical coordinate (lev):

```
$ ncatted -a axis,lev,o,c,"Z" myfile.nc
$ ncatted -a positive,lev,o,c,"up" myfile.nc
```

7. Add/change the units attribute of the latitude (lat) coordinate to degrees_north:

```
$ ncatted -a units,lat,o,c,"degrees_north" myfile.nc
```

8. Convert the units attribute of the CHLA variable from mg/m3 to kg/m3

```
$ ncap2 -v -s "CHLA=CHLA/1000000.0f" myfile.nc tmp.nc
$ ncatted -a units,CHLA,o,c,"kg/m3" tmp.nc
$ mv tmp.nc myfile.nc
```

9. Add/change the references, title, and history global attributes

```
$ ncatted -a references,global,o,c,"www.geos-chem.org; wiki.geos-chem.org" myfile.
↪nc
$ ncatted -a history,global,o,c,"Tue Mar 3 12:18:38 EST 2015" myfile.nc
$ ncatted -a title,global,o,c,"XYZ data from ABC source" myfile.nc
```

10. Remove the references global attribute:

```
$ ncatted -a references,global,d,, myfile.nc
```

11. Add a time dimension to a file that does not have one:

```
$ ncap2 -h -s 'defdim("time",1);time[time]=0.0;time@long_name="time";
↪time@calendar="standard";time@units="days since 2007-01-01 00:00:00"' -O myfile.
↪nc tmp.nc
$ mv tmp.nc myfile.nc
```

12. Add a time dimension to a variable:

```
# Assume myVar has lat and lon dimensions to start with
$ ncap2 -h -s 'myVar[$time,$lat,$lon]=myVar;' myfile.nc tmp.nc
$ mv tmp.nc myfile.nc
```

13. Make the time dimension unlimited:

```
$ ncks --mk_rec_dmn time myfile.nc tmp.nc
$ mv tmp.nc myfile.nc
```

14. Change the file reference date and time (i.e. time:units) from 1 Jan 1985 to 1 Jan 2000:

```
$ cdo setreftime,2000-01-01,00:00:00 myfile.nc tmp.nc
$ mv tmp.nc myfile.nc
```

15. Shift all time values ahead or back by 1 hour in a file:

```
# Shift ahead 1 hour
$ cdo shifttime,1hour myfile.nc tmp.nc
$ mv tmp.nc myfile.nc

# Shift back 1 hour
$ cdo shifttime,-1hour myfile.nc tmp.nc
$ mv tmp.nc myfile.nc
```

16. Set the date of all variables in the file. (Useful for files that have only one time point.)

```
$ cdo setdate,2019-07-02 myfile.nc tmp.nc
$ mv tmp.nc myfile.nc
```

Tip: The following **cdo** commands are similar to **cdo setdate**, but allow you to manipulate other time variables:

```
$ cdo settime,03:00:00 ... # Sets time to 03:00 UTC
$ cdo setday,26, ... # Sets day of month to 26
$ cdo setmon,10, ... # Sets month to 10 (October)
$ cdo setyear,1992, ... # Sets year to 1992
```

See the [cdo user manual](#) for more information.

17. Change the time:calendar attribute:

GEOS-Chem and HEMCO cannot read data from netCDF files where:

```
time:calendar = "360_day"
time:calendar = "365_day"
time:calendar = "noleap"
```

We recommend converting the calendar used in the netCDF file to the standard netCDF calendar with these commands:

```
$ cdo setcalendar,standard myfile.nc tmp.nc
$ mv tmp.nc myfile.nc
```

18. Change the type of the time coordinate from `int` to `double`:

```
$ ncap2 -s 'time=double(time)' myfile.nc tmp.nc
$ mv tmp.nc myfile.nc
```

22.6 Concatenate netCDF files

There are a couple of ways to concatenate multiple netCDF files into a single netCDF file, as shown in the sections below.

22.6.1 Concatenate with the netCDF operators

You can use the **ncrcat** utility (from *nc*) to concatenate the individual netCDF files into a single netCDF file.

Let's assume we want to combine 12 monthly data files (e.g. `month_01.nc`, `month_02.nc`, .. `month_12.nc`) into a single file called `annual_data.nc`.

First, make sure that each of the `month_*.nc` files has an unlimited `time` dimension. Type this at the command line:

```
$ ncdump -ct month_01.nc | grep "time"
```

Then you should see this as the first line in the output:

```
time = UNLIMITED ; // (1 currently)
```

This indicates that the time dimension is unlimited. If on the other hand you see this output:

```
time = 1 ;
```

Then it means that the time dimension is fixed. If this is the case, you will have to use the **ncks** command to make the time dimension unlimited, as follows:

```
$ ncks --mk_rec_dmn time month_01.nc tmp.nc
$ mv tmp.nc month_01.nc
... etc for the other files ...
```

Then use **ncrcat** to combine the monthly data along the time dimension, and save the result to a single netCDF file:

```
$ ncrcat -h0 month_*.nc annual_data.nc
```

You may then discard the `month_*.nc` files if so desired.

22.6.2 Concatenate with Python

You can use the `xarray` Python package to create a single netCDF file from multiple files. [Click HERE](#) to view a sample Python script that does this.

22.7 Regrid netCDF files

The following tools can be used to regrid netCDF data files (such as GEOS-Chem restart files and GEOS-Chem diagnostic files).

22.7.1 Regrid with `cdo`

`cdo` includes several tools for regridding netCDF files. For example:

```
# Apply conservative regridding
$ cdo remapcon,gridfile infile.nc outfile.nc
```

For `gridfile`, you can use the files [here](#). Also see [this reference](#).

Issue with `cdo remapdis` regridding tool

GEOS-Chem user **Bram Maasakkers** wrote:

I have noticed a problem regridding GEOS-Chem diagnostic file to 2x2.5 using `cdo` version 1.9.4. When I use:

```
$ cdo remapdis,geos.2x25.grid GEOSChem.Restart.4x5.nc GEOSChem.Restart.2x25.
↪nc
```

The last latitudinal band (-89.5) remains empty and gets filled with the standard missing value of `cdo`, which is really large. This leads to immediate problems in the methane simulation as enormous concentrations enter the domain from the South Pole. For now I've solved this problem by just using bicubic interpolation

```
$ cdo remapbic,geos.2x25.grid GEOSChem.Restart.4x5.nc GEOSChem.Restart.2x25.
↪nc
```

You can also use conservative regridding:

```
$ cdo remapcon,geos.2x25.grid GEOSChem.Restart.4x5.nc GEOSChem.Restart.2x25.nc
```

22.7.2 Regrid with `GCPy`

`GCPy` (the GEOS-Chem Python Toolkit) has contains file regridding utilities that allow you to regrid from lat/lon to cubed-sphere grids (and vice versa). Regridding weights can be generated on-the-fly, or can be archived and reused. For detailed instructions, please see the [GCPy Regridding documentation](#).

22.7.3 Regrid with `nco`

`nco` also includes several regridding utilities. See the [Regridding](#) section of the [NCO User Guide](#) for more information.

22.7.4 Regrid with `xarray`

The `xarray` Python package has a built-in capability for 1-D interpolation. It wraps the [SciPy interpolation module](#). This functionality can also be used for vertical regridding.

22.7.5 Regrid with `xESMF`

`xESMF` is a universal regridding tool for geospatial data, which is written in Python. It can be used to regrid data not only on cartesian grids, but also on cubed-sphere and unstructured grids.

Note: `xESMF` only handles horizontal regridding.

22.8 Crop netCDF files

If needed, a netCDF file can be cropped to a subset of the globe with the `nco` or `cdo` utilities (cf. [Useful tools](#)).

For example, `cdo` has a `selbox` operator for selecting a box by specifying the lat/lon bounds:

```
$ cdo sellonlatbox,lon1,lon2,lat1,lat2 myfile.nc tmp.nc
$ mv tmp.nc myfile.nc
```

See the [cdo guide](#) for more information.

22.9 Add a new variable to a netCDF file

You have a couple of options for adding a new variable to a netCDF file (for example, when having to add a new species to an existing GEOS-Chem restart file).

1. You can use `cdo` and `nco` utilities to copy the data from one variable to another variable. For example:

```
#!/bin/bash

# Extract field SpeciesRst_PMN from the original restart file
cdo selvar,SpeciesRst_PMN initial_GEOSChem_rst.4x5_standard.nc NPMN.nc4

# Rename selected field to SpeciesRst_NPMN
ncrename -h -v SpeciesRst_PMN,Species_Rst_NPMN NMPN.nc4

# Append new species to existing restart file
ncks -h -A -M NMPN.nc4 initial_GEOSChem_rst.4x5_standard.nc
```

2. **Sal Farina** wrote a simple Python script for adding a new species to a netCDF restart file:

```
#!/usr/bin/env python

import netCDF4 as nc
import sys
import os

for nam in sys.argv[1:]:
    f = nc.Dataset(nam, mode='a')
    try:
        o = f['SpeciesRst_OCPI']
    except:
        print "SpeciesRst_OCPI not defined"
        f.createVariable('SpeciesRst_SOAP', o.datatype, dimensions=o.dimensions, fill_
↪value=o._FillValue)
        soap = f['SpeciesRst_SOAP']
        soap[:] = 0.0
        soap.long_name= 'SOAP species'
        soap.units = o.units
        soap.add_offset = 0.0
        soap.scale_factor = 1.0
        soap.missing_value = 1.0e30
        f.close()
```

3. Bob Yantosca wrote this Python script to insert a fake species into GEOS-Chem Classic and GCHP restart files (13.3.0)

```
#!/usr/bin/env python
"""
Adds an extra DataArray for into restart files:
Calling sequence:
    ./append_species_into_restart.py
"""
# Imports
import gcpy.constants as gcon
import xarray as xr
from xarray.coding.variables import SerializationWarning
import warnings

# Suppress harmless run-time warnings (mostly about underflow or NaNs)
warnings.filterwarnings("ignore", category=RuntimeWarning)
warnings.filterwarnings("ignore", category=UserWarning)
warnings.filterwarnings("ignore", category=SerializationWarning)

def main():
    """
    Appends extra species to restart files.
    """
    # Data vars to skip
    skip_vars = gcon.skip_these_vars
    # List of dates
    file_list = [
        'GEOSChem.Restart.fullchem.20190101_0000z.nc4',
        'GEOSChem.Restart.fullchem.20190701_0000z.nc4',
        'GEOSChem.Restart.TOMAS15.20190701_0000z.nc4',
        'GEOSChem.Restart.TOMAS40.20190701_0000z.nc4',
        'GCHP.Restart.fullchem.20190101_0000z.c180.nc4',
        'GCHP.Restart.fullchem.20190101_0000z.c24.nc4',
```

(continues on next page)

(continued from previous page)

```

'GCHP.Restart.fullchem.20190101_0000z.c360.nc4',
'GCHP.Restart.fullchem.20190101_0000z.c48.nc4',
'GCHP.Restart.fullchem.20190101_0000z.c90.nc4',
'GCHP.Restart.fullchem.20190701_0000z.c180.nc4',
'GCHP.Restart.fullchem.20190701_0000z.c24.nc4',
'GCHP.Restart.fullchem.20190701_0000z.c360.nc4',
'GCHP.Restart.fullchem.20190701_0000z.c48.nc4',
'GCHP.Restart.fullchem.20190701_0000z.c90.nc4'
]
# Keep all netCDF attributes
with xr.set_options(keep_attrs=True):
    # Loop over dates
    for f in file_list:
        # Input and output files
        infile = '../' + f
        outfile = f
        print("Creating " + outfile)

        # Open input file
        ds = xr.open_dataset(infile, drop_variables=skip_vars)
        # Create a new DataArray from a given species (EDIT ACCORDINGLY)
        if "GCHP" in infile:
            dr = ds["SPC_ETO"]
            dr.name = "SPC_ETOO"
        else:
            dr = ds["SpeciesRst_ETO"]
            dr.name = "SpeciesRst_ETOO"

        # Update attributes (EDIT ACCORDINGLY)
        dr.attrs["FullName"] = "peroxy radical from ethene"
        dr.attrs["Is_Gas"] = "true"
        dr.attrs["long_name"] = "Dry mixing ratio of species ETOO"
        dr.attrs["MW_g"] = 77.06
        # Merge the new DataArray into the Dataset
        ds = xr.merge([ds, dr], compat="override")

        # Create a new file
        ds.to_netcdf(outfile)

        # Free memory by setting ds to a null dataset
        ds = xr.Dataset()

if __name__ == "__main__":
    main()

```

22.10 Chunk and deflate a netCDF file to improve I/O

We recommend that you **chunk** the data in your netCDF file. Chunking specifies the order in along which the data will be read from disk. The Unidata web site has a good overview of why chunking a netCDF file matters.

For GEOS-Chem with the high-performance option (aka GCHP), the best file I/O performance occurs when the file is split into one chunk per level (assuming your data has a lev dimension). This allows each individual vertical level of data to be read in parallel.

You can use the **nccopy** command of *ncio* to do the chunking. For example, say you have a netCDF file called

myfile.nc with these dimensions:

```
dimensions:
    time = UNLIMITED ; // (12 currently)
    lev = 72 ;
    lat = 181 ;
    lon = 360 ;
```

Then you can use the **nccopy** command to apply the optimal chunking along levels:

```
$ nccopy -c lon/360,lat/181,lev/1,time/1 -d1 myfile.nc tmp.nc
$ mv tmp.nc myfile.nc
```

This will create a new file called tmp.nc that has the proper chunking. We then replace myfile.nc with this temporary file.

You can specify the chunk sizes that will be applied to the variables in the netCDF file with the **-c** argument to **nccopy**. To obtain the optimal chunking, the lon chunksize must be identical to the number of values along the longitude dimension (e.g. lon/360 and the lat chunksize must be equal to the number of points in the latitude dimension (e.g. lat/181).

We also recommend that you **deflate** (i.e. compress) the netCDF data variables at the same time you apply the chunking. Deflating can substantially reduce the file size, especially for emissions data that are only defined over the land but not over the oceans. You can deflate the data in a netCDF file by specifying the **-d** argument to nccopy. There are 10 possible deflation levels, ranging from 0 (no deflation) to 9 (max deflation). For most purposes, a deflation level of 1 (**d1**) is sufficient.

The [GEOS-Chem Support Team](#) has created a Perl script named **nc_chunk.pl** (contained in the [netcdf-scripts repository at GitHub](#)) that will automatically chunk and compress data for you.

```
$ nc_chunk.pl myfile.nc      # Chunk netCDF file
$ nc_chunk.pl myfile.nc 1    # Chunk and compress file using deflate level 1
```

You can use the **ncdump -cts myfile.nc** command to view the chunk size and deflation level in the file. After applying the chunking and compression to myfile.nc, you would see output such as this:

```
dimensions:
    time = UNLIMITED ; // (12 currently)
    lev = 72 ;
    lat = 181 ;
    lon = 360 ;
variables:
    float PRPE(time, lev, lat, lon) ;
        PRPE:long_name = "Propene" ;
        PRPE:units = "kgC/m2/s" ;
        PRPE:add_offset = 0.f ;
        PRPE:scale_factor = 1.f ;
        PRPE:_FillValue = 1.e+15f ;
        PRPE:missing_value = 1.e+15f ;
        PRPE:gamap_category = "ANTHSRCE" ;
        PRPE:_Storage = "chunked" ;
        PRPE:_ChunkSizes = 1, 1, 181, 360 ;
        PRPE:_DeflateLevel = 1 ;
        PRPE:_Endianness = "little" ;\
    float CO(time, lev, lat, lon) ;
        CO:long_name = "CO" ;
        CO:units = "kg/m2/s" ;
        CO:add_offset = 0.f ;
```

(continues on next page)

(continued from previous page)

```
CO:scale_factor = 1.f ;
CO:_FillValue = 1.e+15f ;
CO:missing_value = 1.e+15f ;
CO:gamap_category = "ANTHSRCE" ;
CO:_Storage = "chunked" ;
CO:_ChunkSizes = 1, 1, 181, 360 ;
CO:_DeflateLevel = 1 ;
CO:_Endianness = "little" ;\
```

The attributes that begin with a `_` character are “hidden” netCDF attributes. They represent file properties instead of user-defined properties (like the long name, units, etc.). The “hidden” attributes can be shown by adding the `-s` argument to `ncdump`.

PREPARE COARDS-COMPLIANT NETCDF FILES

On this page we discuss how you can generate netCDF data files in the proper format for HEMCO and GEOS-Chem.

23.1 The COARDS netCDF standard

The [Harmonized Emissions Component \(HEMCO\)](#) reads data stored in the [netCDF file format](#), which is a common data format used in atmospheric and climate sciences. NetCDF files contain **data arrays** as well as **metadata**, which is a description of the data.

Several netCDF conventions have been developed in order to facilitate data exchange and visualization. The [Co-operative Ocean Atmosphere Research Data Service \(COARDS\) standard](#) defines regular conventions for naming dimensions as well as the [attributes](#) describing the data. You will find more information about these conventions in the sections below. HEMCO requires its input data to adhere to the COARDS standard.

Our *our “Work with netCDF files” supplemental guide* contains detailed instructions on how you can check a netCDF file for COARDS compliance.

23.2 COARDS dimensions

The **dimensions** of a netCDF file define how many grid boxes there are along a given direction. While the COARDS standard does not require any specific names for dimensions, accepted practice is to use these names for rectilinear grids:

time

Specifies the number of points along the time (T) axis.

The *time* dimension must always be specified. When you create the netCDF file, you may declare *time* to be UNLIMITED and then later define its size. This allows you to append further time points into the file later on.

lev

Specifies the number of points along the vertical level (Z) axis.

This dimension may be omitted none of the data arrays in the netCDF file have a vertical dimension.

lat

Specifies the number of points along the latitude (Y) axis.

lon

Specifies the number of points along the longitude (X) axis.

Note: For non-rectilinear grids (e.g. cubed-sphere), the *lat* and *lon* dimensions may be named NY and NX instead.

23.3 COARDS coordinate vectors

Coordinate vectors (aka **index variables** or **axis variables**) are 1-dimensional arrays that define the values along each axis.

The only COARDS requirement for coordinate vectors are these:

1. Each coordinate vector must be given the same name as the dimension that is used to define it.
2. All of the values contained within a coordinate vector must be either monotonically increasing or monotonically decreasing.

23.3.1 time

A COARDS-compliant *time* coordinate vector will have these features:

```
dimensions
    time = UNLIMITED ; // (12 currently)
...
variables
    double time(time) ;
        time:long_name = "time" ;
        time:units = "hours since 2010-01-01 00:00:00" ;
        time:calendar = "standard" ;
        time:axis = "T";
```

Note: The above was generated by the **ncdump** command.

As you can see, *time* is an 8-byte floating point (aka REAL*8 with 12 time points).

The *time* coordinate vector has following attributes:

time:long_name

A detailed description of the contents of this array. This is usually set to *time* or *Time*.

time:units

Specifies the number of hours, minutes, seconds, etc. that has elapsed with respect to a reference datetime YYYY-MM-DD hh:mn:ss. Set this to one of the following values:

- "days since YYYY-MM-DD hh:mn:ss"
- "hours since YYYY-MM-DD hh:mn:ss"
- "minutes since YYYY-MM-DD hh:mn:ss"
- "seconds since YYYY-MM-DD hh:mn:ss"

Tip: We recommend that you choose the reference datetime to correspond to the first time value in the file (i.e. `time(0) = 0`).

time:calendar

Specifies the calendar used to define the time system. Set this to one of the following values:

standard

Synonym for *gregorian*.

gregorian

Selects the Gregorian calendar system.

time:axis

Identifies the axis (X, Y, Z, T) corresponding to this coordinate vector. Set this to T.

Special considerations for time vectors

1. We recommend that index variables (such as `time`) be declared with type `float` or `double`. **GCHP** cannot parse files with that have index variables of type `int`.
2. We have noticed that netCDF files having a `time:units` reference datetime prior to 1900/01/01 00:00:00 may not be read properly when using **HEMCO** or **GCHP** within an ESMF environment. We therefore recommend that you use reference datetime values after 1900 whenever possible.
3. Weekly data must contain seven time slices in increments of one day. The first entry must represent Sunday data, regardless of the real weekday of the assigned datetime. It is possible to store weekly data for more than one time interval, in which case the first weekday (i.e. Sunday) must hold the starting date for the given set of (seven) time slices.
 - For instance, weekly data for every month of a year can be stored as 12 sets of 7 time slices. The reference datetime of the first entry of each set must fall on the first day of every month, and the following six entries must be increments of one day.

Currently, weekly data from netCDF files is not correctly read in an ESMF environment.

23.3.2 lev

A COARDS-compliant *lev* coordinate vector will have these features:

```
dimensions:
    lev = 72 ;
...
variables:
    double lev(lev) ;
        lev:long_name = "level" ;
        lev:units = "level" ;
        lev:positive = "up" ;
        lev:axis = "Z" ;
```

Here, *lev* is an 8-byte floating point (aka REAL*8) with 72 levels.

The *lev* coordinate vector has the following attributes:

lev:long_name

A detailed description of the contents of this array. You may set this to values such as:

- "level"
- "GEOS-Chem levels"
- "Eta centers"
- "Sigma centers"

lev:units

(Required) Specifies the units of vertical levels. Set this to one of the following:

- "levels"
- "eta_level"
- "sigma_level"

Important: If you set `long_name:` to `level` as well, then HEMCO will be able to regrid between GEOS-Chem vertical grids.

lev:axis

Identifies the axis (X, Y, Z, T) corresponding to this coordinate vector. Set this to Z.

lev:positive

Specifies the direction in which the vertical dimension is indexed. Set this to one of these values:

- "up" (Level 1 is the surface, and level indices increase upwards)
- "down" (Level 1 is the atmosphere top, and level indices increase downwards)

For emisissions and most other data sets, you can set `lev:positive` to "up".

Important: GCHP and the NASA GEOS-ESM use a vertical grid where `lev:positive` is "down".

Additional considerations for lev vectors:

When using [GEOS-Chem](#) or [HEMCO](#) in a non-ESMF environment, data is interpolated onto the simulation levels if the input data is on vertical levels other than the HEMCO model levels (see [HEMCO vertical regridding](#)).

Data on non-model levels must be on a hybrid sigma pressure coordinate system. In order to properly determine the vertical pressure levels of the input data, the file must contain the surface pressure values and the hybrid coefficients (a, b) of the coordinate system. Furthermore, the level variable must contain the attributes `standard_name` and `formula_terms` (the attribute `positive` is recommended but not required). A header excerpt of a valid netCDF file is shown below:

```
float lev(lev) ;
    lev:standard_name = "atmosphere_hybrid_sigma_pressure_coordinate" ;
    lev:units = "level" ;
    lev:positive = "down" ;
    lev:formula_terms = "ap: hyam b: hybm ps: PS" ;
float hyam(nhym) ;
    hyam:long_name = "hybrid A coefficient at layer midpoints" ;
    hyam:units = "hPa" ;
float hybm(nhym) ;
    hybm:long_name = "hybrid B coefficient at layer midpoints" ;
    hybm:units = "1" ;
float time(time) ;
    time:standard_name = "time" ;
    time:units = "days since 2000-01-01 00:00:00" ;
    time:calendar = "standard" ;
float PS(time, lat, lon) ;
    PS:long_name = "surface pressure" ;
    PS:units = "hPa" ;
float EMIS(time, lev, lat, lon) ;
```

(continues on next page)

(continued from previous page)

```
EMIS:long_name = "emissions" ;
EMIS:units = "kg m-2 s-1" ;
```

23.3.3 lat

A COARDS-compliant *lat* coordinate vector will have these features:

```
dimensions:
    lat = 181 ;
variables:
    double lat(lat) ;
        lat:long_name = "Latitude" ;
        lat:units = "degrees_north" ;
        lat:axis = "Y" ;
```

Here, *lat* is an 8-byte floating point (aka REAL*8) with 181 values.

The *lat* coordinate vector has the following attributes:

lat:long_name

A detailed description of the contents of this array. Set this to *Latitude*.

lat:units

Specifies the units of latitude. Set this to *degrees_north*.

lat:axis

Identifies the axis (X, Y, Z, T) corresponding to this coordinate vector. Set this to *Y*.

23.3.4 lon

A COARDS-compliant *lon* coordinate vector will have these features:

```
dimensions:
    lon = 360 ;
variables:
    double lon(lon) ;
        lon:long_name = "Longitude" ;
        lon:units = "degrees_east" ;
        lon:axis = "X" ;
```

Here, *lon* is an 8-byte floating point (aka REAL*8) with 360 values.

The *lon* coordinate vector has following attributes:

lon:long_name

A detailed description of the contents of this array. Set this to *Longitude*.

lon:units

Specifies the units of latitude. Set this to *degrees_east*.

lon:axis

Identifies the axis (X, Y, Z, T) corresponding to this coordinate vector. Set this to *X*.

Longitudes may be represented modulo 360. For example, -180, 180, and 540 are all valid representations of the International Dateline and 0 and 360 are both valid representations of the Prime Meridian. Note, however, that the sequence of numerical longitude values stored in the netCDF file must be monotonic in a non-modulo sense.

Practical guidelines:

1. If your grid begins at the International Dateline (-180°), then place your longitudes into the range -180..180.
2. If your grid begins at the Prime Meridian (0°), then place your longitudes into the range 0..360.

23.4 COARDS data arrays

A COARDS-compliant netCDF file may contain several **data arrays**. In our example file shown above, there are two data arrays:

```
dimensions:
    time = UNLIMITED ; // (12 currently)
    lev = 72 ;
    lat = 181 ;
    lon = 360 ;
variables:
    float PRPE(time, lev, lat, lon) ;
        PRPE:long_name = "Propene" ;
        PRPE:units = "kgC/m2/s" ;
        PRPE:add_offset = 0.f ;
        PRPE:missing_value = 1.e+15f ;
    float CO(time, lev, lat, lon) ;
        CO:long_name = "CO" ;
        CO:units = "kg/m2/s" ;
        CO:_FillValue = 1.e+15f ;
        CO:missing_value = 1.e+15f ;
```

These arrays contain emissions for species tracers PRPE (lumped < C3 alkenes) and CO.

23.4.1 Attributes for data arrays

long_name

Gives a detailed description of the contents of the array.

units

Specifies the units of data contained within the array. SI units are preferred.

Special usage for HEMCO:

- Use kg/m2/s or kg m⁻² s⁻¹ for emission fluxes of species
- Use kg/m3 or kg m⁻³ for concentration data;
- Use 1 for dimensionless data instead of unitless. HEMCO will recognize unitless, but it is non-standard and not recommended.

missing_value

Specifies the value that should represent missing data. This should be set to a number that will not be mistaken for a valid data value.

_FillValue

Synonym for *missing_value*. It is recommended to set both *missing_value* and *_FillValue* to the same value. Some data visualization packages look for one but not the other.

23.4.2 Ordering of the data

2D and 3D array variables in netCDF files must have specific dimension order. If the order is incorrect you will encounter netCDF read error “start+count exceeds dimension bound”. You can check the dimension ordering of your arrays by using the **ncdump** command as shown below:

```
$ ncdump file.nc -h
```

Be sure to check the dimensions listed next to the array name rather than the ordering of the dimensions listed at the top of the **ncdump** output.

The following dimension orders are acceptable:

```
array(time,lat,lon)
array(time,lat,lon,lev)
```

The rest of this section explains why the dimension ordering of arrays matters.

When you use **ncdump** to examine the contents of a netCDF file, you will notice that it displays the dimensions of the data in the opposite order with respect to Fortran. In our sample file, **ncdump** says that the CO and PRPE arrays have these dimensions:

```
CO(time,lev,lat,lon)
PRPE(time,lev,lat,lon)
```

But if you tried to read this netCDF file into GEOS-Chem (or any other program written in Fortran), you must use data arrays that have these dimensions:

```
CO(lon,lat,lev,time)
PRPE(lon,lat,lev,time)
```

Here’s why:

Fortran is a **column-major** language, which means that arrays are stored in memory by columns first, then by rows. If you have declared an arrays such as:

```
INTEGER      :: I, J, L, T
INTEGER, PARAMETER :: N_LON  = 360
INTEGER, PARAMETER :: N_LAT  = 181
INTEGER, PARAMETER :: N_LEV  = 72
INTEGER, PARAMETER :: N_TIME = 12
REAL*4       :: CO  (N_LON,N_LAT,N_LEV,N_TIME)
REAL*4       :: PRPE(N_LON,N_LAT,N_LEV,N_TIME)
```

then for optimal efficiency, the leftmost dimension (I) needs to vary the fastest, and needs to be accessed by the innermost DO-loop. Then the next leftmost dimension (J) should be accessed by the next innermost DO-loop, and so on. Therefore, the proper way to loop over these arrays is:

```
DO T = 1, N_TIME
DO L = 1, N_LEV
DO J = 1, N_LAT
DO I = 1, N_LON
  CO  (I,J,L,N) = ...
  PRPE(I,J,L,N) = ...
ENDDO
ENDDO
ENDDO
ENDDO
```

Note that the I index is varying most often, since it is the innermost DO-loop, then J , L , and T . This is opposite to how a car's odometer reads.

If you loop through an array in this fashion, with leftmost indices varying fastest, then the code minimizes the number of times it has to load subsections of the array into cache memory. In this optimal manner of execution, all of the array elements sitting in the cache memory are read in the proper order before the next array subsection needs to be loaded into the cache. But if you step through array elements in the wrong order, the number of cache loads is proportionally increased. Because it takes a finite amount of time to reload array elements into cache memory, the more times you have to access the cache, the longer it will take the code to execute. This can slow down the code dramatically.

On the other hand, C is a **row-major** language, which means that arrays are stored by rows first, then by columns. This means that the outermost do loop (I) is varying the fastest. This is identical to how a car's odometer reads.

If you use a Fortran program to write data to disk, and then try to read that data from disk into a program written in C, then unless you reverse the order of the DO loops, you will be reading the array in the wrong order. In C you would have to use this ordering scheme (using Fortran-style syntax to illustrate the point):

```
DO I = 1, N_LON
DO J = 1, N_LAT
DO L = 1, N_LEV
DO T = 1, N_TIME
    CO(T,L,J,I) = ...
    PRPE(T,L,J,I) = ...
ENDDO
ENDDO
ENDDO
ENDDO
```

Because **ncdump** is written in C, the order of the array appears opposite with respect to Fortran. The same goes for any other code written in a row-major programming language.

23.5 COARDS Global attributes

Global attributes are **netCDF attributes** that contain information about a netCDF file, as opposed to information about an individual data array.

From our example in the *Examine the contents of a netCDF file*, the output from **ncdump** showed that our sample netCDF file has several global attributes:

```
// global attributes:
:Title = "COARDS/netCDF file containing X data"
:Contact = "GEOS-Chem Support Team (geos-chem-support@as.harvard.edu)" ;
:References = "www.geos-chem.org; wiki.geos-chem.org" ;
:Conventions = "COARDS" ;
:Filename = "my_sample_data_file.1x1"
:History = "Mon Mar 17 16:18:09 2014 GMT" ;
:ProductionDateTime = "File generated on: Mon Mar 17 16:18:09 2014 GMT" ;
:ModificationDateTime = "File generated on: Mon Mar 17 16:18:09 2014 GMT" ;

:VersionID = "1.2" ;
:Format = "NetCDF-3" ;
:Model = "GEOS5" ;
:Grid = "GEOS_1x1" ;
:Delta_Lon = 1.f ;
:Delta_Lat = 1.f ;
:SpatialCoverage = "global" ;
```

(continues on next page)

(continued from previous page)

```
:NLayers = 72 ;  
:Start_Date = 20050101 ;  
:Start_Time = 00:00:00.0 ;  
:End_Date = 20051231 ;  
:End_Time = 23:59:59.99999 ;
```

Title (or title)

Provides a short description of the file.

Contact (or contact)

Provides contact information for the person(s) who created the file.

References (or references)

Provides a reference (citation, DOI, or URL) for the data contained in the file.

Conventions (or conventions)

Indicates if the netCDF file adheres to a standard (e.g. COARDS or CF).

Filename (or filename)

Specifies the name of the file.

History (or history)

Specifies the datetime of file creation, and of any subsequent modifications.

Note: If you edit the file with **nco** or **cdo**, then this attribute will be updated to reflect the modification that was done.

Format (or format)

Specifies the format of the netCDF file (such as netCDF-3 or netCDF-4).

23.6 For more information

Please see our [Work with netCDF files](#) Supplemental Guide for more information about commands that you can use to combine, edit, or manipulate data in netCDF files.

VIEW GEOS-CHEM SPECIES PROPERTIES

Properties for GEOS-Chem species are stored in the **GEOS-Chem Species Database**, which is a [YAML](#) file (`species_database.yml`) that is placed into each GEOS-Chem run directory.

View species properties from the current stable GEOS-Chem version:

- [View properties for most GEOS-Chem species](#)
- [View properties for APM microphysics species](#)
- [View properties for TOMAS microphysics species](#)
- [View properties for Hg simulation species](#)

24.1 Species properties defined

The following sections contain a detailed description of GEOS-Chem species properties.

24.1.1 Required default properties

All GEOS-Chem species should have these properties defined:

```
Name:
  FullName: full name of the species
  Formula: chemical formula of the species
  MW_g: molecular weight of the species in grams
EITHER Is_Gas: true
OR      Is_Aerosol: true
```

All other properties are species-dependent. You may omit properties that do not apply to a given species. GEOS-Chem will assign a “missing value” (e.g. `false`, `-999`, `-999.0`, or, `UNKNOWN`) to these properties when it reads the `species_database.yml` file from disk.

24.1.2 Identification

Name

Species short name (e.g. ISOP).

Formula

Species chemical formula (e.g. $\text{CH}_2=\text{C}(\text{CH}_3)\text{CH}=\text{CH}_2$). This is used to define the species' `formula` attribute, which gets written to GEOS-Chem diagnostic files and restart files.

FullName

Species long name (e.g. Isoprene). This is used to define the species' `long_name` attribute, which gets written to GEOS-Chem diagnostic files and restart files.

Is_Aerosol

Indicates that the species is an aerosol (`true`), or isn't (`false`).

Is_Advected

Indicates that the species is advected (`true`), or isn't (`false`).

Is_DryAlt

Indicates that dry deposition diagnostic quantities for the species can be archived at a specified altitude above the surface (`true`), or can't (`false`).

Note: The `Is_DryAlt` flag only applies to species `O3` and `HNO3`.

Is_DryDep

Indicates that the species is dry deposited (`true`), or isn't (`false`).

Is_HygroGrowth

Indicates that the species is an aerosol that is capable of hygroscopic growth (`true`), or isn't (`false`).

Is_Gas

Indicates that the species is a gas (`true`), or isn't (`false`).

Is_Hg0

Indicates that the species is elemental mercury (`true`), or isn't (`false`).

Is_Hg2

Indicates that the species is a mercury compound with oxidation state +2 (`true`), or isn't (`false`).

Is_HgP

Indicates that the species is a particulate mercury compound (`true`), or isn't (`false`).

Is_Photolysis

Indicates that the species is photolyzed (`true`), or isn't (`false`).

Is_Radionuclide

Indicates that the species is a radionuclide (`true`), or isn't (`false`).

24.1.3 Physical properties

Density

Density ($kg\ m^{-3}$) of the species. Typically defined only for aerosols.

Henry_K0

Henry's law solubility constant ($M\ atm^{-1}$), used by the default wet deposition scheme.

Henry_K0_Luo

Henry's law solubility constant ($M\ atm^{-1}$) used by the Luo *et al.* [[Luo et al., 2020]] wet deposition scheme.

Henry_CR

Henry's law volatility constant (K) used by the default wet deposition scheme.

Henry_CR_Luo

Henry's law volatility constant (K) used by the Luo *et al.* [[Luo et al., 2020]] wet deposition scheme.

Henry_pKa

Henry's Law pH correction factor.

MW_g

Molecular weight ($g\ mol^{-1}$) of the species.

Note: Some aerosol-phase species (such as MONITA and IONITA) are given the molar mass corresponding to the number of nitrogens that they carry, whereas gas-phase species (MONITS and MONITU) get the full molar mass of the compounds that they represent. This treatment has its origins in J. Fisher et al [2016].

Radius

Radius (m) of the species. Typically defined only for aerosols.

24.1.4 Dry deposition properties

DD_AeroDryDep

Indicates that dry deposition should consider hygroscopic growth for this species (`true`), or shouldn't (`false`).

Note: DD_AeroDryDep is only defined for sea salt aerosols.

DD_DustDryDep

Indicates that dry deposition should exclude hygroscopic growth for this species (`true`), or shouldn't (`false`).

Note: DD_DustDryDep is only defined for mineral dust aerosols.

DD_DvzAerSnow

Specifies the dry deposition velocity ($cm\ s^{-1}$) over ice and snow for certain aerosol species. Typically, DD_DvzAerSnow = 0.03.

DD_DvzAerSnow_Luo

Specifies the dry deposition velocity ($cm\ s^{-1}$) over ice and snow for certain aerosol species.

Note: DD_DvzAerSnow_Luo is only used when the Luo *et al.* [[Luo et al., 2020]] wet scavenging scheme is activated.

DD_DvzMinVal

Specifies minimum dry deposition velocities ($cm\ s^{-1}$) for sulfate species (SO_2 , SO_4 , MSA , NH_3 , NH_4 , NIT). This follows the methodology of the GOCART model.

DD_DvzMinVal is defined as a two-element vector:

- DD_DvzMinVal (1) sets a minimum dry deposition velocity onto snow and ice.
- DD_DvzMinVal (2) sets a minimum dry deposition velocity over land.

DD_Hstar_Old

Specifies the Henry's law constant (K_0) that is used in dry deposition. This will be used to assign the HSTAR variable in the GEOS-Chem dry deposition module.

Note: The value of the DD_Hstar_old parameter was tuned for each species so that the dry deposition velocity would match observations.

DD_F0

Specifies the reactivity factor for oxidation of biological substances in dry deposition.

DD_KOA

Specifies the octanal-air partition coefficient, used for the dry deposition of species POPG.

Note: DD_KOA is only used in the [POPs simulation](#).

24.1.5 Wet deposition properties

WD_Is_H2SO4

Indicates that the species is H_2SO_4 (true), or isn't (false). This allows the wet deposition code to perform special calculations when computing H_2SO_4 rainout and washout.

WD_Is_HNO3

Indicates that the species is HNO_3 (true), or isn't (false). This allows the wet deposition code to perform special calculations when computing HNO_3 rainout and washout.

WD_Is_SO2

Indicates that the species is SO_2 (true), or isn't (false). This allows the wet deposition code to perform special calculations when computing SO_2 rainout and washout.

WD_CoarseAer

Indicates that the species is a coarse aerosol (true), or isn't (false). For wet deposition purposes, the definition of coarse aerosol is radius $> 1\ \mu m$.

WD_LiqAndGas

Indicates that the ice-to-gas ratio can be computed for this species by co-condensation (true), or can't (false).

WD_ConvFacI2G

Specifies the conversion factor (i.e. ratio of sticking coefficients on the ice surface) for computing the ice-to-gas ratio by co-condensation, as used in the default wet deposition scheme.

Note: WD_ConvFacI2G only needs to be defined for those species for which WD_LiqAndGas is true.

WD_ConvFacI2G_Luo

Specifies the conversion factor (i.e. ratio of sticking coefficients on the ice surface) for computing the ice-to-gas ratio by co-condensation, as used in the Luo *et al.* [[Luo et al., 2020]] wet deposition scheme.

Note: WD_ConvFacI2G_Luo only needs to be defined for those species for which WD_LiqAndGas is true, and is only used when the Luo *et al.* [[Luo et al., 2020]] wet deposition scheme is activated.

WD_RetFactor

Specifies the retention efficiency R_i of species in the liquid cloud condensate as it is converted to precipitation. $R_i < 1$ accounts for volatilization during riming.

WD_AerScavEff

Specifies the aerosol scavenging efficiency. This factor multiplies F , the fraction of aerosol species that is lost to convective updraft scavenging.

- WD_AerScavEff = 1.0 for most aerosols.
- WD_AerScavEff = 0.8 for secondary organic aerosols.
- WD_AerScavEff = 0.0 for hydrophobic aerosols.

WD_KcScaleFac

Specifies a temperature-dependent scale factor that is used to multiply K (aka K_c), the rate constant for conversion of cloud condensate to precipitation.

WD_KcScaleFac is defined as a 3-element vector:

- WD_KcScaleFac(1) multiplies K when $T < 237$ kelvin.
- WD_KcScaleFac(2) multiplies K when $237 \leq T < 258$ kelvin
- WD_KcScaleFac(3) multiplies K when $T \geq 258$ kelvin.

WD_KcScaleFac_Luo

Specifies a temperature-dependent scale factor that is used to multiply K , aka K_c , the rate constant for conversion of cloud condensate to precipitation.

Used only in the Luo *et al.* [[Luo et al., 2020]] wet deposition scheme.

WD_KcScaleFac_Luo is defined as a 3-element vector:

- WD_KcScaleFac_Luo(1) multiplies K when $T < 237$ kelvin.
- WD_KcScaleFac_Luo(2) multiplies K when $237 \leq T < 258$ kelvin.
- WD_KcScaleFac_Luo(3) multiplies K when $T \geq 258$ kelvin.

WD_RainoutEff

Specifies a temperature-dependent scale factor that is used to multiply F_i (aka RAINFRAC), the fraction of species scavenged by rainout.

WD_RainoutEff is defined as a 3-element vector:

- WD_RainoutEff(1) multiplies F_i when $T < 237$ kelvin.
- WD_RainoutEff(2) multiplies F_i when $237 \leq T < 258$ kelvin.
- RainoutEff(3) multiplies F_i when $T \geq 258$ kelvin.

This allows us to better simulate scavenging by snow and impaction scavenging of BC. For most species, we need to be able to turn off rainout when $237 \leq T < 258$ kelvin. This can be easily done by setting RainoutEff(2) = 0.

Note: For SOA species, the maximum value of `WD_RainoutEff` will be 0.8 instead of 1.0.

WD_RainoutEff_Luo

Specifies a temperature-dependent scale factor that is used to multiply F_i (aka `RAINFRAC`), the fraction of species scavenged by rainout. (Used only in the [\[Luo et al., 2020\]](#) wet deposition scheme).

`WD_RainoutEff_Luo` is defined as a 3-element vector:

- `WD_RainoutEff_Luo(1)` multiplies F_i when $T < 237$ kelvin.
- `WD_RainoutEff_Luo(2)` multiplies F_i when $237 \leq T < 258$ kelvin.
- `RainoutEff_Luo(3)` multiplies F_i when $T \geq 258$ kelvin.

This allows us to better simulate scavenging by snow and impaction scavenging of BC. For most species, we need to be able to turn off rainout when $237 \leq T < 258$ kelvin. This can be easily done by setting `RainoutEff(2) = 0`.

Note: For SOA species, the maximum value of `WD_RainoutEff_Luo` will be 0.8 instead of 1.0.

24.1.6 Transport tracer properties

These properties are defined for species used in the `TransportTracers` simulation. We will refer to these species as **tracers**.

Is_Tracer

Indicates that the species is a transport tracer (`true`), or is not (`false`).

Snk_Horiz

Specifies the horizontal domain of the tracer sink term. Allowable values are:

all

The tracer sink term will be applied throughout the entire horizontal domain of the simulation grid.

lat_zone

The tracer sink term will be applied only within the latitude range specified by `Snk_Lats`.

Snk_Lats

Defines the latitude range [`min_latitude`, `max_latitude`] for the tracer sink term. Will only be used if `Snk_Horiz` is set to `lat_zone`.

Snk_Mode

Specifies how the tracer sink term will be applied. Allowable values are:

constant

The tracer sink term is a constant value (specified by `Snk_Value`).

efolding

The tracer sink term has an e-folding decay constant (specified in `Snk_Period`).

halflife

A tracer sink term has a half-life (specified in `Snk_Period`).

none

The tracer does not have a sink term.

Snk_Period

Specifies the period (in days) for which the tracer sink term will be applied.

Snk_Value

Specifies a value for the tracer sink term.

Snk_Vert

Specifies the vertical domain of the tracer sink term. Allowable values are:

all

The tracer sink term will be applied throughout the entire vertical domain of the simulation grid.

boundary_layer

The tracer sink term will only be applied within the planetary boundary layer.

surface

The tracer sink term will only be applied at the surface.

troposphere

The tracer sink term will only be applied within the troposphere.

Src_Add

Specifies whether the tracer has a source term (*true*) or not (*false*).

Src_Horiz

Specifies the horizontal domain of the tracer source term. Allowable values are:

all

The tracer source term will be applied across the entire horizontal extent of the simulation grid.

lat_zone

The tracer source term will only be applied within the latitude range specified by *Src_Lats*.

Src_Lats

Defines the latitude range [*min_latitude*, *max_latitude*] for the tracer source term. Will only be applied if *Src_Horiz* is set to *lat_zone*.

Src_Mode

Describes the type of tracer source term. Allowable values are:

constant

The tracer source term is a constant value (specified by *Src_Value*).

decay_of_another_species

The tracer source term comes from the decay of another species (e.g. Pb210 source comes from Rn222 decay).

HEMCO

The tracer source term will be read from a file via HEMCO.

maintain_mixing_ratio

The tracer source term will be calculated as needed to maintain a constant mixing ratio at the surface.

none

The tracer does not have a source term.

Src_Unit

Specifies the unit of the source term that will be applied to the tracer.

ppbv

The source term has units of parts per billion by volume.

timestep

The source term has units of per emissions timestep.

Src_Value

Specifies a value for the tracer source term in *Src_Units*.

Src_Vert

Specifies the vertical domain of the tracer source term. Allowable values are:

all

The tracer source term will be applied throughout the entire vertical domain of the simulation grid.

pressures

The tracer source term will only be applied within the pressure range specified in *Src_Pressures*.

stratosphere

The tracer source term will only be applied in the stratosphere.

troposphere

The tracer source term will only be applied in the troposphere.

surface

The tracer source term will only be applied at the surface.

Src_Pressures

Defines the pressure range [min_pressure, max_pressure], in hPa for the tracer source term. Will only be used if *Src_Vert* is set to pressures.

Units

Specifies the default units of the tracers (e.g. *aoa*, *aoa_nh*, *aoa_bl* are carried in units days, while all other species in GEOS-Chem are kg/kg dry air).

Properties used by each transport tracer

The list below shows the various *transport tracer properties* that are used in the current TransportTracers simulation.

```
Is_Tracer
- true                : all

Snk_Horiz:
- lat_zone           : aoa_nh
- all                : all others

Snk_Lats
- 30 50              : aoa_nh

Snk_Mode
- constant           : aoa, aoa_bl, aoa_nh
- efolding           : CH3I, CO_25
- none               : SF6
- halflife           : Be7, Be7s, Be10, Be10s

Snk_Period (days)
- 5                  : CH3I
- 25                 : CO_25
- 50                 : CO_50
- 90                 : e90, e90_n, e90_s
- 11742.8            : Pb210, Pb210s
- 5.5                : Rn222
- 53.3               : Be7, Be7s
- 5.84e8             : Be10, Be10s

Snk_Value
- 0                  : aoa, aoa_bl, aoa_nh
```

(continues on next page)

(continued from previous page)

```

Snk_Vert
- boundary_layer      : aoa_bl
- surface             : aoa, aoa_nh
- troposphere         : stOx
- all                 : all others

Src_Add
- false              : Passive, stOx, st80_25
- true               : all others

Src_Horiz
- lat_zone           : e90_n, e90_s, nh_5, nh_50
- all                : all others

Src_Lats
- [ 40.0,  91.0]      : e90_n
- [-91.0, -40.0]      : e90_s
- [ 30.0,  50.0]      : nh_5, nh_50

Src_Mode
- constant           : aoa, aoa_bl, aoa_nh, nh_50, nh_5, st80_25
- file2d             : CH3I, CO_25, CO_50, Rn222, SF6 - HEMCO
- file3d             : Be10, Be7 - HEMCO
- maintain_mixing_ratio : e_90, e90_n, e90_s
- decay_of_another_species : Pb210, Pb210s

Src_Unit
- ppbv              : e90, e90_n, e90_s, st80_25
- timestep           : aoa, aoa_bl, aoa_nh

Src_Value
- 1                 : aoa, aoa_bl, aoa_nh
- 100               : e90, e90_n, e90_s
- 200               : st80_25

Src_Vert
- all               : aoa, aoa_bl, aoa_nh, Pb210
- pressures         : st80_25
- stratosphere      : Be10s, Be7s, Pb210s, stOx
- surface           : all others (not specified when Src_Mode: HEMCO)

Src_Pressures
- [0, 80]           : st80_25

Units
- days              : aoa, aoa_bl, aoa_bl

```

24.1.7 Other properties

BackgroundVV

If a restart file does not contain an global initial concentration field for a species, GEOS-Chem will attempt to set the initial concentration (in vol vol^{-1} dry air) to the value specified in `BackgroundVV` globally. But if `BackgroundVV` has not been specified, GEOS-Chem will set the initial concentration for the species to $10^{-20} \text{vol vol}^{-1}$ dry air instead.

Note: Recent versions of GCHP may require that all initial conditions for all species to be used in a simulation be present in the restart file. See gchp.readthedocs.io for more information.

MP_SizeResAer

Indicates that the species is a size-resolved aerosol species (`true`), or isn't (`false`). Used only by simulations using either `APM` or `TOMAS` microphysics packages.

MP_SizeResNum

Indicates that the species is a size-resolved aerosol number (`true`), or isn't (`false`). Used only by simulations using either `APM` or `TOMAS` microphysics packages.

24.2 Access species properties in GEOS-Chem

In this section we will describe the derived types and objects that are used to store GEOS-Chem species properties. We will also describe how you can extract species properties from the GEOS-Chem Species Database when you create new GEOS-Chem code routines.

24.2.1 The Species derived type

The `Species` derived type (defined in module `Headers/species_mod.F90`) describes a complete set of properties for a single GEOS-Chem species. In addition to the fields mentioned in the preceding sections, the `Species` derived type also contains several species indices.

Table 1: Indices stored in the `Species` derived type

Index	Description
<code>ModelId</code>	Model species index
<code>AdvectId</code>	Advected species index
<code>AerosolId</code>	Aerosol species index
<code>DryAltId</code>	Dry dep species at altitude Id
<code>DryDepId</code>	Dry deposition species index
<code>GasSpcId</code>	Gas-phase species index
<code>HygGrthId</code>	Hygroscopic growth species index
<code>KppVarId</code>	KPP variable species index
<code>KppFixId</code>	KPP fixed species index
<code>KppSpcId</code>	KPP species index
<code>PhotolId</code>	Photolysis species index
<code>RadNuclId</code>	Radionuclide index
<code>TracerId</code>	Transport tracer index
<code>WetDepId</code>	Wet deposition index

24.2.2 The SpcPtr derived type

The *SpcPtr* derived type (also defined in Headers/species_mod.F90) describes a container for an object of type *Species*.

```
TYPE, PUBLIC :: SpcPtr
  TYPE(Species), POINTER :: Info    ! Single entry of Species Database
END TYPE SpcPtr
```

24.2.3 The GEOS-Chem Species Database object

The GEOS-Chem Species database is stored in the State_Chm%SpcData object. It describes an array, where each element of the array is of type *SpcPtr* (which is a container for an object of type type *Species*).

```
TYPE(SpcPtr), POINTER :: SpcData(:) ! GC Species database
```

24.2.4 Species index lookup with Ind_()

Use function Ind_() (in module Headers/state_chm_mod.F90) to look up species indices by name. For example:

```
SUBROUTINE MySub( ..., State_Chm, ... )

  USE State_Chm_Mod, ONLY : Ind_

  ! Local variables
  INTEGER :: id_O3, id_Br2, id_CO

  ! Find tracer indices with function the Ind_() function
  id_O3  = Ind_( 'O3' )
  id_Br2 = Ind_( 'Br2' )
  id_CO  = Ind_( 'CO' )

  ! Print tracer concentrations
  print*, 'O3 at (23,34,1) : ', State_Chm%Species(id_O3)%Conc(23,34,1)
  print*, 'Br2 at (23,34,1) : ', State_Chm%Species(id_Br2)%Conc(23,34,1)
  print*, 'CO at (23,34,1) : ', State_Chm%Species(id_CO)%Conc(23,34,1)

  ! Print the molecular weight of O3 (obtained from the Species Database object)
  print*, 'Mol wt of O3 [g]: ', State_Chm%SpcData(id_O3)%Info%MW_g

END SUBROUTINE MySub
```

Once you have obtained the species ID (aka ModelId) you can use that to access the individual fields in the Species Database object. In the example above, we use the species ID for O3 (stored in id_O3) to look up the molecular weight of O3 from the Species Database.

You may search for other model indices with Ind_() by passing an optional second argument:

```
! Position of HNO3 in the list of advected species
AdvectId = Ind_( 'HNO3', 'A' )

! Position of HNO3 in the list of gas-phase species
AdvectId = Ind_( 'HNO3', 'G' )
```

(continues on next page)

(continued from previous page)

```
! Position of HNO3 in the list of dry deposited species
DryDepId = Ind_( 'HNO3', 'D' )

! Position of HNO3 in the list of wet deposited species
WetDepId = Ind_( 'HNO3', 'W' )

! Position of HNO3 in the lists of fixed KPP, active, & overall KPP species
KppFixId = Ind_( 'HNO3', 'F' )
KppVarId = Ind_( 'HNO3', 'V' )
KppVarId = Ind_( 'HNO3', 'K' )

! Position of SALA in the list of hygroscopic growth species
HygGthId = Ind_( 'SALA', 'H' )

! Position of Pb210 in the list of radionuclide species
HygGthId = Ind_( 'Pb210', 'N' )

! Position of ACET in the list of photolysis species
PhotolId = Ind_( 'ACET', 'P' )
```

Ind_() will return -1 if a species does not belong to any of the above lists.

Tip: For maximum efficiency, we recommend that you use Ind_() to obtain the species indices during the initialization phase of a GEOS-Chem simulation. This will minimize the number of name-to-index lookup operations that need to be performed, thus reducing computational overhead.

Implementing the tip mentioned above:

```
MODULE MyModule

  IMPLICIT NONE
  . . .

  ! Species ID of CO. All subroutines in MyModule can refer to id_CO.
  INTEGER, PRIVATE :: id_CO

CONTAINS

  . . . other subroutines . . .

  SUBROUTINE Init_MyModule

    ! This subroutine only gets called at startup

    . . .

    ! Store ModelId in the global id_CO variable
    id_CO = Ind_('CO')

    . . .

  END SUBROUTINE Init_MyModule

END MODULE MyModule
```

24.2.5 Species lookup within a loop

If you need to access species properties from within a loop, it is better not to use the `Ind_()` function, as repeated name-to-index lookups will incur computational overhead. Instead, you can access the species properties directly from the GEOS-Chem Species Database object, as shown here.

```
SUBROUTINE MySub( ..., State_Chm, ... )

  !%% MySub is an example of species lookup within a loop %%

  ! Uses
  USE Precision_Mod
  USE State_Chm_Mod, ONLY : ChmState
  USE Species_Mod,    ONLY : Species

  ! Chemistry state object (which also holds the species database)
  TYPE(ChmState), INTENT(INOUT) :: State_Chm

  ! Local variables
  INTEGER                                :: N
  TYPE(Species), POINTER                :: ThisSpc
  INTEGER                                :: ModelId, DryDepId, WetDepId
  REAL(fp)                                :: Mw_g
  REAL(f8)                                :: Henry_K0, Henry_CR, Henry_pKa

  ! Loop over all species
  DO N = 1, State_Chm%nSpecies

    ! Point to the species database entry for this species
    ! (this makes the coding simpler)
    ThisSpc => State_Chm%SpcData(N)%Info

    ! Get species properties
    ModelId  = ThisSpc%ModelId
    DryDepId = ThisSpc%DryDepId
    WetDepId = ThisSpc%WetDepId
    MW_g     = ThisSpc%MW_g
    Henry_K0 = ThisSpc%Henry_K0
    Henry_CR = ThisSpc%Henry_CR
    Henry_pKa = ThisSpc%Henry_pKa

    IF ( ThisSpc%Is_Gas )
      ! ... The species is a gas-phase species
      ! ... so do something appropriate
    ELSE
      ! ... The species is an aerosol
      ! ... so do something else appropriate
    ENDIF

    IF ( ThisSpc%Is_Advected ) THEN
      ! ... The species is advected
      ! ... (i.e. undergoes transport, PBL mixing, cloud convection)
    ENDIF

    IF ( ThisSpc%Is_DryDep ) THEN
      ! ... The species is dry deposited
    ENDIF
```

(continues on next page)

(continued from previous page)

```
IF ( ThisSpc%Is_WetDep ) THEN
  ! ... The species is soluble and wet deposits
  ! ... it is also scavenged in convective updrafts
  ! ... it probably has defined Henry's law properties
ENDIF

... etc ...

! Free the pointer
ThisSpc => NULL()

ENDDO

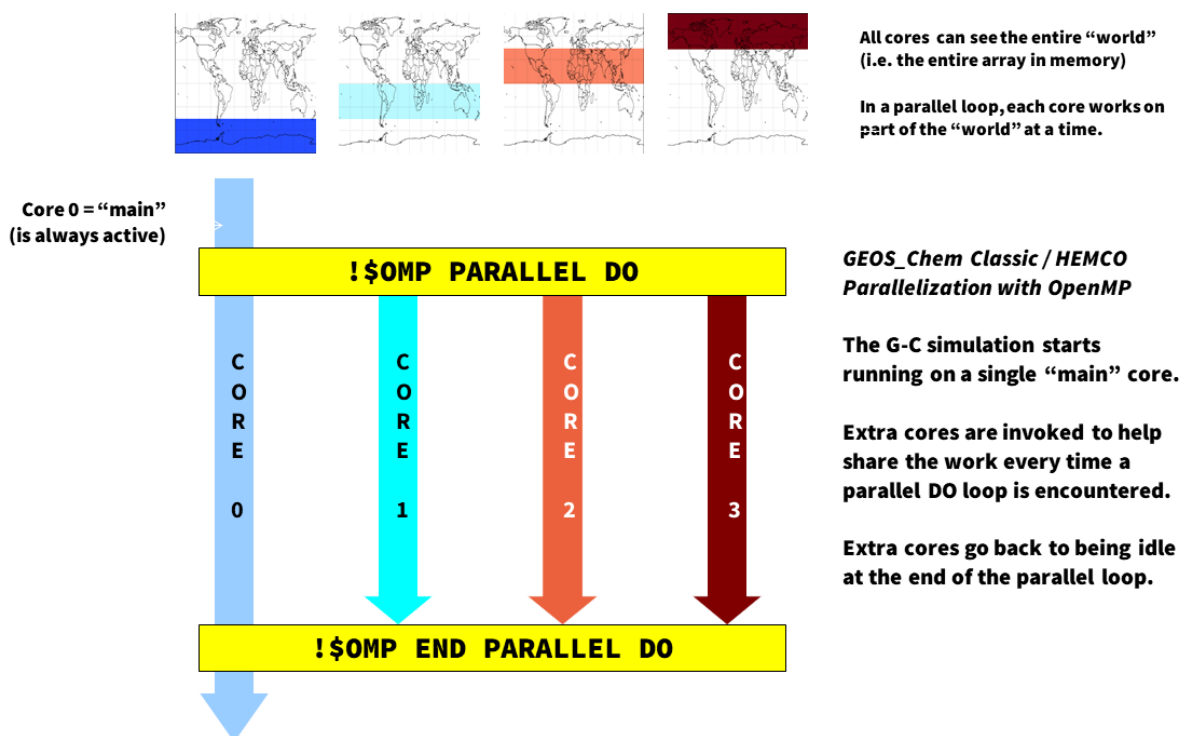
END SUBROUTINE MySub
```

PARALLELIZE GEOS-CHEM AND HEMCO SOURCE CODE

Single-node parallelization in **GEOS-Chem Classic** and **HEMCO** is achieved with **OpenMP**. OpenMP directives, which are included in every modern compiler, allow you to divide the work done in DO loops among several computational cores. In this Guide, you will learn more about how GEOS-Chem Classic and HEMCO utilize OpenMP.

25.1 Overview of OpenMP parallelization

Most GEOS-Chem and HEMCO arrays represent quantities on a geospatial grid (such as meteorological fields, species concentrations, production and loss rates, etc.). When we parallelize the GEOS-Chem and HEMCO source code, we give each computational core its own region of the “world” to work on, so to speak. However, all cores can see the entire “world” (i.e. the entire memory on the machine) at once, but is just restricted to working on its own region of the “world”.



It is important to remember that OpenMP is **loop-level parallelization**. That means that only commands within selected DO loops will execute in parallel. GEOS-Chem Classic and HEMCO (when running within GEOS-Chem Classic, or as the HEMCO standalone) start off on a single core (known as the “main core”). Upon entering a parallel

DO loop, other cores will be invoked to share the workload within the loop. At the end of the parallel DO loop, the other cores return to standby status and the execution continues only on the “main” core.

One restriction of using OpenMP parallelization is that simulations may use only as many cores that share by the same memory. In practice, this limits GEOS-Chem Classic and HEMCO standalone simulations to using 1 node (typically less than 64 cores) of a shared computer cluster.

We should also note that **GEOS-Chem High Performance** (aka **GCHP**) uses a *different type of parallelization (MPI)*. This allows GCHP to use hundreds or thousands of cores across several nodes of a computer cluster. We encourage you to consider using GCHP for hour high-resolution simulations.

25.2 Example using OpenMP directives

Consider the following nested loop that has been parallelized with OpenMP directives:

```
!$OMP PARALLEL DO           &
!$OMP SHARED( A             ) &
!$OMP PRIVATE( I, J, B      ) &
!$OMP COLLAPSE( 2           ) &
!$OMP SCHEDULE( DYNAMIC, 4 )
DO J = 1, NY
DO I = 1, NX
  B = A(I, J)
  A(I, J) = B * 2.0
ENDDO
ENDDO
!$OMP END PARALLEL DO
```

This loop will assign different (I, J) pairs to different computational cores. The more cores specified, the less time it will take to do the operation.

Let us now look at the important features of this loop.

!\$OMP PARALLEL DO

This is known as a **loop sentinel**. It tells the compiler that the following DO-loop is to be executed in parallel. The **clauses** following the sentinel specify further options for the parallelization. These clauses may be spread across multiple lines by using a continuation command (&) at the end of the line.

!\$OMP SHARED(A)

This clause tells the compiler that all computational cores can write to A simultaneously. This is OK because each core will receive a unique set of (I, J) pairs. Thus data corruption of the A array will not happen. We say that A is a **SHARED** variable.

Note: We recommend using the clause **!\$OMP DEFAULT(SHARED)**, which will declare all variables as shared, unless they are explicitly placed in an **!\$OMP PRIVATE** clause.

!\$OMP PRIVATE(I, J, B)

Because different cores will be handling different (I, J) pairs, each core needs its own private copy of variables I and J. The compiler creates these temporary copies of these variables in memory “under the hood”.

If the I and J variables were not declared **PRIVATE**, then all of the computational cores could simultaneously write to I and J. This would lead to data corruption. For the same reason, we must also place the variable B within the **!\$OMP PRIVATE** clause.

!\$OMP COLLAPSE(2)

By default, OpenMP will parallelize the outer loop in a set of nested loops. To gain more efficiency, we can

vectorize the loop. “Under the hood”, the compiler can convert the two nested loops over NX and NY into a single loop of size $NX * NY$, and then parallelize over the single loop. Because we wish to collapse 2 loops together, we use the `!$OMP COLLAPSE (2)` statement.

```
!$OMP SCHEDULE ( DYNAMIC, 4 )
```

Normally, OpenMP will evenly split the domain to be parallelized (i.e. (NX, NY)) evenly between the cores. But if some computations take longer than others (i.e. photochemistry at the day/night boundary), this static scheduling may be inefficient.

The `SCHEDULE (DYNAMIC, 4)` will send groups of 4 grid boxes to each core. As soon as a core finishes its work, it will immediately receive another group of 4 grid boxes. This can help to achieve better load balancing.

```
!$OMP END PARALLEL DO
```

This is a sentinel that declares the end of the parallel DO loop. It may be omitted. But we encourage you to include them, as defining both the beginning and end of a parallel loop is good programming style.

25.3 Environment variable settings for OpenMP

Please see [Set environment variables for parallelization](#) to learn which environment variables you must add to your login environment to control OpenMP parallelization.

25.4 OpenMP parallelization FAQ

Here are some frequently asked questions about parallelizing GEOS-Chem and HEMCO code with OpenMP:

25.4.1 How can I tell what should go into the !\$OMP PRIVATE clause?

Here is a good rule of thumb:

All variables that appear on the left side of an equals sign, and that have lower dimensionality than the dimensionality of the parallel loop must be placed in the `!$OMP PRIVATE` clause.

In the *example shown above*, I, J, and B are scalars, so their dimensionality is 0. But the parallelization occurs over two DO loops (`1 . . NY` and `1 . . NX`), so the dimensionality of the parallelization is 2. Thus I, J, and B must go inside the `!$OMP PRIVATE` clause.

Tip: You can also think of dimensionality as the number of indices a variable has. For example A has dimensionality 0, but `A (I)` has dimensionality 1, `A (I, J)` has dimensionality 2, etc.

25.4.2 Why do the !\$OMP statements begin with a comment character?

This is by design. In order to invoke the parallel processing commands, you must use a specific compiler command (such as `-openmp`, `-fopenmp`, or similar, depending on the compiler). If you omit these compiler switches, then the parallel processing directives will be considered as Fortran comments, and the associated DO-loops will be executed on a single core.

25.4.3 Do subroutine variables have to be declared PRIVATE?

Consider this subroutine:

```
SUBROUTINE mySub( X, Y, Z )

  ! Dummy variables for input
  REAL, INTENT(IN)  :: X, Y

  ! Dummy variable for output
  REAL, INTENT(OUT) :: Z

  ! Add X + Y to make Z
  Z = X + Y

END SUBROUTINE mySub
```

which is called from within a parallel loop:

```
INTEGER :: N
REAL    :: A, B, C

!$OMP PARALLEL DO           &
!$OMP DEFAULT( SHARED     ) &
!$OMP PRIVATE( N, A, B, C )
DO N = 1, nIterations

  ! Get inputs from some array
  A = Input(N,1)
  B = Input(N,2)

  ! Add A + B to make C
  CALL mySub( A, B, C )

  ! Save the output in an array
  Output(N) = C
ENDDO
!$OMP END PARALLEL DO
```

Using the *rule of thumb described above*, because N, A, B, and C are scalars (having dimensionality = 0), they must be placed in the !\$OMP PRIVATE clause.

But note that the variables X, Y, and Z do not need to be placed within a !\$OMP PRIVATE clause within subroutine mySub. This is because each core calls mySub in a separate thread of execution, and will create its own private copy of X, Y, and Z in memory.

25.4.4 What does the THREADPRIVATE statement do?

Let's modify the *above example* slightly. Let's now suppose that subroutine mySub from the prior example is now part of a Fortran-90 module, which looks like this:

```
MODULE myModule

  ! Module variable:
  REAL, PUBLIC :: Z
```

(continues on next page)

(continued from previous page)

CONTAINS

```

SUBROUTINE mySub( X, Y )

  ! Dummy variables for input
  REAL, INTENT(IN)  :: X, Y

  ! Add X + Y to make Z
  ! NOTE that Z is now a global variable
  Z = X + Y

END SUBROUTINE mySub

END MODULE myModule

```

Note that Z is now a global scalar variable with dimensionality = 0. Let's now use the same parallel loop (dimensionality = 1) as before:

```

! Get the Z variable from myModule
USE myModule, ONLY : Z

INTEGER :: N
REAL    :: A, B, C

!$OMP PARALLEL DO                &
!$OMP DEFAULT( SHARED          ) &
!$OMP PRIVATE( N, A, B, C )
DO N = 1, nIterations

  ! Get inputs from some array
  A = Input(N,1)
  B = Input(N,2)

  ! Add A + B to make C
  CALL mySub( A, B )

  ! Save the output in an array
  Output(N) = Z

ENDDO
!$OMP END PARALLEL DO

```

Because Z is now a global variable with lower dimensionality than the loop, we must try to place it within an !\$OMP PRIVATE clause. However, Z is defined in a different program unit than where the parallel loop occurs, so we cannot place it in an !\$OMP PRIVATE clause for the loop.

In this case we must place Z into an !\$OMP THREADPRIVATE clause within the module where it is declared, as shown below:

```

MODULE myModule

  ! Module variable:
  ! This is global and acts as if it were in a F77-style common block
  REAL, PUBLIC :: Z
  !$OMP THREADPRIVATE( Z )

  ... etc ...

```

This tells the computer to create a separate private copy of Z in memory for each core.

Important: When you place a variable into an `!$OMP PRIVATE` or `!$OMP THREADPRIVATE` clause, this means that the variable will have no meaning outside of the parallel loop where it is used. So you should not rely on using the value of `PRIVATE` or `THREADPRIVATE` variables elsewhere in your code.

Most of the time you won't have to use the `!$OMP THREADPRIVATE` statement. You may need to use it if you are trying to parallelize code that came from someone else.

25.4.5 Can I use pointers within an OpenMP parallel loop?

You may use pointer-based variables (including derived-type objects) within an OpenMP parallel loop. But you must make sure that you point to the target within the parallel loop section AND that you also nullify the pointer within the parallel loop section. For example:

INCORRECT:

```
! Declare variables
REAL, TARGET :: myArray(NX,NY)
REAL, POINTER :: myPtr (:)

! Declare an OpenMP parallel loop
!$OMP PARALLEL DO
!$OMP DEFAULT( SHARED
!$OMP PRIVATE( I, J, myPtr, ...etc... )
DO J = 1, NY
DO I = 1, NX

    ! Point to a variable.
    !This must be done in the parallel loop section.
    myPtr => myArray(:,J)

    . . . do other stuff . . .

ENDDO
!$OMP END PARALLEL DO

! Nullify the pointer.
! NOTE: This is incorrect because we nullify the pointer outside of the loop.
myPtr => NULL()
```

CORRECT:

```
! Declare variables
REAL, TARGET :: myArray(NX,NY)
REAL, POINTER :: myPtr (:)

! Declare an OpenMP parallel loop
!$OMP PARALLEL DO
!$OMP DEFAULT( SHARED
!$OMP PRIVATE( I, J, myPtr, ...etc... )
DO J = 1, NY
DO I = 1, NX

    ! Point to a variable.
```

(continues on next page)

(continued from previous page)

```

!This must be done in the parallel loop section.
myPtr => myArray(:,J)

. . . do other stuff . . .

! Nullify the pointer within the parallel loop
myPtr => NULL()

ENDDO
!$OMP END PARALLEL DO

```

In other words, pointers used in OpenMP parallel loops only have meaning within the parallel loop.

25.4.6 How many cores may I use for GEOS-Chem or HEMCO?

You can use as many computational cores as there are on a single node of your cluster. With [OpenMP parallelization](#), the restriction is that all of the cores have to see all the memory on the machine (or node of a larger machine). So if you have 32 cores on a single node, you can use them. We have shown that run times will continue to decrease (albeit asymptotically) when you increase the number of cores.

25.4.7 Why is GEOS-Chem is not using all the cores I requested?

The number of threads for an OpenMP simulation is determined by the environment variable `OMP_NUM_THREADS`. You must define `OMP_NUM_THREADS` in your [environment file](#) to specify the desired number of computational cores for your simulation. For the `bash` shell, use the command to request 8 cores:

```
export OMP_NUM_THREADS=8
```

25.5 MPI parallelization

The [OpenMP parallelization](#) used by GEOS-Chem Classic and HEMCO standalone is an example of **shared memory parallelization** (also known as **serial parallelization**). As we have seen, we are restricted to using a single node of a computer cluster. This is because all of the cores need to talk with all of the memory on the node.

On the other hand, MPI (Message Passing Interface) parallelization is an example of **distributed parallelization**. An MPI library installation is required for passing memory from one physical system to another (i.e. across nodes).

GEOS-Chem High Performance (GCHP) uses Earth System Model Framework (ESMF) and MAPL libraries to implement MPI parallelization. For detailed information, please see gchp.readthedocs.io.

RUN NESTED-GRID SIMULATIONS

A **nested-grid simulation** is a GEOS-Chem Classic simulation running at the native horizontal resolution of the GEOS-FP ($0.25^\circ \times 0.3125^\circ$) or MERRA-2 ($0.5^\circ \times 0.6125^\circ$) meteorology fields over a subset of the globe. Nested-grid simulations use boundary conditions for transport that are archived from a global simulation.

Follow these steps to set up a GEOS-Chem Classic nested-grid simulation:

26.1 Run a global simulation to create boundary conditions

26.1.1 1. Download the GEOS-Chem source code

Download the GEOS-Chem Classic source code by *following these instructions*.

26.1.2 2. Create a global simulation run directory

Create a run directory for your global simulation by executing these commands:

```
$ cd /path/to/GCClassic/run    # or whatever you named the source code directory
$ ./createRunDir.sh
```

and then follow the prompts.

Tip: A $4^\circ \times 5^\circ$ global simulation should be adequate for producing boundary condition output.

26.1.3 3. Activate the BoundaryConditions diagnostic collection

The BoundaryConditions diagnostic collection is deactivated by default in the *HISTORY.rc* file that ships with the run directory. Activate this collection by removing the comment character (#) as shown below.

```
COLLECTIONS: 'Restart',
             'SpeciesConc',
             ... etc ...
             #'BoundaryConditions',    <== Remove the # sign in front
::
```

The BoundaryConditions collection will save out instantaneous concentrations of advected species every three hours to daily files. You may change those settings by modifying the BoundaryConditions collection section in the *HISTORY.rc* file.

Tip: If you wish to save disk space, Use the `.LON_RANGE` and `.LAT_RANGE` to reduce the size of the region in which the boundary conditions will be saved. The region in which the boundary condition is archived should be a little larger than the nested-grid simulation window.

```
#=====
# %%%% THE BoundaryConditions COLLECTION %%%%
#
# GEOS-Chem boundary conditions for use in nested grid simulations
#
# Available for all simulations
#=====
BoundaryConditions.template:  '%y4%m2%d2_%h2%n2z.nc4',
BoundaryConditions.format:    'CFIO',
BoundaryConditions.frequency:  00000000 030000
BoundaryConditions.duration:   00000001 000000
BoundaryConditions.mode:       'instantaneous'
BoundaryConditions.LON_RANGE:  -130.0 -60.0,
BoundaryConditions.LAT_RANGE:  10.0 60.0,
BoundaryConditions.fields:     'SpeciesBC_?ADV?          ', 'GIGCchem',
::
```

26.1.4 4. Configure the global simulation

Configure your global simulation by *changing settings in the relevant configuration files*. If you do not need the output from your global simulation, you may choose to turn off most of the diagnostic output in `HISTORY.rc` and `HEMCO_Diagn.rc`.

Tip: Turn off most diagnostic output in the *HISTORY.rc* and *HEMCO_Diagn.rc* files. This will minimize the run time and reduce the size of diagnostic output.

26.1.5 5. Compile GEOS-Chem and run the global simulation

Follow the steps outlined in these sections to compile and run your GEOS-Chem global simulation.

1. *Compile the source code*
2. *Download input data* (e.g. do a dry-run and download data if necessary)
3. *Run your simulation*

Once your global simulation finishes, the boundary conditions files will be placed into the `OutputDir` subdirectory of your run directory. You should see files named `GEOSChem.BoundaryConditions.YYYYMMDD_0000z.nc4` (where `YYYYMMDD` are replaced by the simulation date) begin to appear in your run directory as your simulation runs. You will need to tell your nested-grid simulation where to find these files.

26.2 Set up your nested grid run directory

26.2.1 1. Create a nested-grid simulation run directory

Using the same GEOS-Chem Classic source code directory that *we downloaded above* follow these steps to *create a run directory* for your nested-grid simulation.

```
$ cd /path/to/GCClassic/run # or whatever you named the source code directory
$ ./createRunDir.sh
```

Select the native resolution corresponding to your choice of meteorology. You will then be asked to specify which nested region you would like to use.

26.2.2 2. Configure your nested-grid simulation

Check the *run-directory configuration files* to make sure that you have the same chemistry, emissions, transport, etc. options selected as in the global simulation.

In *HEMCO_Config.rc*, make sure the GC_BCs option is set to true and update the BC_ entry to point to your boundary condition files.

```
# ExtNr ExtName          on/off Species
0      Base             : on      *
# ----- RESTART FIELDS -----
--> GC_RESTART           :         true
--> GC_BCs               :         true  <== make sure this is true
--> HEMCO_RESTART        :         true
...
#=====
# --- GEOS-Chem boundary condition file ---
#=====
((GC_BCs
* BC_ /path/to/your/GEOSChem.BoundaryConditions.$YYYY$MM$DD_$HH$MNz.nc4 SpeciesBC_?
→ADV? 1980-2023/1-12/1-31/0-23 RFY xyz 1 * - 1 1
))GC_BCs
```

Activate your preferred diagnostics by changing the relevant settings in these configuration files:

1. *HISTORY.rc*
2. *HEMCO_Diagn.rc*
3. *Planeflight.dat.YYYYMMDD*
4. *The ObsPack menu of geoschem_config.yml*

26.2.3 3. Copy the executable to the nested-grid run directory

You do not have to recompile GEOS-Chem Classic when changing grids. Therefore, you can copy the `gcclassic` executable from your *global simulation* run directory to your nested-grid run directory.

26.2.4 4. Run the nested-grid simulation

Follow the steps outlined in these sections to run your nested-grid simulation.

1. *Download input data* (e.g. do a dry-run and download data if necessary)
2. *Run your simulation*

26.3 Frequently asked questions

26.3.1 Can I run nested GEOS-Chem simulations on the AWS cloud?

Yes, you can run the nested grid simulations on AWS cloud. Please see the [Running GEOS-Chem on AWS cloud online tutorial](#) and contact the [GEOS-Chem Support Team](#) with any questions.

26.3.2 Can I save out boundary conditions for more than one nested grid in the same global run?

We recommend that you *generate boundary conditions* over the entire global domain (at 4° x 5° or 2° x 2.5°). Then these boundary conditions can be used as input to simulations on different nested domains.

26.3.3 How can I find which data are available for nested grid simulations?

You will download meteorology and emissions data from one of the *GEOS-Chem data portals*. You can browse the WashU data portal (<http://geoschemdata.wustl.edu/ExtData>) to see if the data you need are available.

26.3.4 Where can I find out more info about nested grid errors?

Please see the following Supplemental Guides:

1. *Understand what error messages mean*
2. *Debug GEOS-Chem and HEMCO errors*

26.3.5 I noticed abnormal concentrations at boundaries of the nested region. Is that normal?

If you see high tracer concentrations right at the boundary of your nested grid region, then this may be normal.

For nested grid simulations, we have to leave a “buffer zone” (i.e. typically 3 boxes along each boundary) in which the TPCORE advection is not applied. However, all other operations (chemistry, wetdep, drydep, convection, PBL mixing) will be applied. Therefore, in the “buffer zone”, the concentrations will not be realistic because the advection is not allowed to transport the tracer out of these boxes.

In any case, the tracer concentrations in the “buffer zone” will get overwritten by the 2° x 2.5° or 4° x 5° boundary conditions at the specified time (usually every 3h).

Attention: You should exclude the boxes in the “buffer zone” from your scientific analysis.

The following diagram illustrates this:

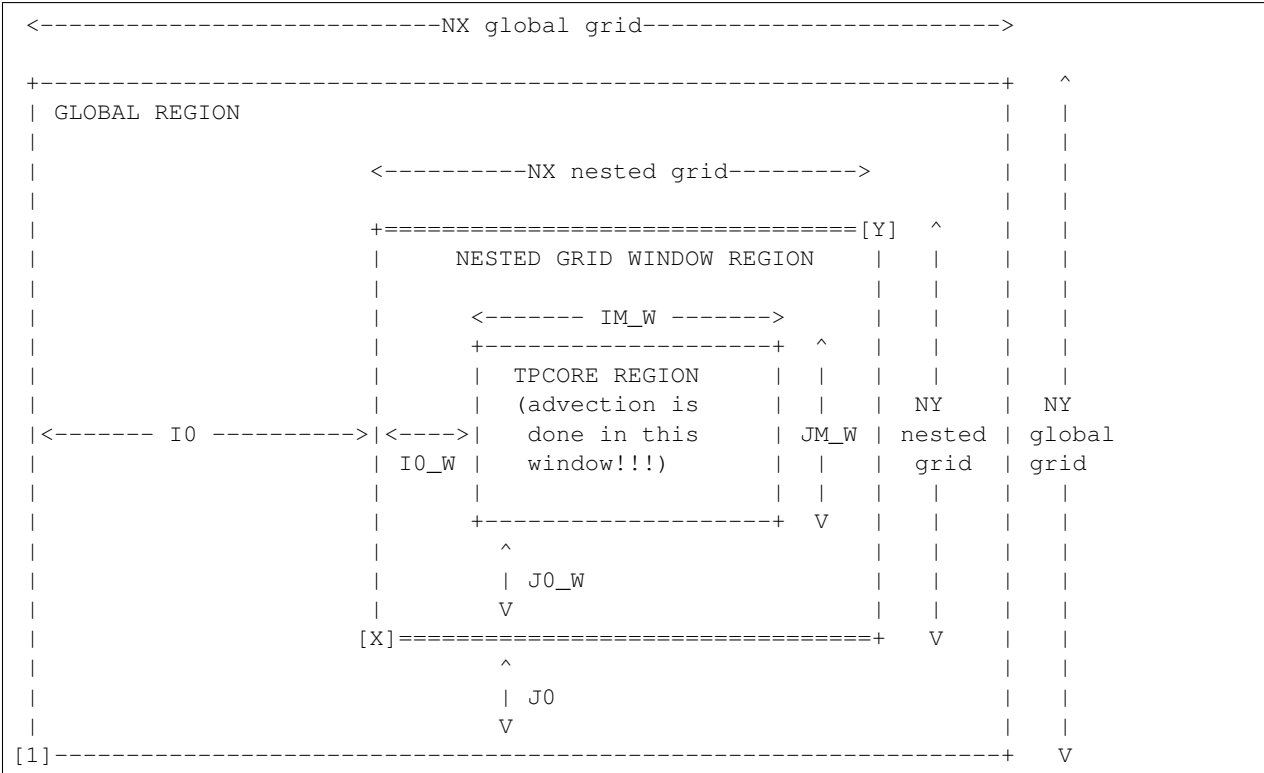


Diagram notes:

1. The outermost box (GLOBAL REGION) is the global grid size. This region has N_X global grid boxes in longitude and N_Y global grid boxes in latitude. The origin of the GLOBAL REGION is at the south pole, at the lower left-hand corner (point [1]).
2. The next innermost box (NESTED GRID WINDOW REGION) is the nested-grid window. This region has N_X nested grid boxes in longitude and N_Y nested grid boxes in latitude. This is the size of the trimmed met fields that will be used for a “nested-grid” simulation.
3. The innermost region TPCORE REGION is the actual area in which **TPCORE** advection will be performed. Note that this region is smaller than the NESTED GRID WINDOW REGION. It is set up this way since a cushion of grid boxes is needed for boundary conditions.
4. I_0 is the longitude offset (# of boxes) and J_0 is the latitude offset (# of boxes) which translate between the GLOBAL REGION and the NESTED GRID WINDOW REGION.
5. I_0_W is the longitude offset (# of boxes), and J_0_W is the latitude offset (# of boxes) which translate between the NESTED GRID WINDOW REGION and the TPCORE REGION. These define the thickness of the buffer zone mentioned above.
6. The lower left-hand corner of the NESTED GRID WINDOW REGION (point [X]) has longitude and latitude indices ($I1_W$, $J1_W$). Similarly, the upper right-hand corner (point [Y]) has longitude and latitude indices ($I2_W$, $J2_W$).

7. Note that if `I0=0`, `J0=0`, `I0_W=0`, `J0_W=0`, `NX` nested grid = `NX` global grid, `NY` nested grid = `NY` global grid specifies a global simulation. In this case the `NESTED GRID WINDOW REGION` totally coincides with the `GLOBAL REGION`.
8. In order for the nested-grid simulation to work we must save out concentrations over the `NESTED GRID WINDOW REGION` from a coarse model (e.g. $2^\circ \times 2.5^\circ$ or $4^\circ \times 5^\circ$). These concentrations are copied along the edges of the `NESTED GRID WINDOW REGION` and are thus used as boundary conditions for **TPCORE**.

UPDATE CHEMICAL MECHANISMS WITH KPP

This Guide demonstrates how you can use [The Kinetic PreProcessor \(aka KPP\)](#) to translate a chemical mechanism specification in plain text format to highly-optimized Fortran90 code for use with GEOS-Chem:

Attention: You must use at least [KPP 3.0.0](#) with the current GEOS-Chem release series.

27.1 Using KPP: Quick start

27.1.1 1. Navigate to the KPP/custom folder within GEOS-Chem

The `KPP/custom` folder is intended for building customized mechanisms. (The standard mechanisms that ship with GEOS-Chem are contained in other folders named `KPP/fullchem` and `KPP/Hg`, but we will leave these alone.)

If you are using GEOS-Chem “Classic”, type:

```
$ cd GCClassic/src/GEOS-Chem/KPP/custom
```

or if you are using GCHP, type:

```
$ cd GCHP/GCHP_GridComp/GEOSChem_GridComp/geos-chem/KPP/custom
```

27.1.2 2. Edit the chemical mechanism configuration files

The `KPP/custom` folder contains sample chemical mechanism specification files (*custom.eqn* and *custom.kpp*). These files define the chemical mechanism and are copies of the default **fullchem** mechanism configuration files found in the `KPP/fullchem` folder. (For a complete description of KPP configuration files, please see the documentation at kpp.readthedocs.io.)

You can edit these *custom.eqn* and *custom.kpp* configuration files to define your own custom mechanism (cf. *Using KPP: Reference section* for details).

Important: We recommend always building a custom mechanism from the `KPP/custom` folder, and to leave the other folders untouched. This will allow you to validate your modified mechanism against one of the standard mechanisms that ship with GEOS-Chem.

custom.eqn

The `custom.eqn` configuration file contains:

- List of active species
- List of inactive species
- Gas-phase reactions
- Heterogeneous reactions
- Photolysis reactions

custom.kpp

The `custom.kpp` configuration file is the main configuration file. It contains:

- Solver options
- Production and loss family definitions
- Functions to compute reaction rates
- Global definitions
- An **#INCLUDE** `custom.eqn` command, which tells **KPP** to look for chemical reaction definitions in *custom.eqn*.

Important: The symbolic link `gckpp.kpp` points to `custom.kpp`. This is necessary in order to generate Fortran files with the the naming convention `gckpp*.F90`.

27.1.3 3. Run the `build_mechanism.sh` script

Once you are satisfied with your custom mechanism specification you may now use **KPP** to build the source code files for GEOS-Chem.

Return to the top-level **KPP** folder from `KPP/custom`:

```
$ cd ..
```

There you will find a script named `build_mechanism.sh`, which is the driver script for running **KPP**. Execute the script as follows:

```
$ ./build_mechanism.sh custom
```

This will run the **KPP** executable (located in the folder `$KPP_HOME/bin`) `custom.kpp` configuration file (via symbolic link `gckpp.kpp`). It also runs a python script to generate code for the OH reactivity diagnostic. You should see output similar to this:

```
This is KPP-X.Y.Z.

KPP is parsing the equation file.
KPP is computing Jacobian sparsity structure.
KPP is starting the code generation.
KPP is initializing the code generation.
KPP is generating the monitor data:
```

(continues on next page)

(continued from previous page)

```

- gckpp_Monitor
KPP is generating the utility data:
- gckpp_Util
KPP is generating the global declarations:
- gckpp_Main
KPP is generating the ODE function:
- gckpp_Function
KPP is generating the ODE Jacobian:
- gckpp_Jacobian
- gckpp_JacobianSP
KPP is generating the linear algebra routines:
- gckpp_LinearAlgebra
KPP is generating the utility functions:
- gckpp_Util
KPP is generating the rate laws:
- gckpp_Rates
KPP is generating the parameters:
- gckpp_Parameters
KPP is generating the global data:
- gckpp_Global
KPP is generating the driver from none.f90:
- gckpp_Main
KPP is starting the code post-processing.

KPP has succesfully created the model "gckpp".

Reactivity consists of xxx reactions      # NOTE: xxx will be replaced by the actual_
↪number
Written to gckpp_Util.F90

```

where X.Y.Z denotes the **KPP** version that you are using.

If this process is successful, the custom folder will have several new files starting with gckpp:

```

$ ls gckpp*
gckpp_Function.F90    gckpp_Jacobian.F90    gckpp.log            gckpp_Precision.
↪F90
gckpp_Global.F90     gckpp_JacobianSP.F90  gckpp_Model.F90      gckpp_Rates.F90
gckpp_Initialize.F90 gckpp.kpp@            gckpp_Monitor.F90    gckpp_Util.F90
gckpp_Integrator.F90 gckpp_LinearAlgebra.F90 gckpp_Parameters.F90

```

The gckpp*.F90 files contain optimized Fortran-90 instructions for solving the chemical mechanism that you have specified. The gckpp.log file is a human-readable description of the mechanism. Also, gckpp.kpp is a symbolic link to the custom.kpp file.

A complete description of these KPP-generated files at kpp.readthedocs.io.

27.1.4 4. Recompile GEOS-Chem with your custom mechanism

GEOS-Chem will always use the default mechanism (which is named `fullchem`). To tell GEOS-Chem to use the custom mechanism instead, follow these steps.

Tip: GEOS-Chem Classic run directories have a subdirectory named `build` in which you can configure and build GEOS-Chem. If you don't have a build directory, you can add one to your run directory with `mkdir build`.

From the build directory, type:

```
$ cmake ../CodeDir -DMECH=custom -DRUNDIR=..
```

You should see output similar to this written to the screen:

```
-- General settings:
* MECH:  fullchem  carbon  Hg  **custom**
```

This confirms that the custom mechanism has been selected.

Once you have configured **GEOS-Chem** to use the `custom` mechanism, you may build the executable. Type:

```
$ make -j
$ make -j install
```

The executable file (`gcclassic` or `gchp`, depending on which mode of GEOS-Chem that you are using) will be placed in the run directory.

27.2 Using KPP: Reference section

27.2.1 Adding species to a mechanism

List chemically-active (aka variable) species in the `#DEFVAR` section of `custom.eqn`, as shown below:

```
#DEFVAR
A3O2      = IGNORE; {CH3CH2CH2OO; Primary RO2 from C3H8}
ACET      = IGNORE; {CH3C(O)CH3; Acetone}
ACTA      = IGNORE; {CH3C(O)OH; Acetic acid}
...etc ...
```

The `IGNORE` tells KPP not to perform mass-balance checks, which would make GEOS-Chem execute more slowly.

List species whose concentrations do not change in the `#DEFFIX` section of `custom.eqn`, as shown below:

```
#DEFFIX
H2        = IGNORE; {H2; Molecular hydrogen}
N2        = IGNORE; {N2; Molecular nitrogen}
O2        = IGNORE; {O2; Molecular oxygen}
... etc ...
```

Species may be listed in any order, but we have found it convenient to list them alphabetically.

27.2.2 Adding reactions to a mechanism

Gas-phase reactions

List gas-phase reactions first in the `#EQUATIONS` section of `custom.eqn`.

```
#EQUATIONS
//
// Gas-phase reactions
//
...skipping over the comment header...
//
O3 + NO = NO2 + O2 : GCARR_ac(3.00E-12, -1500.0);
O3 + OH = HO2 + O2 : GCARR_ac(1.70E-12, -940.0);
O3 + HO2 = OH + O2 + O2 : GCARR_ac(1.00E-14, -490.0);
O3 + NO2 = O2 + NO3 : GCARR_ac(1.20E-13, -2450.0);
... etc ...
```

Gas-phase reactions: General form

No matter what reaction is being added, the general procedure is the same. A new line must be added to `custom.eqn` of the following form:

```
A + B = C + 2.000D : RATE_LAW_FUNCTION(ARG_A, ARG_B ...);
```

The denotes the reactants (A and B) as well as the products (C and D) of the reaction. If exactly one molecule is consumed or produced, then the factor can be omitted; otherwise the number of molecules consumed or produced should be specified with at least 1 decimal place of accuracy. The final section, between the colon and semi-colon, specifies the function `RATE_LAW_FUNCTION` and its arguments which will be used to calculate the reaction rate constant k . Rate-law functions are specified in the `custom.kpp` file.

For an equation such as the one above, the overall rate at which the reaction will proceed is determined by $k[A][B]$. However, if the reaction rate does not depend on the concentration of A or B , you may write it with a constant value, such as:

```
A + B = C + 2.000D : 8.95d-17
```

This will save the overhead of a function call.

Rates for two-body reactions according to the Arrhenius law

For many reactions, the calculation of k follows the Arrhenius law:

```
k = a0 * ( 300 / TEMP )**b0 * EXP( c0 / TEMP )
```

Important: In relation to Arrhenius parameters that you may find in scientific literature, a_0 represents the A term and c_0 represents $-E/R$ (not E/R , which is usually listed).

For example, the [JPL chemical data evaluation](#), (Feb 2017) specifies that the reaction $\text{O}_3 + \text{NO}$ produces NO_2 and O_2 , and its Arrhenius parameters are $A = 3.0 \times 10^{-12}$ and $E/R = 1500$. To use the Arrhenius formulation above, we must specify $a_0 = 3.0e - 12$ and $c_0 = -1500$.

To specify a two-body reaction whose rate follows the Arrhenius law, you can use the GCARR rate-law function, which is defined in `gckpp.kpp`. For example, the entry for the $O_3 + NO = NO_2 + O_2$ reaction can be written as in `custom.eqn` as:

```
O3 + NO = NO2 + O2 : GCARR(3.00E12, 0.0, -1500.0);
```

Other rate-law functions

The `gckpp.kpp` file contains other rate law functions, such as those required for three-body, pressure-dependent reactions. Any rate function which is to be referenced in the `custom.eqn` file must be available in `gckpp.kpp` prior to building the reaction mechanism.

Making your rate law functions computationally efficient

We recommend writing your rate-law functions so as to avoid explicitly casting variables from `REAL*4` to `REAL*8`. Code that looks like this:

```
REAL, INTENT(IN) :: A0, B0, C0
rate = DBLE(A0) + ( 300.0 / TEMP )**DBLE(B0) + EXP( DBLE(C0) / TEMP )
```

Can be rewritten as:

```
REAL(kind=dp), INTENT(IN) :: A0, B0, C0
rate = A0 + ( 300.0d0 / TEMP )**B0 + EXP( C0 / TEMP )
```

Not only do casts lead to a loss of precision, but each cast takes a few CPU clock cycles to execute. Because these rate-law functions are called for each cell in the chemistry grid, wasted clock cycles can accumulate into a noticeable slowdown in execution.

You can also make your rate-law functions more efficient if you rewrite them to avoid computing terms that evaluate to 1. We saw above (cf. [Rates for two-body reactions according to the Arrhenius law](#)) that the rate of the reaction $O_3 + NO = NO_2 + O_2$ can be computed according to the Arrhenius law. But because `b0 = 0`, term `(300 / TEMP) ** b0` evaluates to 1. We can therefore rewrite the computation of the reaction rate as:

```
k = 3.0x10^-12 + EXP( 1500 / TEMP )
```

Tip: The `EXP()` and `**` mathematical operations are among the most costly in terms of CPU clock cycles. Avoid calling them whenever necessary.

A recommended implementation would be to create separate rate-law functions that take different arguments depending on which parameters are nonzero. For example, the Arrhenius law function GCARR can be split into multiple functions:

1. `GCARR_abc(a0, b0, c0)`: Use when `a0 > 0` and `b0 > 0` and `c0 > 0`
2. `GCARR_ab(a0, b0)`: Use when `a0 > 0` and `b0 > 0`
3. `GCARR_ac(a0, c0)`: Use when `a0 > 0` and `c0 > 0`

Thus we can write the $O_3 + NO$ reaction in `custom.eqn` as:

```
O3 + NO = NO2 + O2 : GCARR_ac(3.00d12, -1500.0d0);
```

using the rate law function for when both `a0 > 0` and `c0 > 0`.

27.2.3 Heterogeneous reactions

List heterogeneous reactions after all of the gas-phase reactions in `custom.eqn`, according to the format below:

```
//
// Heterogeneous reactions
//
HO2 = O2 :                                HO2uptk1stOrd( State_Het );
↳{2013/03/22; Paulot2009; FP,EAM,JMAO,MJE}
NO2 = 0.500HNO3 + 0.500HNO2 :            NO2uptk1stOrdAndCloud( State_Het );
NO3 = HNO3 :                             NO3uptk1stOrdAndCloud( State_Het );
NO3 = NIT :                             NO3hypsIsClonSALA( State_Het );
↳{2018/03/16; XW}
... etc ...
```

A simple example is uptake of HO2, specified as

```
HO2 = H2O : HO2uptk1stOrd( State_Het );
```

Note: KPP requires that each reaction have at least one product. In order to satisfy this requirement, you might need to set the product of your heterogeneous reaction to a dummy product or a fixed species (i.e. one whose concentration does not change with time).

The rate law function `NO2uptk1stOrd` is contained in the Fortran module `KPP/fullchem/fullchem_RateLawFuncs.F90`, which is symbolically linked to the `custom` folder. The `fullchem_RateLawFuncs.F90` file is inlined into `gckpp_Rates.F90` so that it can be used within the custom mechanism.

To implement an additional heterogeneous reaction, the rate calculation must be added to the `KPP/custom/custom.eqn` file. Rate calculations may be specified as mathematical expressions (using any of the variables contained in the `gckpp_Global.F90`)

```
SPC1 + SPC2 = SPC3 + SPC4: 8.0e-13 * TEMP_OVER_K300; {Example}
```

or you may define a new rate law function in the `fullchem_RateLawFuncs.F90` such as:

```
SPC1 + SPC2 = SPC3 + SPC4: myNewRateFunction( State_Het ); {Example}
```

27.2.4 Photolysis reactions

List photolysis reactions after the heterogeneous reactions, as shown below.

```
//
// Photolysis reactions
//
O3 + hv = O + O2 :                PHOTOL(2);        {2014/02/03; Eastham2014;
↳ SDE}
O3 + hv = O1D + O2 :             PHOTOL(3);        {2014/02/03; Eastham2014;
↳ SDE}
O2 + hv = 2.0000 :                PHOTOL(1);        {2014/02/03; Eastham2014;
↳ SDE}
... etc ...
```

(continues on next page)

(continued from previous page)

```

NO3 + hv = NO2 + O :          PHOTOL(12);      {2014/02/03; Eastham2014;
↪ SDE}
... etc ...

```

A photolysis reaction can be specified by giving the correct index of the PHOTOL array. This index can be determined by inspecting the file `FJX_j2j.dat`.

Tip: See the photolysis section of `geoschem_config.yml` to determine the folder in which `FJX_j2j.dat` is located.

For example, one branch of the NO_3 photolysis reaction is specified in the `custom.eqn` file as

```
NO3 + hv = NO2 + O : PHOTOL(12)
```

Referring back to `FJX_j2j.dat` shows that reaction 12, as specified by the left-most index, is indeed $\text{NO}_3 = \text{NO}_2 + \text{O}$:

12	NO3	PHOTON	NO2	O	0.886	/NO3	/
----	-----	--------	-----	---	-------	------	---

If your reaction is not already in `FJX_j2j.dat`, you may add it there. You may also need to modify `FJX_spec.dat` (in the same folder as `FJX_j2j.dat`) to include cross-sections for your species. Note that if you add new reactions to `FJX_j2j.dat` you will also need to set the parameter `JVN_` in GEOS-Chem module `Headers/CMN_FJX_MOD.F90` to match the total number of entries.

If your reaction involves new cross section data, you will need to follow an additional set of steps. Specifically, you will need to:

1. Estimate the cross section of each wavelength bin (using the correlated-k method), and
2. Add this data to the `FJX_spec.dat` file.

For the first step, you can use tools already available on the Prather research group website. To generate the cross-sections used by Fast-JX, download the file [UCI_fastJ_addX_73cx.tar.gz](#). You can then simply add your data to `FJX_spec.dat` and refer to it in `FJX_j2j.dat` as specified above. The following then describes how to generate a new set of cross-section data for the example of some new species MEKR:

To generate the photolysis cross sections of a new species, come up with some unique name which you will use to refer to it in the `FJX_j2j.dat` and `FJX_spec.dat` files - e.g. MEKR. You will need to copy one of the `addX_*.f` routines and make your own (say, `addX_MEKR.f`). Your edited version will need to read in whatever cross section data you have available, and you'll need to decide how to handle out-of-range information - this is particularly crucial if your cross section data is not defined in the visible wavelengths, as there have been some nasty problems in the past caused by implicitly assuming that the XS can be extrapolated (I would recommend buffering your data with zero values at the exact limits of your data as a conservative first guess). Then you need to compile that as a standalone code and run it; this will spit out a file fragment containing the aggregated 18-bin cross sections, based on a combination of your measured/calculated XS data and the non-contiguous bin subranges used by Fast-JX. Once that data has been generated, just add it to `FJX_spec.dat` and refer to it as above. There are examples in the `addX` files of how to deal with variations of cross section with temperature or pressure, but the main takeaway is that you will generate multiple cross section entries to be added to `FJX_spec.dat` with the same name.

Important: If your cross section data varies as a function of temperature AND pressure, you need to do something a little different. The acetone XS documentation shows one possible way to handle this; Fast-JX currently interpolates over either T or P, but not both, so if your data varies over both simultaneously then this will take some thought. The general idea seems to be that one determines which dependence is more important and uses that to generate a set of 3 cross sections (for interpolation), assuming values for the unused variable based on the standard atmosphere.

27.2.5 Adding production and loss families to a mechanism

Certain common families (e.g. PO_x , LO_x) have been pre-defined for you. You will find the family definitions near the top of the `custom.kpp` file (which is symbolically linked to `gckpp.kpp`):

```
#FAMILIES
POx : O3 + NO2 + 2NO3 + PAN + PPN + MPAN + HNO4 + 3N2O5 + HNO3 + BrO + HOBr + BrNO2 +
↪ 2BrNO3 + MPN + ETHLN + MVKN + MCRHN + MCRHNB + PROPNN + R4N2 + PRN1 + PRPN + R4N1 +
↪ HONIT + MONITS + MONITU + OLND + OLNN + IHN1 + IHN2 + IHN3 + IHN4 + INPB + INPD +
↪ ICN + 2IDN + ITCN + ITHN + ISOPNOO1 + ISOPNOO2 + INO2B + INO2D + INA + IDHNBOO +
↪ IDHNDOO1 + IDHNDOO2 + IHPNBOO + IHPNDOO + ICNOO + 2IDNOO + MACRNO2 + ClO + HOC1 +
↪ ClNO2 + 2ClNO3 + 2Cl2O2 + 2OC1O + O + O1D + IO + HOI + IONO + 2IONO2 + 2OIO + 2I2O2
↪ + 3I2O3 + 4I2O4;
LOx : O3 + NO2 + 2NO3 + PAN + PPN + MPAN + HNO4 + 3N2O5 + HNO3 + BrO + HOBr + BrNO2 +
↪ 2BrNO3 + MPN + ETHLN + MVKN + MCRHN + MCRHNB + PROPNN + R4N2 + PRN1 + PRPN + R4N1 +
↪ HONIT + MONITS + MONITU + OLND + OLNN + IHN1 + IHN2 + IHN3 + IHN4 + INPB + INPD +
↪ ICN + 2IDN + ITCN + ITHN + ISOPNOO1 + ISOPNOO2 + INO2B + INO2D + INA + IDHNBOO +
↪ IDHNDOO1 + IDHNDOO2 + IHPNBOO + IHPNDOO + ICNOO + 2IDNOO + MACRNO2 + ClO + HOC1 +
↪ ClNO2 + 2ClNO3 + 2Cl2O2 + 2OC1O + O + O1D + IO + HOI + IONO + 2IONO2 + 2OIO + 2I2O2
↪ + 3I2O3 + 4I2O4;
PCO : CO;
LCO : CO;
PSO4 : SO4;
LCH4 : CH4;
PH2O2 : H2O2;
```

Note: The PO_x , LO_x , PCO , and LCO families are used for computing budgets in the GEOS-Chem benchmark simulations. $PSO4$ is required for simulations using [TOMAS aerosol microphysics](#).

To add a new prod/loss family, add a new line to the `#FAMILIES` section with the format

```
FAM_NAME : MEMBER_1 + MEMBER_2 + ... + MEMBER_N;
```

The family name must start with P or L to indicate whether KPP should calculate a production or a loss rate. You will also need to make a corresponding update to the [GEOS-Chem species database](#) (`species_database.yml`) in order to define the `FullName`, `Is_Gas`, and `MW_g`, and attributes. For example, the entries for family species LCO and PCO are:

```
LCO:
  FullName: Dummy species to track loss rate of CO
  Is_Gas: true
  MW_g: 28.01
PCO:
  FullName: Dummy species to track production rate of CO
  Is_Gas: true
  MW_g: 28.01
```

The maximum number of families allowed by KPP is currently set to 300. Depending on how many prod/loss families you add, you may need to increase that to a larger number to avoid errors in KPP. You can change the number for `MAX_FAMILIES` in `KPP/kpp-code/src/gdata.h` and then [rebuild the KPP executable](#).

```
// - Many limits can be changed here by adjusting the MAX_* constants
// - To increase the max size of inlined code (F90_GLOBAL etc.),
//   change MAX_INLINE in scan.h.
//
```

(continues on next page)

(continued from previous page)

```
//  NOTES:
//  -----
//  (1) Note: MAX_EQN or MAX_SPECIES over 1023 causes a seg fault in CI build
//      -- Lucas Estrada, 10/13/2021
//
//  (2) MacOS has a hard limit of 65532 bytes for stack memory. To make
//      sure that you are using this max amount of stack memory, add
//      "ulimit -s 65532" in your .bashrc or .bash_aliases script. We must
//      also set smaller limits for MAX_EQN and MAX_SPECIES here so that we
//      do not exceed the available stack memory (which will result in the
//      infamous "Segmentation fault 11" error). If you are still having
//      problems on MacOS then consider reducing MAX_EQN and MAX_SPECIES
//      to smaller values than are listed below.
//      -- Bob Yantosca (03 May 2022)
#ifdef MACOS
#define MAX_EQN      2000      // Max number of equations (MacOS only)
#define MAX_SPECIES  1000      // Max number of species   (MacOS only)
#else
#define MAX_EQN      11000     // Max number of equations
#define MAX_SPECIES  6000     // Max number of species
#endif
#define MAX_SPNAME    30      // Max char length of species name
#define MAX_IVAL      40      // Max char length of species ID ?
#define MAX_EQNTAG    32      // Max length of equation ID in eqn file
#define MAX_K         1000     // Max length of rate expression in eqn file
#define MAX_ATOMS     10      // Max number of atoms
#define MAX_ATNAME    10      // Max char length of atom name
#define MAX_ATNR      250     // Max number of atom tables
#define MAX_PATH      300     // Max char length of directory paths
#define MAX_FILES     20      // Max number of files to open
#define MAX_FAMILIES  300     // Max number of family definitions
#define MAX_MEMBERS   150     // Max number of family members
#define MAX_EQNLEN    300     // Max char length of equations
#define MAX_EQNLEN    200
```

Important: When adding a prod/loss family or changing any of the other settings in `gckpp.kpp`, you must *re-run KPP* to produce new Fortran90 files for GEOS-Chem.

Production and loss families are archived via the HISTORY diagnostics. For more information, please see the [Guide to GEOS_Chem History diagnostics](#) on the GEOS-Chem wiki.

27.2.6 Changing the numerical integrator

Several global options for **KPP** are listed at the top of the `gckpp.kpp` file:

#MINVERSION	3.0.0	{ Need this version of KPP or later	}
#INTEGRATOR	rosenbrock_autoreduce	{ Use Rosenbrock integration method	}
#AUTOREDUCE	on	{ ... with autoreduce enabled but optional	}
#LANGUAGE	Fortran90	{ Generate solver code in Fortran90 ...	}
#UPPERCASEF90	on	{ ... with .F90 suffix (instead of .f90)	}
#DRIVER	none	{ Do not create gckpp_Main.F90	}
#HESSIAN	off	{ Do not create the Hessian matrix	}
#MEX	off	{ MEX is for Matlab, so skip it	}
#STOICMAT	off	{ Do not create stoichiometric matrix	}

The `#INTEGRATOR` tag specifies the choice of numerical integrator that you wish to use with your chemical mechanism. The table below lists

Table 1: Integrators used for each KPP-based GEOS-Chem mechanism

Simulation	<code>#INTEGRATOR</code>	<code>#AUTOREDUCE</code>
carbon	feuler	
custom	rosenbrock_autoreduce	on
fullchem	rosenbrock_autoreduce	on
Hg	rosenbrock	

Attention: The auto-reduction option is activated but disabled by default in the GEOS-Chem carbon and fullchem mechanisms. You must activate the auto-reduction option in `geoschem_config.yml`.

If you wish to use a different integrator for research purposes, you may select from [several more options](#).

The `#LANGUAGE` should be set to `Fortran90` and `#UPPERCASEF90` should be set to `on`.

The `#MINVERSION` should be set to 3.0.0. This is the minimum KPP version you should be using with GEOS-Chem.

The other options should be left as they are, as they are not relevant to **GEOS-Chem**.

For more information about **KPP** settings, please see <https://kpp.readthedocs.io>.

VIEW RELATED DOCUMENTATION

Table 1: GEOS-Chem web, wiki and Youtube channel

Site	Link
GEOS-Chem web site	geos-chem.org
GEOS-Chem wiki	wiki.geos-chem.org
Video tutorials on Youtube (various)	youtube.com/~geoschem

Table 2: User manuals for GEOS-Chem and related software

Software	Maintained by	Documentation and contact info
GEOS-Chem Classic	GCST	geos-chem.readthedocs.io
GCHP	GCST	gchp.readthedocs.io
HEMCO	GCST	hemco.readthedocs.io
GEOS-Chem on the cloud	GCST	geos-chem-cloud.readthedocs.io
GCPy (Python toolkit)	GCST	gcpy.readthedocs.io
WRF-GC (GEOS-Chem in WRF)	WRF-GC developers	wrf.geos-chem.org
KPP (The Kinetic PreProcessor)	KPP developers	kpp.readthedocs.io
IMI (Integrated Methane Inversion)	IMI developers	imi.readthedocs.io
CHEEREIO (Data assimilation & emissions inversions)	Drew Pendergrass (Harvard)	cheereio.readthedocs.io

Table 3: User manual archives for unsupported legacy software

Legacy Software	Documentation archive
Unsupported GEOS-Chem Classic versions (prior to 13.0.0)	geoschem.github.io/gcclassic-manpage-archive
GAMAP (superseded by GCPy)	geoschem.github.io/gamap-manual

GEOS-CHEM VERSION HISTORY

For a list of updates by GEOS-Chem version, please see:

- [CHANGELOG.md](#) for the GEOS-Chem science codebase
- [CHANGELOG.md](#) for the GCClassic wrapper
- [CHANGELOG.md](#) for HEMCO

KNOWN BUGS AND ISSUES

Please see our [Issue tracker on GitHub](#) for a list of recent bugs and fixes.

30.1 Current bug reports

These [bug reports](#) (listed at [GitHub](#)) are currently unresolved. We hope to fix these in future releases.

CONTRIBUTING GUIDELINES

Thank you for looking into contributing to GEOS-Chem! GEOS-Chem is a grass-roots model that relies on contributions from community members like you. Whether you're new to GEOS-Chem or a longtime user, you're a valued member of the community, and we want you to feel empowered to contribute.

Updates to the GEOS-Chem model benefit both you and the [entire GEOS-Chem community](#). You benefit through [coauthorship and citations](#). Priority development needs are identified at GEOS-Chem users' meetings with updates between meetings based on [GEOS-Chem Steering Committee \(GCSC\)](#) input through [Working Groups](#).

31.1 We use GitHub and ReadTheDocs

We use GitHub to host the GEOS-Chem Classic source code, to track issues, user questions, and feature requests, and to accept pull requests: <https://github.com/geoschem/geos-chem>. Please help out as you can in response to issues and user questions.

GEOS-Chem Classic documentation can be found at geos-chem.readthedocs.io.

31.2 When should I submit updates?

Submit bug fixes right away, as these will be given the highest priority. Please see “Support Guidelines” for more information.

Submit updates (code and/or data) for mature model developments once you have submitted a paper on the topic. Your Working Group chair can offer guidance on the timing of submitting code for inclusion into GEOS-Chem.

The practical aspects of submitting code updates are listed below.

31.3 How can I submit updates?

We use [GitHub Flow](#), so all changes happen through pull requests. This workflow is [described here](#).

As the author you are responsible for:

- Testing your changes
- Updating the user documentation (if applicable)
- Supporting issues and questions related to your changes

31.3.1 Process for submitting code updates

1. Contact your GEOS-Chem Working Group leaders to request that your updates be added to GEOS-Chem. They will forward your request to the GCSC.
2. The GCSC meets quarterly to set [GEOS-Chem model development priorities](#). Your update will be slated for inclusion into an upcoming GEOS-Chem version.
3. Create or log into your [GitHub](#) account.
4. [Fork the relevant GEOS-Chem repositories](#) into your Github account.
5. Clone your forks of the GEOS-Chem repositories to your computer system.
6. Add your modifications into a [new branch](#) off the **main** branch.
7. Test your update thoroughly and make sure that it works. For structural updates we recommend performing a difference test (i.e. testing against the prior version) in order to ensure that identical results are obtained).
8. Review the coding conventions and checklists for code and data updates listed below.
9. Create a [pull request in GitHub](#).
10. The [GEOS-Chem Support Team](#) will add your updates into the development branch for an upcoming GEOS-Chem version. They will also validate your updates with [benchmark simulations](#).
11. If the benchmark simulations reveal a problem with your update, the GCST will request that you take further corrective action.

31.3.2 Coding conventions

The GEOS-Chem codebase dates back several decades and includes contributions from many people and multiple organizations. Therefore, some inconsistent conventions are inevitable, but we ask that you do your best to be consistent with nearby code.

31.3.3 Checklist for submitting code updates

1. Use Fortran-90 free format instead of Fortran-77 fixed format.
2. Include thorough comments in all submitted code.
3. Include full citations for references at the top of relevant source code modules.
4. Remove extraneous code updates (e.g. testing options, other science).
5. Submit any related code or configuration files for [GCHP](#) along with code or configuration files for [GEOS-Chem Classic](#).

31.3.4 Checklist for submitting data files

1. Choose a final file naming convention before submitting data files for inclusion to GEOS-Chem.
2. Make sure that all netCDF files [adhere to the COARDS conventions](#).
3. [Concatenate netCDF files](#) to reduce the number of files that need to be opened. This results in more efficient I/O operations.
4. [Chunk and deflate netCDF files](#) in order to improve file I/O.
5. Include an updated [HEMCO configuration file](#) corresponding to the new data.

6. Include a README file detailing data source, contents, etc.
7. Include script(s) used to process original data
8. Include a summary or description of the expected results (e.g. emission totals for each species)

Also follow these additional steps to ensure that your data can be read by GCHP:

1. All netCDF data variables should be of type `float` (aka `REAL*4`) or `double` (aka `REAL*8`).
2. Use a recent reference datetime (i.e. after 1900-01-01) for the netCDF `time:units` attribute.
3. The first time value in each file should be 0, corresponding with the reference datetime.

31.4 How can I request a new feature?

We accept feature requests through issues on GitHub. To request a new feature, [open a new issue](#) and select the feature request template. Please include all the information that might be relevant, including the motivation for the feature.

31.5 How can I report a bug?

Please see [Support Guidelines](#).

31.6 Where can I ask for help?

Please see [Support Guidelines](#)

SUPPORT GUIDELINES

GEOS-Chem support is maintained by the **GEOS-Chem Support Team (GCST)**, which is based jointly at Harvard University and Washington University in St. Louis.

We track bugs, user questions, and feature requests through [GitHub issues](#). Please help out as you can in response to issues and user questions.

32.1 How to report a bug

We use GitHub to track issues. To report a bug, [open a new issue](#). Please include your name, institution, and all relevant information, such as simulation log files and instructions for replicating the bug.

32.2 Where can I ask for help?

We use GitHub issues to support user questions. To ask a question, [open a new issue](#) and select the question template. Please include your name and institution in the issue.

32.3 What type of support can I expect?

We will be happy to assist you in resolving bugs and technical issues that arise when compiling or running GEOS-Chem. User support and outreach is an important part of our mission to support the [International GEOS-Chem User Community](#).

Even though we can assist in several ways, we cannot possibly do everything. We rely on GEOS-Chem users being resourceful and willing to try to resolve problems on their own to the greatest extent possible.

If you have a science question rather than a technical question, you should contact the relevant [GEOS-Chem Working Group\(s\)](#) directly. But if you do not know whom to ask, you may open a new issue (See “Where can I ask for help” above) and we will be happy to direct your question to the appropriate person(s).

32.4 How to submit changes

Please see [Contributing Guidelines](#).

32.5 How to request an enhancement

Please see [Contributing Guidelines](#).

EDITING THIS USER GUIDE

This user guide is generated with [Sphinx](#). Sphinx is an open-source Python project designed to make writing software documentation easier. The documentation is written in a reStructuredText (it's similar to markdown), which Sphinx extends for software documentation. The source for the documentation is the `docs/source` directory in top-level of the source code.

33.1 Quick start

To build this user guide on your local machine, you need to install Sphinx and its dependencies. Sphinx is a Python 3 package and it is available via **pip**. This user guide uses the Read The Docs theme, so you will also need to install `sphinx-rtd-theme`. It also uses the [sphinxcontrib-bibtex](#) and [recommonmark](#) extensions, which you'll need to install.

```
$ cd docs
$ pip install -r requirements.txt
```

To build this user guide locally, navigate to the `docs/` directory and make the `html` target.

```
$ make html
```

This will build the user guide in `docs/build/html`, and you can open `index.html` in your web-browser. The source files for the user guide are found in `docs/source`.

Note: You can clean the documentation with `make clean`.

33.2 Learning reST

Writing reST can be tricky at first. Whitespace matters, and some directives can be easily miswritten. Two important things you should know right away are:

- Indents are 3-spaces
- “Things” are separated by 1 blank line. For example, a list or code-block following a paragraph should be separated from the paragraph by 1 blank line.

You should keep these in mind when you're first getting started. Dedicating an hour to learning reST will save you time in the long-run. Below are some good resources for learning reST.

- [reStructuredText primer](#): (single best resource; however, it's better read than skimmed)

- Official [reStructuredText reference](#) (there is *a lot* of information here)
- [Presentation by Eric Holscher](#) (co-founder of Read The Docs) at DjangoCon US 2015 (the entire presentation is good, but reST is described from 9:03 to 21:04)
- [YouTube tutorial by Audrey Tavares](#)

A good starting point would be Eric Holscher's presentations followed by the reStructuredText primer.

33.3 Style guidelines

Important: This user guide is written in semantic markup. This is important so that the user guide remains maintainable. Before contributing to this documentation, please review our style guidelines (below). When editing the source, please refrain from using elements with the wrong semantic meaning for aesthetic reasons. Aesthetic issues can be addressed by changes to the theme.

For **titles and headers**:

- Section headers should be underlined by # characters
- Subsection headers should be underlined by – characters
- Subsubsection headers should be underlined by ^ characters
- Subsubsubsection headers should be avoided, but if necessary, they should be underlined by " characters

File paths (including directories) occurring in the text should use the `:file:` role.

Program names (e.g. `cmake`) occurring in the text should use the `:program:` role.

OS-level commands (e.g. `rm`) occurring in the text should use the `:command:` role.

Environment variables occurring in the text should use the `:envvar:` role.

Inline code or code variables occurring in the text should use the `:code:` role.

Code snippets should use `.. code-block:: <language>` directive like so

```
.. code-block:: python

import gcpy
print("hello world")
```

The language can be “none” to omit syntax highlighting.

For command line instructions, the “console” language should be used. The `$` should be used to denote the console's prompt. If the current working directory is relevant to the instructions, a prompt like `$~/path1/path2$` should be used.

Inline literals (e.g. the `$` above) should use the `:literal:` role.

BIBLIOGRAPHY

- [Bey et al., 2001] Bey, I., Jacob, D. J., Yantosca, R. M., Logan, J. A., Field, B. D., Fiore, A. M., Li, Q., Liu, H. Y., Mickley, L. J., and Schultz, M. G. Global modeling of tropospheric chemistry with assimilated meteorology: Model description and evaluation. *J. Geophys. Res.*, 106(D19):23073–23095, Oct 2001. doi:10.1029/2001JD000807.
- [Bindle et al., 2021] Bindle, L., Martin, R. V., Cooper, M. J., Lundgren, E. W., Eastham, S. D., Auer, B. M., Clune, T. L., Weng, H., Lin, J., Murray, L. T., Meng, J., Keller, C. A., Putman, W. M., Pawson, S., and Jacob, D. J. Grid-stretching capability for the GEOS-Chem 13.0.0 atmospheric chemistry model. *Geosci. Model Dev.*, 14(10):5977–5997, 2021. doi:10.5194/gmd-14-5977-2021.
- [Bukosa et al., 2023] Bukosa, B., Fisher, J., Deutscher, N., and Jones, D. A Coupled CH₄, CO and CO₂ Simulation for Improved Chemical Source Modelling. *Atmosphere*, 14:764, 2023. doi:10.3390/atmos14050764.
- [Eastham et al., 2014] Eastham, S.D., Weisenstein, D.K., and Barrett, S.R.H. Development and evaluation of the unified tropospheric-stratospheric chemistry extension (UCX) for the global chemistry-transport model GEOS-Chem. *Atmos. Env.*, 89:52–63, 2014. doi:10.1016/j.atmosenv.2014.02.001.
- [Eastham et al., 2018] Eastham, S. D., Long, M. S., Keller, C. A., Lundgren, E., Yantosca, R. M., Zhuang, J., Li, C., Lee, C. J., Yannetti, M., Auer, B. M., Clune, T. L., Kouatchou, J., Putman, W. M., Thompson, M. A., Trayanov, A. L., Molod, A. M., Martin, R. V., and Jacob, D. J. GEOS-Chem High Performance (GCHP v11-02c): a next-generation implementation of the GEOS-Chem chemical transport model for massively parallel applications. *Geoscientific Model Development*, 11(7):2941–2953, July 2018. doi:10.5194/gmd-11-2941-2018.
- [Henze et al., 2007] Henze, D.K., Hakami, A., and Seinfeld, J.H. Development of the adjoint of GEOS-Chem. *Atmos. Chem. Phys.*, 7:2413–2433, 2007. doi:10.5194/acp-7-2413-2007.
- [Hu et al., 2018] Hu, L., C.A. Keller and, M.S. L., Sherwen, T., Auer, B., Silva, A. D., Nielsen, J.E., Pawson, S., Thompson, M.A., Trayanov, A.L., Travis, K.R., Grange, S.K., Evans, M.J., and Jacob, D.J. Global simulation of tropospheric chemistry at 12.5km resolution: performance and evaluation of the GEOS-Chem chemical module (v10-1) within the NASA GEOS Earth System Model (GEOS-5 ESM). *Geosci. Model Dev.*, 11:4603–4620, 2018. doi:10.5194/gmd-11-4603-2018.
- [Keller et al., 2014] Keller, C. A., M.S. Long, Yantosca, R.M., Silva, A.M. D., Pawson, S., and Jacob, D.J. HEMCO v1.0: a versatile, ESMF-compliant component for calculating emissions in atmospheric models. *Geosci. Model Dev.*, 7(4):1409–1417, July 2014. doi:10.5194/gmd-7-1409-2014.
- [Lin et al., 2020] Lin, H., Feng, X., Fu, T.-M., Tian, H., Ma, Y., Zhang, L., Jacob, D. J., Yantosca, R. M., Sulprizio, M. P., Lundgren, E. W., Zhuang, J., Zhang, Q., Lu, X., Zhang, L., Shen, L., Guo, J., Eastham, S. D., and Keller, C. A. WRF-GC (v1.0): online coupling of WRF (v3.9.1.1) and GEOS-Chem (v12.2.1) for regional atmospheric chemistry modeling – Part 1: Description of the one-way model. *Geosci. Model Dev.*, 13:3241–3265, 2020. doi:10.5194/gmd-13-3241-2020.

- [Lin et al., 2023] Lin, H., Long, M. S., Sander, R., Sandu, A., Yantosca, R. M., Estrada, L. A., Shen, L., and Jacob, D. J. An adaptive auto-reduction solver for speeding up integration of chemical kinetics in atmospheric chemistry models: implementation and evaluation within the Kinetic Pre-Processor (KPP) version 3.0.0. *J. Adv. Model. Earth Syst.*, pages 2022MS003293, 2023. doi:10.1029/2022MS003293.
- [Lin et al., 2021] Lin, H., Jacob, D. J., Lundgren, E. W., Sulprizio, M. P., Keller, C. A., Fritz, T. M., Eastham, S. D., Emmons, L. K., Campbell, P. C., Baker, B., Saylor, R. D., and Montuoro, R. Harmonized Emissions Component (HEMCO) 3.0 as a versatile emissions component for atmospheric models: application in the GEOS-Chem, NASA GEOS, WRF-GC, CESM2, NOAA GEFS-Aerosol, and NOAA UFS models. *Geosci. Model. Dev.*, 14:5487–5506, 2021. doi:0.5194/gmd-14-5487-2021.
- [Long et al., 2015] Long, M.S., and J.E. Nielsen, R. Y., Keller, C.A., da Silva, A., Sulprizio, M.P., Pawson, S., and Jacob, D.J. Development of a grid-independent GEOS-Chem chemical transport model (v9-02) as an atmospheric chemistry module for Earth system models. *Geosci. Model Dev.*, 8(3):595–602, March 2015. doi:10.5194/gmd-8-595-2015.
- [Luo et al., 2020] Luo, G., Yu, F., and Moch, J. Further improvement of wet process treatments in GEOS-Chem v12.6.0: impact on global distributions of aerosols and aerosol precursors. *Geosci. Model. Dev.*, 13:2879–2903, 2020. doi:10.5194/gmd-13-2879-2020.
- [Murray et al., 2021] Murray, L.T., Leibensperger, E.M., Orbe, C., Mickley, L.J., and Sulprizio, M. GCAP 2.0: a global 3-D chemical-transport model framework for past, present, and future climate scenarios. *Geosci. Model Dev.*, 14:5789–5823, 2021. doi:10.5194/gmd-14-5789-2021.
- [Park et al., 2004] Park, R.J., Jacob, D.J., Field, B.D., R.M. Yantosca, and Chin, M. Natural and transboundary pollution influences on sulfate-nitrate-ammonium aerosols in the United States: implications for policy. *J. Geophys. Res.*, 109(D15):204ff, 2004. doi:10.1029/2003JD004473.
- [Philip et al., 2016] Philip, S., Martin, R. V., and Keller, C. A. Sensitivity of chemistry-transport model simulations to the duration of chemical and transport operators: a case study with GEOS-Chem v10-01. *Geosci. Model Dev.*, 9:1683–1695, 2016. doi:10.5194/gmd-9-1683-2016.
- [Selin et al., 2007] Selin, N.E., D.J. Jacob, Park, R.J., Yantosca, R.M., Strode, S., L. Jaeglé, and Jaffe, D. Chemical cycling and deposition of atmospheric mercury: Global constraints from observations. *J. Geophys. Res.*, 112(D02308):, 2007. doi:10.1029/2006JD007450.
- [Trivitayanurak et al., 2008] Trivitayanurak, W., Adams, P., Spracklen, D., and Carslaw, K. Tropospheric aerosol microphysics simulation with assimilated meteorology: model description and intermodel comparison. *Atmos. Chem. Phys.*, 8:3149–3168, 2008.
- [Wang et al., 2004] Y.X. Wang, McElroy, M.B., Jacob, D.J., and Yantosca, R.M. A Nested Grid Formulation for Chemical Transport over Asia: Applications to CO. *J. Geophys. Res.*, 109(D22):307ff, 2004. doi:10.1029/2004jd005237.
- [Yu and Luo 2009] Yu, F. and Luo, G. Simulation of particle size distribution with a global aerosol model: Contribution of nucleation to aerosol and CCN number concentrations. *Atmos. Chem. Phys.*, 9(7):7691–7710, 2009.
- [Zhuang et al., 2019] Zhuang, J., D.J. Jacob, J. Flo-Gaya, Yantosca, R.M., Lundgren, E.W., Sulprizio, M.P., and Eastham, S.D. Enabling immediate access to Earth science models through cloud computing: application to the GEOS-Chem model. *Bull. Amer. Met. Soc.*, pages 1943–1960, October 2019. doi:10.1175/BAMS-D-18-0243.1.
- [Zhuang et al., 2020] Zhuang, J., Jacob, D. J., Lin, H., Lundgren, E. W., Yantosca, R. M., Gaya, J. F., Sulprizio, M. P., and Eastham, S. D. Enabling High-Performance Cloud Computing for Earth Science Modeling on Over a Thousand Cores: Application to the GEOS-Chem Atmospheric Chemistry Model. *Journal of Advances in Modeling Earth Systems*, May 2020. doi:10.1029/2020MS002064.

Symbols

!\$OMP COLLAPSE(2)
 command line option, 210
 !\$OMP END PARALLEL DO
 command line option, 211
 !\$OMP PARALLEL DO
 command line option, 210
 !\$OMP PRIVATE(I
 command line option, 210
 !\$OMP SCHEDULE(DYNAMIC
 command line option, 211
 !\$OMP SHARED(A)
 command line option, 210
 .dirty
 command line option, 49
 ::
 command line option, 84
 #SBATCH -N 1
 command line option, 102
 #SBATCH --mail-type=END
 command line option, 102
 #SBATCH --mem=15000
 command line option, 102
 #SBATCH -c 8
 command line option, 102
 #SBATCH -p MYQUEUE
 command line option, 102
 #SBATCH -t 0-12:00
 command line option, 102
 \$PATH, 132
 \${HOME}, 132
 _FillValue
 command line option, 190
 -DRUNDIR=/path/to/run/dir, 52
 -DRUNDIR=/path/to/run/directory, 51
 `GEOS-Chem/Interfaces/GCCClassic
 command line option, 121
 <collection-name>.LAT_RANGE
 command line option, 84
 <collection-name>.LON_RANGE
 command line option, 84
 <collection-name>.duration

 command line option, 83
 <collection-name>.fields
 command line option, 83
 <collection-name>.frequency
 command line option, 83
 <collection-name>.levels
 command line option, 84
 <collection-name>.mode
 command line option, 83
 <collection-name>.template
 command line option, 82
 0.5x0.625
 command line option, 58
 2.0x2.5
 command line option, 58
 3D_chemical_oxidation_source
 command line option, 75
 4)
 command line option, 211
 4.0x5.0
 command line option, 58

Numbers

15
 command line option, 46
 40
 command line option, 46, 58
 47
 command line option, 58
 72
 command line option, 58

A

abcd1234
 command line option, 49
 absolute_threshold
 command line option, 61
 acid_uptake_on_dust
 command line option, 69
 activate
 command line option, 57, 59–72
 active_strat_H2O

- command line option, 60
- aerosol
 - command line option, 56
- AIRS
 - command line option, 74
- all
 - command line option, 57, 200–202
- all_sky_flux
 - command line option, 66
- allow_homogeneous_NAT
 - command line option, 70
- Amazon
 - command line option, 89
- aod_wavelengths_in_nm
 - command line option, 65
- APM
 - command line option, 47
- append_in_internal_timestep
 - command line option, 62
- archiveRun.sh
 - command line option, 39
- autoreduce_solver
 - command line option, 61
- aviation_emissions
 - command line option, 75

B

- B)
 - command line option, 210
- BackgroundVV
 - command line option, 204
- balanced_biosphere_exchange
 - command line option, 75
- boundary_layer
 - command line option, 201
- BPCH_DIAG
 - command line option, 46
- brown_carbon
 - command line option, 67
- buffer_zone_NSEW
 - command line option, 59
- build/
 - command line option, 39
- build_info/
 - command line option, 39

C

- C
 - command line option, 49
- calc_strat_aod
 - command line option, 71
- carbon
 - command line option, 46, 56
- CC, 20, 21

- cdo
 - command line option, 169
- center_at_180
 - command line option, 58
- CH4
 - command line option, 56
- CH4_boundary_condition_ppb_increase_NSEW
 - command line option, 75
- chemistry_timestep_in_s
 - command line option, 59
- cleanRunDir.sh
 - command line option, 39
- clear_sky_flux
 - command line option, 66
- cloudj_input_dir
 - command line option, 64
- CMAKE_BUILD_TYPE
 - command line option, 45
- CO2
 - command line option, 56
- CO2_effect
 - command line option, 62
- CO2_level
 - command line option, 63
- CodeDir
 - command line option, 39
- COLLECTIONS:
 - command line option, 82
- command line option
 - !\$OMP COLLAPSE(2), 210
 - !\$OMP END PARALLEL DO, 211
 - !\$OMP PARALLEL DO, 210
 - !\$OMP PRIVATE(I, 210
 - !\$OMP SCHEDULE(DYNAMIC, 211
 - !\$OMP SHARED(A), 210
 - .dirty, 49
 - ::, 84
 - #SBATCH -N 1, 102
 - #SBATCH --mail-type=END, 102
 - #SBATCH --mem=15000, 102
 - #SBATCH -c 8, 102
 - #SBATCH -p MYQUEUE, 102
 - #SBATCH -t 0-12:00, 102
 - _FillValue, 190
 - `GEOS-Chem/Interfaces/GCClassic, 121
 - <collection-name>.LAT_RANGE, 84
 - <collection-name>.LON_RANGE, 84
 - <collection-name>.duration, 83
 - <collection-name>.fields, 83
 - <collection-name>.frequency, 83
 - <collection-name>.levels, 84
 - <collection-name>.mode, 83
 - <collection-name>.template, 82
 - 0.5x0.625, 58

- 2.0x2.5, 58
- 3D_chemical_oxidation_source, 75
- 4), 211
- 4.0x5.0, 58
- 15, 46
- 40, 46, 58
- 47, 58
- 72, 58
- abcd1234, 49
- absolute_threshold, 61
- acid_uptake_on_dust, 69
- activate, 57, 59–72
- active_strat_H2O, 60
- aerosol, 56
- AIRS, 74
- all, 57, 200–202
- all_sky_flux, 66
- allow_homogeneous_NAT, 70
- Amazon, 89
- aod_wavelengths_in_nm, 65
- APM, 47
- append_in_internal_timestep, 62
- archiveRun.sh, 39
- autoreduce_solver, 61
- aviation_emissions, 75
- B), 210
- BackgroundVV, 204
- balanced_biosphere_exchange, 75
- boundary_layer, 201
- BPCH_DIAG, 46
- brown_carbon, 67
- buffer_zone_NSEW, 59
- build/, 39
- build_info/, 39
- C, 49
- calc_strat_aod, 71
- carbon, 46, 56
- cdo, 169
- center_at_180, 58
- CH4, 56
- CH4_boundary_condition_ppb_increase_NSEW, 75
- chemistry_timestep_in_s, 59
- cleanRunDir.sh, 39
- clear_sky_flux, 66
- cloudj_input_dir, 64
- CMAKE_BUILD_TYPE, 45
- CO2, 56
- CO2_effect, 62
- CO2_level, 63
- CodeDir, 39
- COLLECTIONS:, 82
- constant, 200, 201
- Contact (*or contact*), 193
- Conventions (*or conventions*), 193
- CreateRunDirLogs/rundir_vars.txt, 39
- custom, 46
- DD_AeroDryDep, 197
- DD_DustDryDep, 197
- DD_DvzAerSnow, 197
- DD_DvzAerSnow_Luo, 197
- DD_DvzMinVal, 197
- DD_F0, 198
- DD_Hstar_Old, 198
- DD_KOA, 198
- Debug, 46
- decay_of_another_species, 201
- Density, 197
- diag_alt_above_sfc_in_m:, 63
- download_data.py, 39
- download_data.yml, 39
- E, 93
- efolding, 200
- end_date, 57
- enhance_black_carbon_absorption, 68
- export OMP_NUM_THREADS=\$SLURM_CPUS_PER_TASK, 102
- export OMP_STACKSIZE=500m, 102
- F, 93
- FASTJX, 47
- fastjx_input_dir, 64
- Filename (*or filename*), 193
- fill_negative_values, 66
- flight_track_file, 72
- Format (*or format*), 193
- Formula, 196
- fossil_fuel_emissions, 75
- fullchem, 46, 56
- FullName, 196
- g, 49
- gamma_HO2, 61
- GCAP2, 57
- GCClassic/src/GEOS-Chem, 121
- GCClassic/src/GEOS-Chem/GeosCore, 121
- GCClassic/src/GEOS-Chem/GeosRad, 121
- GCClassic/src/GEOS-Chem/GeosUtil, 121
- GCClassic/src/GEOS-Chem/GTMM, 121
- GCClassic/src/GEOS-Chem/Headers, 121
- GCClassic/src/GEOS-Chem/History, 121
- GCClassic/src/GEOS-Chem/ISORROPIA, 121
- GCClassic/src/GEOS-Chem/KPP, 121
- GCClassic/src/GEOS-Chem/NcdfUtil, 121
- GCClassic/src/GEOS-Chem/ObsPack, 121
- GCClassic/src/GEOS-Chem/PKUCPL, 121

GCClassic/src/HEMCO/src/Core, 122
 GCClassic/src/HEMCO/src/Extensions, 122
 GCClassic/src/HEMCO/src/Interfaces, 122
 GCClassic/src/HEMCO/src/shared, 122
 gcclassic_tpcore, 66
 GCPy, 169
 GEOS-FP, 57
 GEOSChem.Restart.YYYYMMDD_hhmmz.nc4, 91
 geoschem_config.yml, 39, 96
 getRunInfo, 39
 GOSAT, 74
 gregorian, 187
 halflife, 200
 HEMCO, 201
 HEMCO_Config.rc, 40, 97
 HEMCO_Config.rc.gcap2_metfields, 40
 HEMCO_Config.rc.gmao_metfields, 40
 HEMCO_Diagn.rc, 40
 HEMCO_restart.YYYYMMDDhhmm.nc, 91
 Henry_CR, 197
 Henry_CR_Luo, 197
 Henry_K0, 197
 Henry_K0_Luo, 197
 Henry_pKa, 197
 het_chem, 70
 Hg, 46, 56
 History (*or history*), 193
 HISTORY.rc, 40, 96
 hydrophilic, 68
 hydrophobic, 68
 input_directories, 64
 input_file, 72
 instantaneous, 83
 iord_jord_kord, 66
 Is_Advected, 196
 Is_Aerosol, 196
 Is_DryAlt, 196
 Is_DryDep, 196
 Is_Gas, 196
 Is_Hg0, 196
 Is_Hg2, 196
 Is_HgP, 196
 Is_HygroGrowth, 196
 Is_Photolysis, 196
 Is_RadioNuclide, 196
 Is_Tracer, 200
 J, 210
 keep_halogens_active, 61
 KPP/carbon, 87
 KPP/custom, 87
 KPP/fullchem, 87
 KPP/Hg, 87
 lat, 185
 lat:axis, 189
 lat:long_name, 189
 lat:units, 189
 lat_zone, 200, 201
 latitude, 58
 lev, 185
 lev:axis, 188
 lev:long_name, 187
 lev:positive, 188
 lev:units, 187
 linear_chemistry_aloft, 60
 lon, 185
 lon:axis, 189
 lon:long_name, 189
 lon:units, 189
 long_name, 190
 longitude, 58
 longwave_fluxes, 65
 LUO_WETDEP, 47
 maintain_mixing_ratio, 201
 marine_organic_aerosols, 70
 MECH, 46
 MERRA2, 57
 met_field, 57
 metal_cat_SO2_oxidation, 71
 metals, 56
 metrics.py, 40
 missing_value, 190
 module load cmake/..., 128
 module load flex/..., 128
 module load gcc/..., 128
 module load git/..., 128
 module load netcdf/..., 128
 module load openmpi/..., 128
 module load perl/..., 128
 module purge, 128
 MP_SizeResAer, 204
 MP_SizeResNum, 204
 MW_g, 197
 n, 46–48
 Name, 196
 name, 56
 NAT_supercooling_req_in_K, 70
 ncdump, 169
 nco, 169
 ncview, 169
 nested_grid_simulation, 59
 net_terrestrial_exchange, 75
 netcdf-scripts, 169
 NIT_Jscale_JHNO2, 65
 NITs_Jscale_JHNO3, 65
 no2 tuning factor, 61

none, 200, 201
 number_of_levels, 58
 O, 93
 ocean_exchange, 75
 oh_tuning_factor, 61
 OMP, 46
 OMP_NUM_THREADS, 21
 OMP_STACKSIZE, 21
 on_cores:, 57
 output_file, 72
 output_species, 72
 OutputDir/, 40
 overhead_O3, 64
 Panoply, 169
 percent_channel_A_HONO, 65
 percent_channel_B_NO2, 65
 perturb_CH4_boundary_conditions, 74
 photolyze_nitrate_aerosol:, 64
 polar_strat_clouds, 70
 POPs, 56
 ppbv, 201
 pressures, 202
 quiet_logfile_output, 72
 radiation_timestep_in_s, 59
 Radius, 197
 range, 58, 59
 README.md, 40
 reference_CO2_level, 63
 References (*or references*), 193
 Release, 45
 resolution, 58
 Restarts/, 40
 Restarts/GEOSChem.Restart.YYYYMMDD_hhmmztime4, 40
 Rochester, 89
 root, 57
 root_data_dir, 57
 RRTMG, 47
 RUNDIR, 45
 runScriptSamples, 40
 SALA_radius_bin_in_um, 69
 SALC_radius_bin_in_um, 69
 SANITIZE, 47
 save_fossil_fuel_in_background, 76
 scale_by_pressure, 61
 semivolatile_POA, 68
 settle_strat_aerosol, 70
 ship_emissions, 75
 shortwave_fluxes, 65
 Snk_Horiz, 200
 Snk_Lats, 200
 Snk_Mode, 200
 Snk_Period, 200
 Snk_Value, 200
 Snk_Vert, 201
 species_database.yml, 41
 species_database_file, 57
 species_metadata_output_file, 57
 Src_Add, 201
 Src_Horiz, 201
 Src_Lats, 201
 Src_Mode, 201
 Src_Pressures, 202
 Src_Unit, 201
 Src_Value, 201
 Src_Vert, 201
 srun -c \$OMP_NUM_THREADS, 102
 standard, 187
 start_date, 56
 stratosphere, 202
 supersat_factor_req_for_ice_nucl, 70
 surface, 201, 202
 TAG, 49
 tag_bio_and_ocean_CO2, 76
 tagCH4, 56
 tagCO, 56
 tagO3, 56
 TCCON, 74
 tie_HgIIaq_reduction_to_UVB, 73
 time, 185
 time -p ./gcclassic > GC.log 2>&1, 102
 time:axis, 187
 time:calendar, 186
 time:long_name, 186
 time:units, 186
 time4averaged, 83
 timestep, 201
 Title (*or title*), 193
 TOMAS, 46
 TOMAS_BINS, 46
 transport_timestep_in_s, 59
 transported_species, 67
 TransportTracers, 56
 troposphere, 201, 202
 ulimit -s unlimited, 102
 Units, 202
 units, 190
 use_arctic_river_Hg, 73
 use_column_O3_from_met, 64
 use_dynamic_ocean_Hg, 73
 use_fullchem_PCO_from_CH4, 76
 use_fullchem_PCO_from_NMVOC, 76
 use_gcclassic_timers, 57
 use_halfpolar_boxes, 59
 use_linoz_for_O3, 60
 use_non_local_pbl, 63
 use_online_O3_from_model, 64

- use_preindustrial_Hg, 73
- use_pressure_threshold, 61
- use_static_bnd_cond, 61
- use_target_threshold, 61
- use_TOMS_SBUV_O3, 64
- verbose:, 57
- WashU, 89
- WD_AerScavEff, 199
- WD_CoarseAer, 198
- WD_ConvFacI2G, 198
- WD_ConvFacI2G_Luo, 198
- WD_Is_H2SO4, 198
- WD_Is_HNO3, 198
- WD_Is_SO2, 198
- WD_KcScaleFac, 199
- WD_KcScaleFac_Luo, 199
- WD_LiqAndGas, 198
- WD_RainoutEff, 199
- WD_RainoutEff_Luo, 200
- WD_RetFactor, 199
- xarray, 170
- Y, 93
- y, 46–48
- constant
 - command line option, 200, 201
- Contact (*or contact*)
 - command line option, 193
- Conventions (*or conventions*)
 - command line option, 193
- CreateRunDirLogs/rundir_vars.txt
 - command line option, 39
- custom
 - command line option, 46
- CXX, 20, 21

D

- Daily, 97
- DD_AeroDryDep
 - command line option, 197
- DD_DustDryDep
 - command line option, 197
- DD_DvzAerSnow
 - command line option, 197
- DD_DvzAerSnow_Luo
 - command line option, 197
- DD_DvzMinVal
 - command line option, 197
- DD_F0
 - command line option, 198
- DD_Hstar_Old
 - command line option, 198
- DD_KOA
 - command line option, 198
- Debug, 106

- command line option, 46
- decay_of_another_species
 - command line option, 201
- Density
 - command line option, 197
- diag_alt_above_sfc_in_m:
 - command line option, 63
- DiagnFreq, 97
- download_data.py
 - command line option, 39
- download_data.yml
 - command line option, 39

E

- E
 - command line option, 93
- efolding
 - command line option, 200
- end_date
 - command line option, 57
- enhance_black_carbon_absorption
 - command line option, 68
- environment variable
 - \$PATH, 132
 - \${HOME}, 132
 - DRUNDIR=/path/to/run/dir, 52
 - DRUNDIR=/path/to/run/directory, 51
 - CC, 20, 21
 - CXX, 20, 21
 - Daily, 97
 - Debug, 106
 - DiagnFreq, 97
 - FC, 20, 21
 - g++, 20
 - gcc, 20
 - gfortran, 20
 - icc, 21
 - icpc, 21
 - icx, 20
 - ifort, 20
 - model, 133
 - Monthly, 97
 - OMP_NUM_THREADS, 21, 124
 - OMP_STACKSIZE, 22
 - Release, 106
 - scope_args, 133
 - scope_dir, 133
 - SLURM_CPUS_PER_TASK, 102
 - SPACK_ROOT, 132
 - YYYYMMDD hhmmss, 97
- export OMP_NUM_THREADS=\$SLURM_CPUS_PER_TASK
 - command line option, 102
- export OMP_STACKSIZE=500m
 - command line option, 102

F

F

command line option, 93
 FASTJX
 command line option, 47
 fastjx_input_dir
 command line option, 64
 FC, 20, 21
 Filename (*or filename*)
 command line option, 193
 fill_negative_values
 command line option, 66
 flight_track_file
 command line option, 72
 Format (*or format*)
 command line option, 193
 Formula
 command line option, 196
 fossil_fuel_emissions
 command line option, 75
 fullchem
 command line option, 46, 56
 FullName
 command line option, 196

G

g

command line option, 49
 g++, 20
 gamma_HO2
 command line option, 61
 GCAP2
 command line option, 57
 gcc, 20
 GCCClassic/src/GEOS-Chem
 command line option, 121
 GCCClassic/src/GEOS-Chem/GeosCore
 command line option, 121
 GCCClassic/src/GEOS-Chem/GeosRad
 command line option, 121
 GCCClassic/src/GEOS-Chem/GeosUtil
 command line option, 121
 GCCClassic/src/GEOS-Chem/GTMM
 command line option, 121
 GCCClassic/src/GEOS-Chem/Headers
 command line option, 121
 GCCClassic/src/GEOS-Chem/History
 command line option, 121
 GCCClassic/src/GEOS-Chem/ISORROPIA
 command line option, 121
 GCCClassic/src/GEOS-Chem/KPP
 command line option, 121
 GCCClassic/src/GEOS-Chem/NcdfUtil
 command line option, 121

GCCClassic/src/GEOS-Chem/ObsPack
 command line option, 121
 GCCClassic/src/GEOS-Chem/PKUCPL
 command line option, 121
 GCCClassic/src/HEMCO/src/Core
 command line option, 122
 GCCClassic/src/HEMCO/src/Extensions
 command line option, 122
 GCCClassic/src/HEMCO/src/Interfaces
 command line option, 122
 GCCClassic/src/HEMCO/src/shared
 command line option, 122
 gccclassic_tpcore
 command line option, 66
 GCPy
 command line option, 169
 GEOS-FP
 command line option, 57
 GEOSChem.Restart.YYYYMMDD_hhmmz.nc4
 command line option, 91
 geoschem_config.yml
 command line option, 39, 96
 getRunInfo
 command line option, 39
 gfortran, 20
 GOSAT
 command line option, 74
 gregorian
 command line option, 187

H

halflife
 command line option, 200
 HEMCO
 command line option, 201
 HEMCO_Config.rc
 command line option, 40, 97
 HEMCO_Config.rc.gcap2_metfields
 command line option, 40
 HEMCO_Config.rc.gmao_metfields
 command line option, 40
 HEMCO_Diagn.rc
 command line option, 40
 HEMCO_restart.YYYYMMDDhhmm.nc
 command line option, 91
 Henry_CR
 command line option, 197
 Henry_CR_Luo
 command line option, 197
 Henry_K0
 command line option, 197
 Henry_K0_Luo
 command line option, 197
 Henry_pKa

- command line option, 197
- het_chem
 - command line option, 70
- Hg
 - command line option, 46, 56
- History (*or history*)
 - command line option, 193
- HISTORY.rc
 - command line option, 40, 96
- hydrophilic
 - command line option, 68
- hydrophobic
 - command line option, 68

I

- icc, 21
- icpc, 21
- icx, 20
- ifort, 20
- input_directories
 - command line option, 64
- input_file
 - command line option, 72
- instantaneous
 - command line option, 83
- iord_jord_kord
 - command line option, 66
- Is_Advected
 - command line option, 196
- Is_Aerosol
 - command line option, 196
- Is_DryAlt
 - command line option, 196
- Is_DryDep
 - command line option, 196
- Is_Gas
 - command line option, 196
- Is_Hg0
 - command line option, 196
- Is_Hg2
 - command line option, 196
- Is_HgP
 - command line option, 196
- Is_HygroGrowth
 - command line option, 196
- Is_PhotoLysis
 - command line option, 196
- Is_RadioNuclide
 - command line option, 196
- Is_Tracer
 - command line option, 200

J

J

- command line option, 210

K

- keep_halogens_active
 - command line option, 61
- KPP/carbon
 - command line option, 87
- KPP/custom
 - command line option, 87
- KPP/fullchem
 - command line option, 87
- KPP/Hg
 - command line option, 87

L

- lat
 - command line option, 185
- lat:axis
 - command line option, 189
- lat:long_name
 - command line option, 189
- lat:units
 - command line option, 189
- lat_zone
 - command line option, 200, 201
- latitude
 - command line option, 58
- lev
 - command line option, 185
- lev:axis
 - command line option, 188
- lev:long_name
 - command line option, 187
- lev:positive
 - command line option, 188
- lev:units
 - command line option, 187
- linear_chemistry_aloft
 - command line option, 60
- lon
 - command line option, 185
- lon:axis
 - command line option, 189
- lon:long_name
 - command line option, 189
- lon:units
 - command line option, 189
- long_name
 - command line option, 190
- longitude
 - command line option, 58
- longwave_fluxes
 - command line option, 65
- LUO_WETDEP

command line option, 47

M

maintain_mixing_ratio

command line option, 201

marine_organic_aerosols

command line option, 70

MECH

command line option, 46

MERRA2

command line option, 57

met_field

command line option, 57

metal_cat_SO2_oxidation

command line option, 71

metals

command line option, 56

metrics.py

command line option, 40

missing_value

command line option, 190

model, 133

module load cmake/...

command line option, 128

module load flex/...

command line option, 128

module load gcc/...

command line option, 128

module load git/...

command line option, 128

module load netcdf/..

command line option, 128

module load openmpi/...

command line option, 128

module load perl/...

command line option, 128

module purge

command line option, 128

Monthly, 97

MP_SizeResAer

command line option, 204

MP_SizeResNum

command line option, 204

MW_g

command line option, 197

N

n

command line option, 46–48

Name

command line option, 196

name

command line option, 56

NAT_supercooling_req_in_K

command line option, 70

ncdump

command line option, 169

nco

command line option, 169

ncview

command line option, 169

nested_grid_simulation

command line option, 59

net_terrestrial_exchange

command line option, 75

netcdf-scripts

command line option, 169

NIT_Jscale_JHNO2

command line option, 65

NITs_Jscale_JHNO3

command line option, 65

no2 tuning factor

command line option, 61

none

command line option, 200, 201

number_of_levels

command line option, 58

O

O

command line option, 93

ocean_exchange

command line option, 75

oh_tuning_factor

command line option, 61

OMP

command line option, 46

OMP_NUM_THREADS, 21, 124

command line option, 21

OMP_STACKSIZE, 22

command line option, 21

on_cores:

command line option, 57

output_file

command line option, 72

output_species

command line option, 72

OutputDir/

command line option, 40

overhead_O3

command line option, 64

P

Panoply

command line option, 169

percent_channel_A_HONO

command line option, 65

percent_channel_B_NO2

- command line option, [65](#)
- perturb_CH4_boundary_conditions
 - command line option, [74](#)
- photolyze_nitrate_aerosol:
 - command line option, [64](#)
- polar_strat_clouds
 - command line option, [70](#)
- POPs
 - command line option, [56](#)
- ppbv
 - command line option, [201](#)
- pressures
 - command line option, [202](#)

Q

- quiet_logfile_output
 - command line option, [72](#)

R

- radiation_timestep_in_s
 - command line option, [59](#)
- Radius
 - command line option, [197](#)
- range
 - command line option, [58](#), [59](#)
- README.md
 - command line option, [40](#)
- reference_CO2_level
 - command line option, [63](#)
- References (*or references*)
 - command line option, [193](#)
- Release, [106](#)
 - command line option, [45](#)
- resolution
 - command line option, [58](#)
- Restarts/
 - command line option, [40](#)
- Restarts/GEOSChem.Restart.YYYYMMDD_hhmmz21.nc4
 - command line option, [40](#)
- Rochester
 - command line option, [89](#)
- root
 - command line option, [57](#)
- root_data_dir
 - command line option, [57](#)
- RRTMG
 - command line option, [47](#)
- RUNDIR
 - command line option, [45](#)
- runScriptSamples
 - command line option, [40](#)

S

- SALA_radius_bin_in_um

- command line option, [69](#)
- SALC_radius_bin_in_um
 - command line option, [69](#)
- SANITIZE
 - command line option, [47](#)
- save_fossil_fuel_in_background
 - command line option, [76](#)
- scale_by_pressure
 - command line option, [61](#)
- scope_args, [133](#)
- scope_dir, [133](#)
- semivolatile_POA
 - command line option, [68](#)
- settle_strat_aerosol
 - command line option, [70](#)
- ship_emissions
 - command line option, [75](#)
- shortwave_fluxes
 - command line option, [65](#)
- SLURM_CPUS_PER_TASK, [102](#)
- Snk_Horiz
 - command line option, [200](#)
- Snk_Lats
 - command line option, [200](#)
- Snk_Mode
 - command line option, [200](#)
- Snk_Period
 - command line option, [200](#)
- Snk_Value
 - command line option, [200](#)
- Snk_Vert
 - command line option, [201](#)
- SPACK_ROOT, [132](#)
- species_database.yml
 - command line option, [41](#)
- species_database_file
 - command line option, [57](#)
- species_metadata_output_file
 - command line option, [57](#)
- Src_Add
 - command line option, [201](#)
- Src_Horiz
 - command line option, [201](#)
- Src_Lats
 - command line option, [201](#)
- Src_Mode
 - command line option, [201](#)
- Src_Pressures
 - command line option, [202](#)
- Src_Unit
 - command line option, [201](#)
- Src_Value
 - command line option, [201](#)
- Src_Vert

command line option, 201
 srun -c \$OMP_NUM_THREADS
 command line option, 102
 standard
 command line option, 187
 start_date
 command line option, 56
 stratosphere
 command line option, 202
 supersat_factor_req_for_ice_nucl
 command line option, 70
 surface
 command line option, 201, 202

T

TAG
 command line option, 49
 tag_bio_and_ocean_CO2
 command line option, 76
 tagCH4
 command line option, 56
 tagCO
 command line option, 56
 tagO3
 command line option, 56
 TCCON
 command line option, 74
 tie_HgIIaq_reduction_to_UVB
 command line option, 73
 time
 command line option, 185
 time -p ./gcclassic > GC.log 2>&1
 command line option, 102
 time:axis
 command line option, 187
 time:calendar
 command line option, 186
 time:long_name
 command line option, 186
 time:units
 command line option, 186
 time-averaged
 command line option, 83
 timestep
 command line option, 201
 Title (*or title*)
 command line option, 193
 TOMAS
 command line option, 46
 TOMAS_BINS
 command line option, 46
 transport_timestep_in_s
 command line option, 59
 transported_species

command line option, 67
 TransportTracers
 command line option, 56
 troposphere
 command line option, 201, 202

U

ulimit -s unlimited
 command line option, 102
 Units
 command line option, 202
 units
 command line option, 190
 use_arctic_river_Hg
 command line option, 73
 use_column_O3_from_met
 command line option, 64
 use_dynamic_ocean_Hg
 command line option, 73
 use_fullchem_PCO_from_CH4
 command line option, 76
 use_fullchem_PCO_from_NMVOC
 command line option, 76
 use_gcclassic_timers
 command line option, 57
 use_halfpolar_boxes
 command line option, 59
 use_linoz_for_O3
 command line option, 60
 use_non_local_pbl
 command line option, 63
 use_online_O3_from_model
 command line option, 64
 use_preindustrial_Hg
 command line option, 73
 use_pressure_threshold
 command line option, 61
 use_static_bnd_cond
 command line option, 61
 use_target_threshold
 command line option, 61
 use_TOMS_SBUV_O3
 command line option, 64

V

verbose:
 command line option, 57

W

WashU
 command line option, 89
 WD_AerScavEff
 command line option, 199
 WD_CoarseAer

- command line option, 198
- WD_ConvFacI2G
 - command line option, 198
- WD_ConvFacI2G_Luo
 - command line option, 198
- WD_Is_H2SO4
 - command line option, 198
- WD_Is_HNO3
 - command line option, 198
- WD_Is_SO2
 - command line option, 198
- WD_KcScaleFac
 - command line option, 199
- WD_KcScaleFac_Luo
 - command line option, 199
- WD_LiqAndGas
 - command line option, 198
- WD_RainoutEff
 - command line option, 199
- WD_RainoutEff_Luo
 - command line option, 200
- WD_RetFactor
 - command line option, 199

X

- xarray
 - command line option, 170

Y

- Y
 - command line option, 93
- Y
 - command line option, 46–48
- YYYYMMDD hhmmss, 97