GCHP

Release 14.3.1

GEOS-Chem Support Team

Apr 02, 2024

GETTING STARTED

The GEOS–Chem model 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

This site provides instructions for GEOS-Chem High Performance, GEOS-Chem's multi-node variant. We provide two different instruction sets for downloading and compiling GCHP: from a clone of the source code, or using the Spack package manager.

Cloning and building from source code ensures you will have direct access to the latest available versions of GCHP, provides additional compile-time options, and allows you to make your own modifications to GCHP's source code. Spack automates downloading and additional parts of the compiling process while providing you with some standard togglable compile-time options.

Our Quick Start Guide and the downloading, compiling, and creating a run directory sections of the User Guide give instructions specifically for using a clone of the source code. Our dedicated Spack guide describes how to install GCHP and create a run directory with Spack, as well as how to use Spack to install GCHP's dependencies if needed.

CHAPTER

ONE

QUICKSTART GUIDE

This quickstart guide assumes your environment satisfies the requirements described in *System Requirements*. This means you should load a compute environment so that programs like **cmake** and **mpirun** are available before continuing. If you do not have some of GCHP's software dependencies, you can find instructions for installing GCHP's external dependencies in our Spack instructions. More detailed instructions on downloading, compiling, and running GCHP can be found in the User Guide.

1.1 1. Clone GCHP

Download the source code. The `--recurse-submodules` option will automatically initialize and update all the submodules:

```
gcuser:~$ git clone --recurse-submodules https://github.com/geoschem/GCHP.git ~/GCHP
gcuser:~$ cd ~/GCHP
```

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.

```
gcuser:~/GCHP$ git log -n 1
```

Tip: To use an older GCHP version (e.g. 14.0.0), follow these additional steps:

```
gcuser:~/GCHP$ git checkout tags/14.0.0                  # Points HEAD to the tag "14.
↪0.0"
gcuser:~/GCHP$ git branch version_14.0.0                 # Creates a new branch at␣
↪tag "14.0.0"
gcuser:~/GCHP$ git checkout version_14.0.0               # Checks out the version_14.
↪0.0 branch
gcuser:~/GCHP$ 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:

```
gcuser:~/GCHP$ 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 `./createRunDir.sh` and answer the prompts:

```
gcuser:~/GCHP$ cd run/
gcuser:~/GCHP$ ./createRunDir.sh
```

1.3 3. Configure your build

Building GCHP will require 1.4G of storage space. You may build GCHP from within the run directory or from anywhere else on your system. Building from within the run directory is convenient because it keeps all build files in close proximity to where you will run GCHP. For this purpose the GCHP run directory includes a build directory called `build/`. However, you can create a build directory elsewhere, such as within the GCHP source code. In this guide we will do both, starting with building from the source code.

```
gcuser:~/GCHP$ mkdir ~/GCHP/build
gcuser:~/GCHP$ cd ~/GCHP/build
```

Initialize your build directory by running **`cmake`**, passing it the path to your source code. Make sure you have loaded all libraries required for GCHP prior to this step.

```
gcuser:~/GCHP/build$ cmake ~/GCHP
```

Now you can configure *build options*. These are persistent settings that are saved to your build directory. A useful build option is `-DRUNDIR`. This option lets you specify one or more run directories that GCHP is "installed" to, meaning where the executable is copied, when you do **`make install`**. Configure your build so it installs GCHP to the run directory you created in Step 2.

```
gcuser:~/GCHP/build$ cmake . -DRUNDIR="/path/to/your/run/directory"
```

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

If you decide instead to build GCHP in your run directory you can do all of the above in one step. This makes use of the `CodeDir` symbolic link in the run directory:

```
gcuser:/path/to/your/run/directory/$ cd build
gcuser:/path/to/your/run/directory/build$ cmake ../CodeDir -DRUNDIR=..
```

GEOS-Chem has a number of optional compiler flags you can add here. For example, to compile with RRTMG:

```
gcuser:/path/to/your/run/directory/build$ cmake ../CodeDir -DRUNDIR=.. -DRRTMG=y
```

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

```
gcuser:/path/to/your/run/directory/build$ cmake ../CodeDir -DRUNDIR=.. -DCMAKE_BUILD_
↪TYPE=Debug
```

See the GEOS-Chem documentation for more information on compiler flags.

1.4 4. Compile and install

Compiling GCHP takes about 20 minutes, but it can vary depending on your system, your compiler, and your compiler flags. To maximize build speed you should compile GCHP in parallel using as many cores as are available. Do this with the `-j` flag:

```
gcuser:~/GCHP/build$ make -j
```

Upon successful compilation, install the compiled executable to your run directory (or directories):

```
gcuser:~/GCHP/build$ make install
```

This copies `bin/gchp` 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`**.

1.5 5. Configure your run directory

Now, navigate to your run directory:

```
$ cd path/to/your/run/directory
```

Commonly changed simulation settings, such as grid resolution, run duration, and number of cores, are set in `setCommonRunSettings.sh`. You should review this file as it explains most settings. Note that `setCommonRunSettings.sh` is actually a helper script that updates other configuration files. You therefore need to run it to actually apply the settings:

```
$ vim setCommonRunSettings.sh          # edit simulation settings here
$ ./setCommonRunSettings.sh            # applies the updated settings
```

Simulation start date is set in `cap_restart`. Run directories come with this file filled in based on date of the initial restart file in subdirectory `Restarts`. You can change the start date only if you have a restart file for the new date in `Restarts`. A symbolic link called `gchp_restart.nc4` points to the restart file for the date in `cap_restart` and the grid resolution in `setCommonRunSettings.sh`. You need to set this symbolic link before running:

```
$ ./setRestartLink.sh                  # sets symbolic link to target file in␣
↪Restarts
```

If you used an environment file to load libraries prior to building GCHP then you should load that file prior to running. A simple way to make sure you always use the correct combination of libraries is to set the GCHP environment symbolic link `gchp.env` in the run directory:

```
$ ./setEnvironment.sh /path/to/env/file # sets symbolic link gchp.env
$ source gchp.env                       # applies the environment settings
```

1.6 6. Run GCHP

Running GCHP is slightly different depending on your MPI library (e.g., OpenMPI, Intel MPI, MVAPICH2, etc.) and scheduler (e.g., SLURM, LSF, etc.). If you aren't familiar with running MPI programs on your system, see *Running GCHP* in the user guide, or ask your system administrator.

Your MPI library and scheduler will have a command for launching MPI programs—it's usually something like **`mpirun`**, **`mpiexec`**, or **`srun`**. This is the command that you will use to launch the **`gchp`** executable. You'll have to refer to your system's documentation for specific instructions on running MPI programs, but generally it looks something like this:

```
$ mpirun -np 6 ./gchp   # example of running GCHP with 6 slots with OpenMPI
```

It's recommended you run GCHP as a batch job. This means that you write a script (usually bash) that configures and runs your GCHP simulation, and then you submit that script to your local job scheduler (SLURM, LSF, etc.). Example job scripts are provided in subdirectory `./runScriptSamples` in the run directory. That folder also includes an example script for running GCHP from the command line.

Several steps beyond running GCHP are included in the example run scripts. These include loading the environment, updating commonly changed run settings, and setting the restart file based on start time and grid resolution. In addition, the output restart file is moved to the `Restarts` subdirectory and renamed to include start date and grid resolution upon successful completion of the run.

Note: File `cap_restart` is over-written to contain the run end date upon successful completion of a GCHP run. This is done within GCHP and not by the run script. You can then easily submit a new GCHP run starting off where your last run left off. In addition, GCHP outputs a restart file to your run directory called `gcchem_internal_checkpoint`. This file is moved to subdirectory `Restarts` and renamed to include the date and grid resolution. This is done by the run script and technically is optional. We recommend doing this since it is is good for archiving (restart files will contain date and grid res) and enables use of the `./setRestartLink.sh` script to set the `gchp_restart.nc4` symbolic link.

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

CHAPTER

TWO

SYSTEM REQUIREMENTS

2.1 Software Requirements

To build and run GCHP your compute *environment* needs the following software:

- Git
- Make (or GNUMake)
- CMake version 3.13
- Compilers (C, C++, and Fortran):
  - Intel compilers version 19, or
  - GNU compilers version 10
- MPI (Message Passing Interface)
  - OpenMPI 4.0, or
  - IntelMPI, or
  - MVAPICH2, or
  - MPICH, or
  - other MPI libraries might work too
- HDF5
- NetCDF (with C, C++, and Fortran support)
- Earth System Modeling Framework (ESMF) version 8.4.2 recommended. Problems with 8.1 and prior have been reported.

Your system administrator should be able to tell you if this software is already available on your cluster, and if so, how to activate it. If it is not already available, they might be able to build it for you. If you need to build GCHP's dependencies yourself, see the supplemental guide for building required software with Spack.

2.1.1 Installing ESMF

If you have all of the needed libraries except ESMF then you can download and build ESMF yourself. The ESMF git repository is available to clone from github.com/esmf-org/esmf. Use `git tag` to browse versions available and then `git checkout tags/tag_name` to checkout the version.

```
git clone https://github.com/esmf-org/esmf ESMF
cd ESMF
git tag
git checkout tags/v8.4.1
```

If you have previously downloaded ESMF you can use your same clone to checkout and build a new ESMF version. Use the same steps as above minus the first step of cloning.

Once you have downloaded ESMF and checked out the version you would like to build, browse the file `ESMF/README.md` to familiarize yourself with ESMF documentation. You do not need to visit the documentation for doing a basic build of ESMF following this tutorial. However, if you are interested in learning more about ESMF and its options then you can use this guide.

ESMF requires that you define environment variables `ESMF_COMPILER`, `ESMF_COMM`, and `ESMF_DIR`, and also export environment variables `CC`, `CXX`, `FC`, and `MPI_ROOT`. Set up an environment file that loads the needed libraries and also defines these environment variables. If you already have a GEOS-Chem environment file set up then you can copy it or repurpose it by including the environment variables needed for ESMF. Here is an example of what the library load and variable exports might look line in your environment file. This example uses GNU compilers and OpenMPI, but there are notes in the comments on how to use Intel instead.

```
module purge
module load gcc/10.2.0-fasrc01              # GNU compiler collection (C, C++, Fortran)
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 cmake/3.25.2-fasrc01            # CMake

export CC=gcc                           # C compiler (use icx for Intel)
export CXX=g++                          # C++ compiler (se icx for Intel)
export FC=gfortran                      # Fortran compiler (use ifort for Intel)
export MPI_ROOT=${MPI_HOME}             # Path to MPI library
export ESMF_COMPILER=gfortran           # Fortran compiler (use intel for Intel)
export ESMF_COMM=openmpi                # MPI (use intelmpi for IntelMPI)
export ESMF_DIR=/home/ESMF/ESMF         # Path to ESMF repository within a generic␣
↪directory called ESMF
```

You can create multiple ESMF builds. This is useful if you want to use different libraries for the same version of ESMF, or if you want to build different ESMF versions. To set yourself up to allow multiple builds you should also export environment variable `ESMF_INSTALL_PREFIX` and define it as a subdirectory within `ESMF_DIR`. Include details about that particular build to distinguish it from others. For example:

```
export ESMF_INSTALL_PREFIX=${ESMF_DIR}/INSTALL_ESMF8.4.1_gfortran10.2_openmpi4.1
```

Using this install in GCHP will require setting `ESMF_ROOT` to the install directory. Add the following line to your ESMF environment file if you plan on repurposing it for use with GCHP. Otherwise remember to add it to your GCHP environment file along with the assignment of `ESMF_INSTALL_PREFIX`.

```
export ESMF_ROOT=${ESMF_INSTALL_PREFIX}
```

Once you are ready to build execute the following commands:

```
$ source path/to/your/env/file
$ cd $ESMF_DIR
$ make -j &> compile.log
```

Once compilation completes check the end of `compile.log` to see if compilation was successful. You may run into known errors with compiling certain ESMF versions with GNU and Intel compilers. If you run into a problem with GNU you can try adding this to your environment file, resourcing it, and then rebuilding.

```
# ESMF may not build with GCC without the following work-around
# for a type mismatch error (https://trac.macports.org/ticket/60954)
if [[ "x${ESMF_COMPILER}" == "xgfortran" ]]; then
   export ESMF_F90COMPILEOPTS="-fallow-argument-mismatch -fallow-invalid-boz"
fi
```

If you run into a problem with Intel compilers then try the following.

```
# Make sure /usr/bin comes first in the search path, so that the build
# will find /usr/bin/gcc compiler, which ESMF uses for preprocessing.
# Also unset the ESMF_F90COMPILEOPTS variable, which is only needed for GNU.
if [[ "x${ESMF_COMPILER}" == "xintel" ]]; then
   export PATH="/usr/bin:${PATH}"
   unset ESMF_F90COMPILEOPTS
fi
```

Once you have a successful run then install ESMF using this command:

```
$ make install &> install.log
```

Check the end of file `install.log`. A message that installation was complete should be there if ESMF installation was a success.

If all went well there should now be a folder in the top-level ESMF directory corresponding to what you defined as environment variable `ESMF_INSTALL_PREFIX`. Archive your compile and install logs to that directory.

```
$ mv compile.log $ESMF_INSTALL_PREFIX
$ mv install.log $ESMF_INSTALL_PREFIX
```

Calling make builds ESMF and calling make install places the build into your install directory. In that folder the build files are placed within subdirectories such as bin and lib, among others. The install directory is not deleted when you clean ESMF source code with `make distclean` in the top-level ESMF directory. Therefore you can clean and rebuild ESMF with different combinations of libraries and versions in advance of needing them to build and run GCHP. Just remember to clean the source code and source the environment file you intend to use prior to creating a new build. Make sure you specify a different `${ESMF_INSTALL_PREFIX}` for each unique build so as not to overwrite others.

Below is a complete summary of build steps, including cleanup at the end and moving logs files and your environment file to the install directory for archiving. This is a complete list of command line steps assuming you have a functional environment file with correct install path and have checked out the version of ESMF you wish to build.

```
$ cd $ESMF_DIR
$ make distclean
$ source path/to/env/file/with/unique/ESMF_INSTALL_PREFIX
$ make &> compile.log
$ install $> install.log
$ mv compile.log $ESMF_INSTALL_PREFIX
$ mv install.log $ESMF_INSTALL_PREFIX
$ cp /path/to/your/env/file $ESMF_INSTALL_PREFIX
```

2.2 Hardware Requirements

High-end HPC infrastructure is not required to use GCHP effectively. Gigabit Ethernet and two nodes is enough for returns on performance compared to GEOS-Chem Classic.

2.2.1 Bare Minimum Requirements

- 6 cores
- 32 GB of memory
- 100 GB of storage for input and output data

Running GCHP on one node with as few as six cores is possible but we recommend this only for testing short low resolution runs such as running GCHP for the first time and for debugging. These bare minimum requirements are sufficient for running GCHP at C24. Please note that we recommend running at C90 or greater for scientific applications.

2.2.2 Recommended Minimum Requirements

- 2 nodes, preferably 24 cores per node
- Gigabit Ethernet (GbE) interconnect or better
- 100+ GB memory per node
- 1 TB of storage, depending on your input and output needs

These recommended minimums are adequate to effectively use GCHP in scientific applications. These runs should be at grid resolutions at or above C90.

2.2.3 Big Compute Recommendations

- 5–50 nodes, or more if running at C720 (12 km grid)
- >24 cores per node (the more the better), preferably Intel Xeon
- High throughput and low-latency interconnect, preferably InfiniBand if using 500 cores
- 1 TB of storage, depending on your input and output needs

These requirements can be met by using a high-performance-computing cluster or a cloud-HPC service like AWS.

2.2.4 General Hardware and Software Recommendations

- Hyper-threading may improve simulation throughput, particularly at low core counts
- MPI processes should be bound sequentially across cores and nodes. For example, a simulation using two nodes with 24 processes per node should bind ranks 0-23 on the first node and ranks 24-47 on the second node. This should be the default, but it's worth checking if your performance is lower than expected. With OpenMPI the `--report-bindings` argument will show you how processes are ranked and binded.
- If using IntelMPI include the following your environment setup to avoid a run-time error:

```
export I_MPI_ADJUST_GATHERV=3
export I_MPI_ADJUST_ALLREDUCE=12
```

- If using OpenMPI and a large number of cores (>1000) we recommend enabling the MAPL o-server functionality for writing restart files, thereby speeding up the model. This is set automatically when executing `setCommonRunSettings.sh` if using over 1000 cores. You can also toggle whether to use it manually in that file.

CHAPTER

THREE

KEY REFERENCES

- GEOS-Chem was first described in [[Bey et al., 2001]].
- HEMCO is described in [[Keller et al., 2014]] and [[Lin et al., 2021]].
- Columnar operators are described in [[Long et al., 2015]].
- GEOS-Chem High Performance (GCHP) is described in [[Eastham et al., 2018]].
- GCHP execution on the cloud and MPI considerations are described in [[Zhuang et al., 2020]].
- Grid-stretching is described in [[Bindle et al., 2021]].
- Major GCHP developments in v13 are described in [[Martin et al., 2022]].

References

CHAPTER

FOUR

DOWNLOAD THE MODEL

The GCHP source code is hosted at https://github.com/geoschem/GCHP. Clone the repository:

```
gcuser:~$ git clone --recurse-submodules https://github.com/geoschem/GCHP.git GCHP
```

The GCHP repository has submodules (other repositories that are nested inside the GCHP repository) that aren't automatically retrieved when you do **`git clone`**. The `--recurse-submodules` option tells Git to finish retrieving the source code for each submodule. It will also initialize and update each submodule's source code to the proper place in its version history.

By default, the source code will be on the `main` branch which is always the last official release of GCHP. Checking out the official release is recommended because it is a scientifically-validated version of the code and is easily citable. You can find the list of past and present GCHP releases here.

Tip: To use an older GCHP version (e.g. 14.0.0), follow these additional steps:

```
gcuser:~/GCHP$ git checkout tags/14.0.0                 # Points HEAD to the tag "14.
↪0.0"
gcuser:~/GCHP$ git branch version_14.0.0                # Creates a new branch at␣
↪tag "14.0.0"
gcuser:~/GCHP$ git checkout version_14.0.0              # Checks out the version_14.
↪0.0 branch
gcuser:~/GCHP$ 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:

```
gcuser:~/GCHP$ git tag
```

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

Before continuing, it is worth checking that the source code was retrieved correctly. Run **`git status`** to check that there are no differences:

```
gcuser:~/GCHP$ git status
HEAD detached at 14.0.0
nothing to commit, working tree clean
gcuser:~/GCHP$
```

The output of **`git status`** should confirm your GCHP version and that there are no modifications (nothing to commit, and a clean working tree). It also says that you are are in detached HEAD state, meaning you are not in a GCHP git software branch. This is true for all submodules in the model as well. If you wish to use version control to track your changes you must checkout a new branch to work on in the directory you will be developing.

Note: Compiling GCHP and creating a run directory are independent steps, and their order doesn't matter. A small exception is the *RUNDIR* build option, which controls the behaviour of **`make install`** which copies the GCHP executable to the run directory; however, this setting can be reconfigured at any time (e.g., after compiling and creating a run directory).

Here in the User Guide we describe compiling GCHP before we describe creating a run directory. This is so that conceptually the instructions have a linear flow. The Quickstart Guide, on the other hand, shows how to make a run directory prior to compiling.

Note: Another resource for GCHP build instructions is our YouTube tutorial.

CHAPTER

FIVE

COMPILE

There are three steps to building GCHP. The first is configuring your build, which is done with **cmake**; the second step is compiling, which is done with **make**. The third step is install, which is also done with **make**.

In the first step (build configuration), **cmake** finds GCHP's *software dependencies* on your system, and you can set *build options* like enabling/disabling components (such as RRTMG), setting paths to run directories, picking between debug or speed-optimizing compiler flags, etc. The second step (running **make**) compiles GCHP according your build configuration. The third step copies GCHP executable to an appropriate location, such as one or more run directories if you specify them.

Important: These instructions assume you have loaded a computing environment that satisfies *GCHP's software requirements* You can find instructions for building GCHP's dependencies yourself in the Spack instructions.

5.1 Create a build directory

A build directory is the working directory for a "build". Conceptually, a "build" is a case/instance of you compiling GCHP. A build directory stores configuration files and intermediate files related to the build. These files and generated and used by CMake, Make, and compilers. You can think a build directory like the blueprints for a construction project.

Create a new directory and initialize it as a build directory by running CMake. When you initialize a build directory, the path to the source code is a required argument:

```
gcuser:~$ cd ~/Code.GCHP
gcuser:~/Code.GCHP$ mkdir build              # create a new directory
gcuser:~/Code.GCHP$ cd build
gcuser:~/Code.GCHP/build$ cmake ~/Code.GCHP  # initialize the current dir as a build␣
↪dir
-- The Fortran compiler identification is GNU 9.2.1
-- The CXX compiler identification is GNU 9.2.1
-- The C compiler identification is GNU 9.2.1
-- Check for working Fortran compiler: /usr/bin/f95
-- Check for working Fortran compiler: /usr/bin/f95  -- works
...
-- Configuring done
-- Generating done
-- Build files have been written to: /src/build
gcuser:~/Code.GCHP/build$
```

If your **cmake** output is similar to the snippet above, and it says configuring & generating done, then your configuration was successful and you can move on to *compiling* or *modifying build settings*. If you got an error, don't worry,

that just means the automatic configuration failed. To fix the error you might need to tweak settings with more **cmake** commands, or you might need to modify your environment and run **cmake** again to retry the automatic configuration.

If you want to restart configuring your build from scratch, delete your build directory. Note that the name and location of your build directory doesn't matter, but a good name is `build/`, and a good place for it is the top-level of your source code.

5.1.1 Resolving initialization errors

If your last step was successful, *skip this section*.

Even if you got a **cmake** error, your build directory was initialized. This means from now on, you can check if the configuration is fixed by running

```
gcuser:~/Code.GCHP/build$ cmake .     # "." because the cwd is the build dir
```

To resolve your errors, you might need to modify your environment (e.g., load different software modules), or give CMake a hint about where some software is installed. Once you identify the problem and make the appropriate update, run **cmake .** to see if the error is fixed.

To start troubleshooting, read the **cmake** output in full. It is human-readable, and includes important information about how the build was set up on your system, and specifically what error is preventing a successful configuration (e.g., a dependency that wasn't found, or a compiler that is broken). To begin troubleshooting you should check that:

- check that the compilers are what you expect (e.g., GNU 9.2, Intel 19.1, etc.)
- check that dependencies like MPI, HDF5, NetCDF, and ESMF were found
- check for obvious errors/incompatibilities in the paths to "Found" dependencies

---

Note: F2PY and ImageMagick are not required. You can safely ignore warnings about them not being found.

Most errors are caused by one or more of the following issues:

- The wrong compilers were chosen. Fix this by explicitly setting the compilers.
- The compiler's version is too old. Fix this by using newer compilers.
- A software dependency is missing. Fix this by loading the appropriate software. Some hints:
  - If HDF5 is missing, does **h5cc -show** or **h5pcc -show** work?
  - If NetCDF is missing, do **nc-config --all** and **nf-config --all** work?
  - If MPI is missing, does **mpiexec --help** work?
- A software dependency is loaded but it wasn't found automatically. Fix this by pointing CMake to the missing software/files with **cmake . -DCMAKE_PREFIX_PATH=/path/to/missing/files**.
  - If ESMF is missing, point CMake to your ESMF install with `-DCMAKE_PREFIX_PATH`
- Software modules that are not compatible. Fix this by loading compatible modules/dependencies/compilers. Some hints:
  - This often shows as an error message saying a compiler is "broken" or "doesn't work"
  - E.g. incompatibility #1: you're using GNU compilers but HDF5 is built for Intel compilers
  - E.g. incompatibility #2: ESMF was compiled for a different compiler, MPI, or HDF5

If you are stumped, don't hesitate to open an issue on GitHub. Your system administrators might also be able to help. Be sure to include `CMakeCache.txt` from your build directory, as it contains useful information for troubleshooting.

Note: If you get a CMake error saying "Could not find XXXX" (where XXXX is a dependency like ESMF, NetCDF, HDF5, etc.), the problem is that CMake can't automatically find where that library is installed. You can add custom paths to CMake's default search list by setting the `CMAKE_PREFIX_PATH` variable.

For example, if you got an error saying "Could not find ESMF", and ESMF is installed to `/software/ESMF`, you would do

```
gcuser:~/Code.GCHP/build$ cmake . -DCMAKE_PREFIX_PATH=/software/ESMF
...
-- Found ESMF: /software/ESMF/include (found version "8.1.0")
...
-- Configuring done
-- Generating done
-- Build files have been written to: /src/build
gcuser:~/Code.GCHP/build$
```

See the next section for details on setting variables like `CMAKE_PREFIX_PATH`.

Note: You can explicitly specify compilers by setting the `CC`, `CXX`, and `FC` environment variables. If the auto-selected compilers are the wrong ones, create a brand new build directory, and set these variables before you initialize it. E.g.:

```
gcuser:~/Code.GCHP/build$ cd ..
gcuser:~/Code.GCHP$ rm -rf build   # build dir initialized with wrong compilers
gcuser:~/Code.GCHP$ mkdir build    # make a new build directory
gcuser:~/Code.GCHP$ cd build
gcuser:~/Code.GCHP/build$ export CC=icc      # select "icc" as C compiler
gcuser:~/Code.GCHP/build$ export CXX=icpc    # select "icpc" as C++ compiler
gcuser:~/Code.GCHP/build$ export FC=icc      # select "ifort" as Fortran compiler
gcuser:~/Code.GCHP/build$ cmake ~/Code.GCHP  # initialize new build dir
-- The Fortran compiler identification is Intel 19.1.0.20191121
-- The CXX compiler identification is Intel 19.1.0.20191121
-- The C compiler identification is Intel 19.1.0.20191121
...
```

5.2 Configure your build

Build settings are controlled by **`cmake`** commands like:

```
$ cmake . -D<NAME>="<VALUE>"
```

where `<NAME>` is the name of the setting, and `<VALUE>` is the value you are assigning it. These settings are persistent and saved in your build directory. You can set multiple variables in the same command, and you can run **`cmake`** as many times as needed to configure your desired settings.

Note: The `.` argument is important. It is the path to your build directory which is `.` here.

No build settings are required. You can find the complete list of *GCHP's build settings here*. The most common setting is `RUNDIR`, which lets you specify one or more run directories to install GCHP to. Here, "install" refers to copying

the compiled executable, and some supplemental files with build settings, to your run directory/directories.

Note: You can update build settings after you compile GCHP. Simply rerun **`make`** and (optionally) **`make install`**, and the build system will automatically figure out what needs to be recompiled.

Since there are no required build settings, so here, we will stick with the default settings.

You should notice that when you run **`cmake`** it ends with:

```
...
-- Configuring done
-- Generating done
-- Build files have been written to: /src/build
```

This tells you that the configuration was successful, and that you are ready to compile.

5.3 Compile GCHP

You compile GCHP with:

```
gcuser:~/Code.GCHP/build$ make -j    # -j enables compiling in parallel
```

Note: You can add `VERBOSE=1` to see all the compiler commands.

Note: If you run out of memory while compiling, restrict the number of processes that can run concurrently (e.g., use `-j20` to restrict to 20 processes)

Compiling GCHP creates `./bin/gchp` (the GCHP executable). You can copy this executable to your run directory manually, or if you set the *RUNDIR* build option, you can do

```
gcuser:~/Code.GCHP/build$ make install  # Requires that RUNDIR build option is set
```

to copy the executable (and supplemental files) to your run directories.

Now you have compiled GCHP! You can move on to creating a run directory!

5.4 Recompiling

You need to recompile GCHP if you update a build setting or modify the source code. With CMake, you do not need to clean before recompiling. The build system automatically figures out which files need to be recompiled (it's usually a small subset). This is known as incremental compiling.

To recompile GCHP, simply do

```
gcuser:~/Code.GCHP/build$ make -j    # -j enables compiling in parallel
```

and then optionally, **make install**.

Note: GNU compilers recompile GCHP faster than Intel compilers. This is because of how **gfortran** formats Fortran modules files (`*.mod` files). Therefore, if you want to be able to recompile quickly, consider using GNU compilers.

5.5 GCHP build options

These are persistent build setting that are set with **cmake** commands like

```
$ cmake . -D<NAME>="<VALUE>"
```

where <NAME> is the name of the build setting, and <VALUE> is the value you are assigning it. Below is the list of build settings for GCHP.

RUNDIR Paths to run directories where **make install** installs GCHP. Multiple run directories can be specified by a semicolon separated list. A warning is issues 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.

CMAKE_BUILD_TYPE The build type. Valid values are `Release`, `Debug`, and `RelWithDebInfo`. Set this to `Debug` if you want to build in debug mode.

CMAKE_PREFIX_PATH Extra directories that CMake will search when it's looking for dependencies. Directories in `CMAKE_PREFIX_PATH` have the highest precedence when CMake is searching for dependencies. Multiple directories can be specified with a semicolon-separated list.

GEOSChem_Fortran_FLAGS_<COMPILER_ID> Compiler options for GEOS-Chem for all build types. Valid values for <COMPILER_ID> are `GNU` and `Intel`.

GEOSChem_Fortran_FLAGS_<BUILD_TYPE>_<COMPILER_ID> Additional compiler options for GEOS-Chem for build type <BUILD_TYPE>.

HEMCO_Fortran_FLAGS_<COMPILER_ID> Same as `GEOSChem_Fortran_FLAGS_<COMPILER_ID>`, but for HEMCO.

HEMCO_Fortran_FLAGS_<BUILD_TYPE>_<COMPILER_ID> Same as `GEOSChem_Fortran_FLAGS_<BUILD_TYPE>_<` but for HEMCO.

RRTMG Switch to enable the RRTMG component. Set value to `y` to turn on.

FASTJX Switch to enable the legacy FAST-JX v7.0 photolysis mechanism. Set value `y` to turn on FAST-JX and turn off Cloud-J.

OMP Switch to enable/disable OpenMP multithreading. As is standard in CMake (see if documentation) valid values are `ON`, `YES`, `Y`, `TRUE`, or `1` (case-insensitive) and valid false values are their opposites.

INSTALLCOPY Similar to `RUNDIR`, except the directories do not need to be run directories.

CHAPTER

SIX

CREATE A RUN DIRECTORY

Run directories are created with the `createRunDir.sh` script in the `run/` subdirectory of the source code. Run directories are version-specific, so you need to create new run directories for every GEOS-Chem version. The gist of creating a run directory is simple: navigate to the `run/` subdirectory, run `./createRunDir.sh`, and answer the prompts:

```
gcuser:~$ cd GCHP/run
gcuser:~/GCHP/run$ ./createRunDir.sh
... <answer the prompts> ...
```

Important: Use *absolute paths* when responding to prompts.

If you are unsure what a prompt is asking, see their explanations below, or ask a question on GitHub. After following all prompts a run directory should be created for you with a confirmation message, and, you can move on to the next section.

6.1 Explanations of Prompts

Below are detailed explanations of the prompts in `./createRunDir.sh`.

6.1.1 Enter ExtData path

The first time you create a GCHP run directory on your system you will be prompted to register as a GEOS-Chem user. Please provide this information so that we can track GEOS-Chem user groups around the world and get to know what GEOS-Chem is used for.

Following registration you will be prompted for a path to GEOS-Chem shared data directories. The path should include the name of your `ExtData/` directory and should not contain symbolic links. The path you enter will be stored in file `.geoschem/config` in your home directory as environment variable `GC_DATA_ROOT`. If that file does not already exist it will be created for you. When creating additional run directories you will only be prompted again if the file is missing or if the path within it is not valid.

```
------------------------------------------------------------
Enter path for ExtData:
------------------------------------------------------------
```

6.1.2 Choose a simulation type

Enter the integer number that is next to the simulation type you want to use.

```
------------------------------------------------------------
Choose simulation type:
------------------------------------------------------------
  1. Full chemistry
  2. TransportTracers
  3. CO2 w/ CMS-Flux emissions
  4. Tagged O3
  5. Carbon
>>>
```

If creating a full chemistry run directory you will be given additional options. Enter the integer number that is next to the simulation option you want to run.

```
------------------------------------------------------------
Choose additional simulation option:
------------------------------------------------------------
  1. Standard
  2. Benchmark
  3. Complex SOA
  4. Marine POA
  5. Acid uptake on dust
  6. TOMAS
  7. APM
  8. RRTMG
>>>
```

6.1.3 Choose meteorology source

Enter the integer number that is next to the input meteorology source you would like to use. The primary difference between GEOS-FP and GEOS-FP native data is that the GEOS-FP native data includes the option to use C720 mass fluxes or derived winds.

```
------------------------------------------------------------
Choose meteorology source:
------------------------------------------------------------
  1. MERRA2 (Recommended)
  2. GEOS-FP
  3. GEOS-FP native data
>>>
```

6.1.4 Enter run directory path

Enter the target path where the run directory will be stored. You will be prompted to enter a new path if the one you enter does not exist.

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

6.1.5 Enter run directory name

Enter the run directory name, or accept the default. You will be prompted for a new name if a run directory of the same name already exists at the target path.

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

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

6.1.6 Enable version control (optional)

Enter whether you would like your run directory tracked with git version control. With version control you can keep track of exactly what you changed relative to the original settings. This is useful for trouble-shooting as well as tracking run directory feature changes you wish to migrate back to the standard model.

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

CHAPTER

SEVEN

DOWNLOAD INPUT DATA

Input data for GEOS-Chem is available at http://geoschemdata.wustl.edu/ExtData/.

The bashdatacatalog is the recommended for downloading and managing your GEOS-Chem input data. Refer to the bashdatacatalog's Instructions for GEOS-Chem Users. Below is a brief summary of using the bashdatacatalog for aquiring GCHP input data.

7.1 Install the bashdatacatalog

Install the bashdatacatalog with the following command. Follow the prompts and restart your console.

```
gcuser:~$ bash <(curl -s https://raw.githubusercontent.com/LiamBindle/bashdatacatalog/
↪main/install.sh)
```

Note: You can rerun this command to upgrade to the latest version.

7.2 Download Data Catalogs

Catalog files can be downloaded from http://geoschemdata.wustl.edu/ExtData/DataCatalogs/.

The catalog files define the input data collections that GEOS-Chem needs. There are four catalogs files:

- MeteorologicalInputs.csv – Meteorological input data collections
- ChemistryInputs.csv – Chemistry input data collections
- EmissionsInputs.csv – Emissions input data collections
- InitialConditions.csv – Initial conditions input data collections (restart files)

The latter 3 are version specific, so you need to download the catalogs for the version you intend to use (you can have catalogs for multiple versions at the same time).

Create a directory to house your catalog files in the top-level of your GEOS-Chem input data directory (commonly known as "ExtData"). You should create subdirectories for version-specific catalog files.

```
gcuser:~$ cd /ExtData  # navigate to GEOS-Chem data
gcuser:/ExtData$ mkdir InputDataCatalogs       # new directory for catalog files
gcuser:/ExtData$ mkdir InputDataCatalogs/13.3  # " for 13.3-specific catalogs␣
↪(example)
```

Next, download the catalog for the appropriate version:

```
gcuser:/ExtData$ cd InputDataCatalogs
gcuser:/ExtData/InputDataCatalogs$ wget http://geoschemdata.wustl.edu/ExtData/
↪DataCatalogs/MeteorologicalInputs.csv
gcuser:/ExtData/InputDataCatalogs$ cd 13.3
gcuser:/ExtData/InputDataCatalogs/13.3$ wget http://geoschemdata.wustl.edu/ExtData/
↪DataCatalogs/13.3/ChemistryInputs.csv
gcuser:/ExtData/InputDataCatalogs/13.3$ wget http://geoschemdata.wustl.edu/ExtData/
↪DataCatalogs/13.3/EmissionsInputs.csv
gcuser:/ExtData/InputDataCatalogs/13.3$ wget http://geoschemdata.wustl.edu/ExtData/
↪DataCatalogs/13.3/InitialConditions.csv
```

7.3 Fetching Metadata and Downloading Input Data

Important: You should always run bashdatacatalog commands from the top-level of your GEOS-Chem data directory (the directory with `HEMCO/`, `CHEM_INPUTS/`, etc.).

Before you can run `bashdatacatalog-list` commands, you need to fetch the metadata of each collection. This is done with the command `bashdatacatalog-fetch` whose arguments are catalog files:

```
gcuser:~$ cd /ExtData  # IMPORTANT: navigate to top-level of GEOS-Chem input data
gcuser:/ExtData$ bashdatacatalog-fetch InputDataCatalogs/*.csv InputDataCatalogs/**/*.
↪csv
```

Fetching downloads the latest metadata for every active collection in your catalogs. You should run `bashdatacatalog-fetch` whenever you add or modify a catalog, as well as periodically so you get updates to your collections (e.g., new meteorological data that is processed and added to the meteorological collections).

Now that you have fetched, you can run `bashdatacatalog-list` commands. You can tailor this command the generate various types of file lists using its command-line arguments. See `bashdatacatalog-list -h` for details. A common use case is generating a list of required input files that missing in your local file system.

```
gcuser:/ExtData$ bashdatacatalog-list -am -r 2018-06-30,2018-08-01 InputDataCatalogs/
↪*.csv InputDataCatalogs/**/*.csv
```

Here, `-a` means "all" files (temporal files and static files), `-m` means "missing" (list files that are absent locally), `-r START,END` is the date-range of your simulation (you should add an extra day before/after your simulation), and the remaining arguments are the paths to your catalog files.

The command can be easily modified so that it generates a list of missing files that is compatible with xargs curl to download all the files you are missing:

```
gcuser:/ExtData$ bashdatacatalog-list -am -r 2018-06-30,2018-08-01 -f xargs-curl␣
↪InputDataCatalogs/*.csv InputDataCatalogs/**/*.csv | xargs curl
```

Here, `-f xargs-curl` means the output file list should be formatted for piping into xargs curl.

7.4 See Also

- bashdatacatalog - Instructions for GEOS-Chem Users
- bashdatacatalog - List of useful commands
- GEOS-Chem Input Data Catalogs

CHAPTER

EIGHT

RUN THE MODEL

Note: Another useful resource for instructions on running GCHP is our YouTube tutorial.

This page presents the basic information needed to run GCHP as well as how to verify a successful run and reuse a run directory. A pre-run checklist is included here for easy reference. Please read the rest of this page to understand these steps.

8.1 Pre-run checklist

Prior to running GCHP, always run through the following checklist to ensure everything is set up properly.

1. Start date is set in `cap_restart`
2. Executable `gchp` is present.
3. All symbolic links are valid (no broken links)
4. Settings are correct in `setCommonRunSettings.sh`
5. `setRestartLink.sh` runs without error (ensures restart file is available)
6. If running via a job scheduler, totals cores are the same in `setCommonRunSettings.sh` and the run script
7. If running interactively, you have available locally the total cores in `setCommonRunSettings.sh`

8.2 How to run GCHP

You can run GCHP locally from within your run directory ("interactively") or by submitting your run to a job scheduler if one is available. Either way, it is useful to put run commands into a reusable script we call the run script. Executing the script will either run GCHP or submit a job that will run GCHP.

There is a symbolic link in the GCHP run directory called `runScriptSamples` that points to a directory in the source code containing example run scripts. Each file includes extra commands that make the run process easier and less prone to user error. These commands include:

1. Define a GCHP log file that includes start date configured in `cap_restart` in its name
2. Source environment file symbolic link `gchp.env`
3. Source config file `setCommonRunSettings.sh` to update commonly changed run settings
4. Set restart file symbolic link `gchp_restart.nc4` to target file in `Restarts` subdirectory for configured start date and grid resolution

5. Check that `cap_restart` now contains end date of your run
6. Move the output restart file to the `Restarts` subdirectory
7. Rename the output restart file to include run start date and grid resolution (format `GEOSChem.Restarts.YYYYMMDD_HHmmz.cN.nc4`)

8.2.1 Run interactively

Copy or adapt example run script `gchp.local.run` to run GCHP locally on your machine. Before running, make sure the total number of cores configured in `setCommonRunSettings.sh` is available locally. It must be at least 6.

To run, type the following at the command prompt:

```
$ ./gchp.local.run
```

Standard output will be displayed on your screen in addition to being sent to a log file with filename format `gchp.YYYYMMDD_HHmmSSz.log`. The HEMCO log output is also included in this file.

8.2.2 Run as batch job

Batch job run scripts will vary based on what job scheduler you have available. We offer a template batch job run script in the `runScriptSamples` subdirectory called `gchp.batch_job.sh`. This file contains examples for 3 types of job scheduler: SLURM, LSF, and PBS. You may copy and adapt this file for your system and preferences as needed.

At the top of all batch job scripts are configurable run settings. Most critically are requested # cores, # nodes, time, and memory. Figuring out the optimal values for your run can take some trial and error. See *hardware requirements* for guidance on what to choose. The more cores you request the faster GCHP will run given the same grid resolution. Configurable job scheduler settings and acceptable formats are often accessible from the command line. For example, type **`man sbatch`** to scroll through configurable options for SLURM, including various ways of specifying number of cores, time and memory requested.

To submit a batch job using a run script called `gchp.run` and the SLURM job scheduler:

```
$ sbatch gchp.run
```

To submit using Grid Engine instead of SLURM:

```
$ qsub gchp.run
```

If your computational cluster uses a different job scheduler, check with your IT staff or search the internet for how to configure and submit batch jobs on your system.

8.3 Verify a successful run

Standard output and standard error will be sent to a file specific to your scheduler, e.g. `slurm-jobid.out`, unless you configured your run script to send it to a different log file. Variable `log` is defined in the template run script as `gchp.YYYYMMDD_HHmmSSz.log` if you wish to use it. The date string in the log filename is the start date of your simulation as configured in `cap_restart`. This log is automatically used if you execute the interactive run script example `gchp.local.run`.

There are several ways to verify that your run was successful. Here are just a few:

1. The GCHP log file shows every timestep (search for `AGCM Date`) and ends with timing information.
2. NetCDF files are present in the `OutputDir/` subdirectory.
3. There is a restart file corresponding to your end date in the `Restarts` subdirectory.
4. The start date in `cap_restart` has been updated to your run end date.
5. The job scheduler log does not contain any error messages.
6. Output file `allPEs.log` does not contain any error messages.

If it looks like something went wrong, scan through the log files to determine where there may have been an error. Here are a few debugging tips:

- Review all of your configuration files to ensure you have proper setup, especially `setCommonRunSettings.sh`.
- "MAPL_Cap" or "CAP" errors in the run log typically indicate an error with your start time and/or duration. Check `cap_restart` and `setCommonRunSettings.sh`.
- "MAPL_ExtData" or "ExtData" errors in the run log indicate an error with your input files. Check `HEMCO_Config.rc` and `ExtData.rc`.
- "MAPL_HistoryGridComp" or "History" errors in the run log are related to your configured diagnostics. Check `HISTORY.rc`.
- Change the warnings and verbose options in `HEMCO_Config.rc` to 3 and rerun
- Change the `root_level` settings for `CAP.ExtData` in `logging.yml` to `DEBUG` and rerun
- Recompile the model with cmake option `-DCMAKE_BUILD_TYPE=Debug` and rerun.

If you cannot figure out where the problem is then please create a GCHP GitHub issue.

8.4 Reuse a run directory

8.4.1 Archive run output

Reusing a GCHP run directory comes with the perils of losing your old work. To mitigate this issue there is utility shell script `archiveRun.sh`. This script archives data output and configuration files to a subdirectory that will not be deleted if you clean your run directory.

Archiving runs is useful for other reasons as well, including:

- Save all settings and logs for later reference after a run crashes
- Generate data from the same executable using different run-time settings for comparison, e.g. c48 versus c180
- Run short runs to compare for debugging

To archive a run, pass the archive script a descriptive subdirectory name where data will be archived. For example:

```
$ ./archiveRun.sh 1mo_c24_24hrdiag
```

Which files are copied and to where will be displayed on the screen. Diagnostic files in the `OutputDir/` directory will be moved rather than copied so as not to duplicate large files. Restart files will not be archived. If you would like include restart files in the archive you must manually copy or move them.

8.4.2 Clean a run directory

It is good practice to clean your run directory prior to your next run if starting on the same date. This avoids confusion about what output was generated when and with what settings. To make run directory cleaning simple we provide utility shell script `cleanRunDir.sh`. To clean the run directory simply execute this script.

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

All GCHP output diagnostic files and logs, including NetCDF files in `OutputDir/`, will be deleted. Restart files in the `Restarts` subdirectory will not be deleted.

CHAPTER

NINE

CONFIGURATION FILES

All GCHP run directories have default simulation-specific run-time settings that are set in the configuration files. This section gives an high-level overview of all run directory configuration files used at run-time in GCHP, followed by links to detailed descriptions if you wish to learn more.

Note: The many configuration files in GCHP can be overwhelming. However, you should be able to accomplish most if not all of what you wish to configure from one place in `setCommonRunSettings.sh`. That file is a bash script used to configure settings in other files from one place.

9.1 High-level summary

This high-level summary of GCHP configuration files gives a short description of each file.

`setCommonRunSettings.sh` This file is a bash script that includes commonly changed run settings. It makes it easier to manage configuring GCHP since settings can be changed from one file rather than across multiple configuration files. When this file is executed it updates settings in other configuration files, overwriting what is there. Please get very familiar with the options in `setCommonRunSettings.sh` and be conscientious about not updating the same setting elsewhere.

`GCHP.rc` Controls high-level aspects of the simulation, including grid type and resolution, core distribution, stretched-grid parameters, timesteps, and restart filename.

`CAP.rc` Controls parameters used by the highest level gridded component (CAP). This includes simulation run time information, name of the Root gridded component (GCHP), config filenames for Root and History, and toggles for certain MAPL logging utilities (timers, memory, and import/export name printing).

`ExtData.rc` Config file for the MAPL ExtData component. Specifies input variable information, including name, regridding method, read frequency, offset, scaling, and file path. All GCHP imports must be specified in this file. Toggles at the top of the file enable MAPL ExtData debug prints and using most recent year if current year of data is unavailable. Default values may be used by specifying file path `/dev/null`.

`geoschem_config.yml` Primary config file for GEOS-Chem. Same file format as in GEOS-Chem Classic but containing only options relevant to GCHP.

`HEMCO_Config.rc` Contains emissions information used by HEMCO. Same function as in GEOS-Chem Classic except only HEMCO name, species, scale IDs, category, and hierarchy are used. Diagnostic frequency, file path, read frequency, and units are ignored, and are instead stored in GCHP config file `ExtData.rc`. All HEMCO variables listed in `HEMCO_Config.rc` for enabled emissions must also have an entry in `ExtData.rc`.

`input.nml` Namelist used in advection for domain stack size and stretched grid parameters.

logging.yml Config file for the NASA pFlogger package included in GCHP for logging. This package uses a hierarchy of loggers, such as info, warnings, error, and debug, to extract non-GEOS-Chem information about GCHP runs and print it to log file `allPEs.log`.

HISTORY.rc Config file for the MAPL History component. It configures diagnostic output from GCHP.

HEMCO_Diagn.rc Contains information mapping `HISTORY.rc` diagnostic names to HEMCO containers. Same function as in GEOS-Chem Classic except that not all items in `HEMCO_Diagn.rc` will be output; only emissions listed in `HISTORY.rc` will be included in diagnostics. All GCHP diagnostics listed in `HISTORY.rc` that start with Emis, Hco, or Inv must have a corresponding entry in `HEMCO_Diagn.rc`.

9.2 Additional resources

Detailed information about each file can be found in the below list of links. You can also reach these pages by continuing with the "next" buttons in this user guide.

9.2.1 setCommonRunSettings.sh

This file is a bash script to specify run-time values for commonly changed settings and update other configuration files that set them. This is intended as a helper script to make configuring GCHP runs easier. There are four sections of the file: (1) configuration, (2) error checks, (3) helper functions, and (4) update files.

The configuration section is usually the only part of the file you need to look at. The configuration section itself is divided into two parts. The first part contains the most frequently changed settings. Categories are:

- Compute resources
- Grid resolution
- Stretched grid
- Simulation duration
- GEOS-Chem components
- Diagnostics
- Mid-run checkpoint files

The second part contains settings that are less frequently changed but that are still convenient to update from one place. These include:

- Model phase (e.g. adjoint)
- Timesteps
- Online dust mass tuning factor
- Domain decomposition

The entire configuration section contains many comments with instructions on how to change the settings and what the options are. Please see that file for more information.

The error checks section is a holdover from the earlier design of GCHP run directories. This section checks to make sure your run directory settings make sense and will not cause an early crash due to a simple mistake, such as a core count that is not divisible by 6. This section will be moving to file `checkRunSetting.sh` that is in your run directory but that is currently just a placeholder. Eventually that script will be able to be run separately from `setCommonRunSettings.sh` as a quick check prior to doing a run.

The helper functions section contains several functions to simplify updating configuration files based on the settings you specified in the configurations section earlier in the script. Some of the functions are general, such as printing a message during file update based on if the script was passed optional argument `--verbose`. Other functions are specialized, such as replacing met-field read frequency in `ExtData.rc` based on the model timestep.

The update files section changes settings in other configuration files based on what you set in the configurables section. You can browse this section to see exactly what files are changed. You can also view this information by running the script with the `--verbose` option.

Using the `setCommonRunSettings.sh` script is technically optional to run GCHP. However, we highly recommend using it to avoid mistakes in your run directory setup. Knowing which configuration files need to be changed for which run-time settings and then changing them all manually is cumbersome and error-prone. We hope that using this file will make it easier to use GCHP without making mistakes.

9.2.2 GCHP.rc

`GCHP.rc` is the resource configuration file for the ROOT component within GCHP. The ROOT gridded component includes three children gridded components, including one each for GEOS-Chem, FV3 advection, and the data utility environment needed to support them.

NX, NY Number of grid cells in the two MPI sub-domain dimensions. NX * NY must equal the number of CPUs. NY must be a multiple of 6.

GCHP.GRID_TYPE Type of grid GCHP will be run at. Should always be Cubed-Sphere.

GCHP.GRIDNAME Descriptive grid label for the simulation. The default grid name is PE24x144-CF. The grid name includes how the pole is treated, the face side length, the face side length times six, and whether it is a Cubed Sphere Grid or Lat/Lon. The name PE24x144-CF indicates polar edge (PE), 24 cells along one face side, 144 for 24*6, and a cubed-sphere grid (CF). Many options here are defined in MAPL_Generic.

Note: Must be consistent with IM and JM.

GCHP.NF Number of cubed-sphere faces. This is set to 6.

GCHP.IM_WORLD Number of grid cells on the side of a single cubed sphere face.

GCHP.IM Number of grid cells on the side of a single cubed sphere face.

GCHP.JM Number of grid cells on one side of a cubed sphere face, times 6. This represents a second dimension if all six faces are stacked in a 2-dimensional array. Must be equal to IM*6.

GCHP.LM Number of vertical grid cells. This must be equal to the vertical resolution of the offline meteorological fields (72) since MAPL cannot regrid vertically.

GCHP.STRETCH_FACTOR Ratio of configured global resolution to resolution of targeted high resolution region if using stretched grid.

GCHP.TARGET_LON Target longitude for high resolution region if using stretched grid.

GCHP.TARGET_LAT Target latitude for high resolution region if using stretched grid.

IM Same as GCHP.IM and GCHP.IM_WORLD.

JM Same as GCHP.JM.

LM Same as GCHP.LM.

GEOChem_CTM If set to 1, tells FVdycore that it is operating as a transport model rather than a prognostic model.

AdvCore_Advection Toggles offline advection. 0 is off, and 1 is on.

DYCORE Should either be set to OFF (default) or ON. This value does nothing, but MAPL will crash if it is not declared.

HEARTBEAT_DT The timestep in seconds that the DYCORE Component should be called. This must be a multiple of HEARTBEAT_DT in `CAP.rc`.

SOLAR_DT The timestep in seconds that the SOLAR Component should be called. This must be a multiple of HEARTBEAT_DT in `CAP.rc`.

IRRAD_DT The timestep in seconds that the IRRAD Component should be called. ESMF checks this value during its timestep check. This must be a multiple of HEARTBEAT_DT in `CAP.rc`.

RUN_DT The timestep in seconds that the RUN Component should be called.

GCHPchem_DT The timestep in seconds that the GCHPchem Component should be called. This must be a multiple of HEARTBEAT_DT in `CAP.rc`.

RRTMG_DT The timestep in seconds that RRTMG should be called. This must be a multiple of HEARTBEAT_DT in `CAP.rc`.

DYNAMICS_DT The timestep in seconds that the FV3 advection Component should be called. This must be a multiple of HEARTBEAT_DT in `CAP.rc`.

SOLARAvrg, IRRADAvrg Default is 0.

GCHPchem_REFERENCE_TIME HHMMSS reference time used for GCHPchem MAPL alarms.

PRINTRC Specifies which resource values to print. Options include 0: non-default values, and 1: all values. Default setting is 0.

PARALLEL_READFORCING Enables or disables parallel I/O processes when writing the restart files. Default value is 0 (disabled).

NUM_READERS, NUM_WRITERS Number of simultaneous readers. Should divide evenly unto NY. Default value is 1.

BKG_FREQUENCY Active observer when desired. Default value is 0.

RECORD_FREQUENCY Frequency of periodic restart file write in format HHMMSS.

RECORD_REF_DATE Reference date(s) used to determine when to write periodic restart files.

RECORD_REF_TIME Reference time(s) used to determine when to write periodic restart files.

GCHOchem_INTERNAL_RESTART_FILE The filename of the internal restart file to be written.

GCHPchem_INTERNAL_RESTART_TYPE The format of the internal restart file. Valid types include pbinary and pnc4. Only use pnc4 with GCHP.

GCHPchem_INTERNAL_CHECKPOINT_FILE The filename of the internal checkpoint file to be written.

GCHPchem_INTERNAL_CHECKPOINT_TYPE The format of the internal checkstart file. Valid types include pbinary and pnc4. Only use pnc4 with GCHP.

GCHPchem_INTERNAL_HEADER Only needed when the file type is set to pbinary. Specifies if a binary file is self-describing.

DYN_INTERNAL_RESTART_FILE The filename of the DYNAMICS internal restart file to be written. Please note that FV3 is not configured in GCHP to use an internal state and therefore will not have a restart file.

DYN_INTERNAL_RESTART_TYPE The format of the DYNAMICS internal restart file. Valid types include pbinary and pnc4. Please note that FV3 is not configured in GCHP to use an internal state and therefore will not have a restart file.

DYN_INTERNAL_CHECKPOINT_FILE The filename of the DYNAMICS internal checkpoint file to be written. Please note that FV3 is not configured in GCHP to use an internal state and therefore will not have a restart file.

DYN_INTERNAL_CHECKPOINT_TYPE The format of the DYNAMICS internal checkpoint file. Valid types include pbinary and pnc4. Please note that FV3 is not configured in GCHP to use an internal state and therefore will not have a restart file.

DYN_INTERNAL_HEADER Only needed when the file type is set to pbinary. Specifies if a binary file is self-describing.

RUN_PHASES GCHP uses only one run phase. The GCHP gridded component for chemistry, however, has the capability of two. The two-phase feature is used only in GEOS.

HEMCO_CONFIG Name of the HEMCO configuration file. Default is `HEMCO_Config.rc` in GCHP.

STDOUT_LOGFILE Log filename template. Default is `PET%%%%%.GEOSCHEMchem.log`. This file is not actually used for primary standard output.

STDOUT_LOGLUN Logical unit number for stdout. Default value is 700.

MEMORY_DEBUG_LEVEL Toggle for memory debugging. Default is 0 (off).

WRITE_RESTART_BY_OSERVER Determines whether MAPL restart write should use o-server. This must be set to YES for high core count (>1000) runs to avoid hanging during file write. It is NO by default.

9.2.3 CAP.rc

`CAP.rc` is the configuration file for the top-level gridded component called CAP. This gridded component can be thought of as the primary driver of GCHP. Its config file handles general runtime settings for GCHP including time parameters, performance profiling routines, and system-wide timestep (hearbeat). Combined with output file `cap_restart`, `CAP.rc` configures the exact dates for the next GCHP run.

ROOT_NAME Sets the name MAPL uses to initialize the ROOT child gridded component component within CAP. CAP uses this name in all operations when querying and interacting with ROOT. It is set to GCHP.

ROOT_CF Resource configuration file for the ROOT component. It is set to `GCHP.rc`.

HIST_CF Resource configuration file for the MAPL HISTORY gridded component (another child gridded component of CAP). It is set to `HISTORY.rc`.

BEG_DATE Simulation begin date in format YYYYMMDD hhmmss. This parameter is overrided in the presence of output file `cap_restart` containing a different start date.

END_DATE Simulation end date in format YYYYMMDD hhmmss. If BEG_DATE plus duration (JOB_SGMT) is before END_DATE then simulation will end at BEG_DATE + JOB_SGMT. If it is after then simulation will end at END_DATE.

JOB_SGMT Simulation duration in format YYYYMMDD hhmmss. The duration must be less than or equal to the difference between start and end date or the model will crash.

HEARTBEAT_DT The timestep of the ESMF/MAPL internal clock, in seconds. All other timesteps in GCHP must be a multiple of HEARTBEAT_DT. ESMF queries all components at each heartbeat to determine if computation is needed. The result is based upon individual component timesteps defined in `GCHP.rc`.

MAPL_ENABLE_TIMERS Toggles printed output of runtime MAPL timing profilers. This is set to YES. Timing profiles are output at the end of every GCHP run.

MAPL_ENABLE_MEMUTILS Enables runtime output of the programs' memory usage. This is set to YES.

PRINTSPEC Allows an abbreviated model run limited to initializat and print of Import and Export state variable names. Options include:

- 0 (default): Off
- 1: Imports and Exports only

- 2: Imports only
- 3: Exports only

USE_SHMEM This setting is deprecated but still has an entry in the file.

REVERSE_TIME Enables running time backwards in CAP. Default is 0 (off).

USE_EXTDATA2G Enables using the next generation of MAPL ExtData (input component) which uses a yaml-format configuration file. Default is .FALSE. (off).

9.2.4 ExtData.rc

`ExtData.rc` contains input variable and file read information for GCHP. Explanatory information about the file is located at the top of the configuration file in all run directories. The file format is the same as that used in the GEOS model, and GMAO/NASA documentation for it can be found at the ExtData component page on the GEOS-5 wiki. Note that this file will be retired in GCHP v15.0 when MAPL version 3 is integrated into GCHP. It will be replaced with a YAML format file with a simplified and easier to understand interface.

The ins and outs of `ExtData.rc` can be hard to grasp, particular with regards to variable data updating, time interpolation, and file read. Reach out on the GCHP GitHub Issues page if you need help. See also the GCHP ReadTheDocs page on enabling ExtData prints for debugging. Enabling ExtData debug prints is the best way to determine what MAPL is doing for file I/O per import.

The following parameter is set at the top of the file:

Ext_AllowExtrap Logical toggle to use data from nearest year available, including meteorology if files for the simulation year are not found. This is set to true for GCHP. Note that GEOS-Chem Classic accomplishes the same effect but with more flexibility in `HEMCO_Config.rc`, and the entries of `HEMCO_Config.rc` which do this are ignored in GCHP.

The rest of the file contains whitespace-delimited lines. Each line describes one data variable imported to the model from an external file. Columns are as follows in order from left to right:

Name Name of the field stored in the MAPL Imports container. This is independent of the name of the data field in the input file. For the case of entries that also appear in `HEMCO_Config.rc` it is also the name of the HEMCO emissions container (left-most column in that file). For those fields it is used to match scaling and masking information in `HEMCO_Config.rc` with file I/O information in `ExtData.rc`. All file I/O information `HEMCO_Config.rc`, including filename, units, dimensions, regridding, and read frequency are ignored by GCHP.

Units Unit string of the import. This entry is informational only.

Clim Whether the data is climatology. Enter `Y` if the data is a 12 month climatology, enter year if the data is daily climatology (i.e. `2019`), `D` if the file is monthly day-of-week scale factors (7 values for each of 12 months), or `N` for all other cases. If you specify monthly climatology then the data must be stored in either 1 or 12 files.

Conservative Method to regrid the input data to the simulation grid. Enter `Y` to use mass conserving regridding, `F;{VALUE}` for fractional regridding, or `N` to use non-conervative bilinear regridding.

Refresh Time template for updating data. This tells MAPL when to look for new data values. It stores previous and next time data in what are called left and right brackets. There are several options for specifying refresh:

- `-` : Update variable data only once. Use this if the data is constant in time.
- `0` : Update variable data at every timestep using linear interpolation. For example, if the data is hourly then MAPL will linearly interpolate between the previous and next hour's data for every timestep.
- `0:003000` (or other HHMMSS specification for hours, minutes, seconds) : Use specified time offset (i.e. 30 minutes in this example) for setting previous and next time, and interpolate every timestep between the two. This is useful if, for example, you have time-averaged hourly data and you want the previous and next

times to update half-way between the hour. This format is used for meteorology fields that are interpolated every timestep, specifically temperature and surface pressure.

- `F0:003000` (or other HHMMSS specification for hours, minutes, seconds) : Like the previous option except there is no time interpolation. This format is used for meteorology fields that are not time-interpolated, such as cloud fraction.
- `%y4-%m2-%h2T%h2:%n2:00` (or other combination of time tokens) : Update variable data when time tokens change. Interpreting this entry gets a little tricky. The data will be updated when the time tokens change, not the hard-coded times. For example, a template in the form `%y4-%m2-%d2T12:00:00` changes at the start of each day because that is when the evaluation of `%y4-%m2-%d2` changes. While the variable will be updated at the start of a new day (e.g. at time 2019-01-02 00:00:00), the time used for reading and interpolation is hour 12 of that day. You can similar hard-code year, month, day, or hour if you always want to use a constant value for that field.
- `F%y4-%m2-%h2T%h2:%n2:00` (or other combination of time tokens) : Like the previous option except that there is no time interpolation.

Offset Factor Value the data will be shifted by upon read. Use `none` for no shifting.

Scale Factor Value the data will be scaled by upon read. This is useful if you want to convert units upon read, such as from `Pa` to `hPa`. Use `none` for no scaling.

External File Variable Name of the variable to read in the netCDF data file.

External File Template Path to the netCDF data file, including time tokens as needed (`%y4` for year, `%m2` for month, `%d2` for day, `%h2` for hour, `%n2` for minutes). If there are no time tokens in the template name then ExtData will assume that all the data is in one file. If you wish to ignore an entry in `ExtData.rc` (i.e. not read the data at all since you will not use it) then put `/dev/null`. This will save processing time.

Reference Time and Period (optional) Period of data with reference time. This optional entry is useful if you have data frequency that is offset from midnight. For example, 3-hourly data available for times 1:30, 4:30, 7:30, etc. The reference time could be specified as `2000-01-01T01:30:00P03:00`. The first part (before `P`) is the reference date (must be on or before your simulation start), and the second part (after `P`) is the period of data availability (in this case 3 hours). This can be used in combination with the file template containing hours and minutes. It tells MAPL to only read the file at times that are regular 3 hr intervals from the reference date and time. Not including this would cause MAPL to read the file every minute if the file template contains the `n2` time token.

9.2.5 geoschem_config.yml

Information about the `geoschem_config.yml` file is the same as for GEOS-Chem Classic with a few exceptions. See the GEOS-Chem ReadTheDocs configuration files section for an overview of the file.

The `geoschem_config.yml` file used in GCHP is different in the following ways:

- Start/End datetimes are ignored. Set this information in `CAP.rc` instead.
- Root data directory is ignored. All data paths are specified in `ExtData.rc` instead with the exception of the FAST-JX data directory which is still listed (and used) in `geoschem_config.yml`.
- Met field is ignored. Met field source is described in file paths in `ExtData.rc`.
- GC classic timers setting is ineffectual. GEOS-Chem Classic timers code is not compiled when building GCHP.

Other parts of the GEOS-Chem Classic `geoschem_config.yml` file that are not relevant to GCHP are simply not included in the file that is copied to the GCHP run directory.

9.2.6 HEMCO_Config.rc

Like `geoschem_config.yml`, information about the `HEMCO_Config.rc` file is the same as for GEOS-Chem Classic with a few exceptions. Refer to the HEMCO documentation for an overview of the file.

Some content of the `HEMCO_Config.rc` file is ignored by GCHP. This is because MAPL ExtData handles file input rather than HEMCO in GCHP.

Items at the top of the file that are ignored include:

- ROOT data directory path
- METDIR path
- DiagnPrefix
- DiagnFreq
- Wildcard

In the BASE EMISSIONS section and beyond, columns that are ignored include:

- sourceFile
- sourceVar
- sourceTime
- C/R/E
- SrcDim
- SrcUnit

All of the above information is specified in file `ExtData.rc` instead with the exception of diagnostic prefix and frequency. Diagnostic filename and frequency information is specified in `HISTORY.rc`.

9.2.7 input.nml

input.nml controls specific aspects of the FV3 dynamical core used for advection. Entries in input.nml are described below.

&fms_nml Header for the FMS namelist which includes all variables directly below the header.

print_memory_usage Toggles memory usage prints to log. However, in practice turning it on or off does not have any effect.

domain_stack_size Domain stack size in bytes. This is set to 20000000 in GCHP to be large enough to use very few cores in a high resolution run. If the domain size is too small then you will get an "mpp domain stack size overflow error" in advection. If this happens, try increasing the domain stack size in this file.

&fv_core_nml Header for the finite-volume dynamical core namelist. This is commented out by default unless running on a stretched grid. Due to the way the file is read, commenting out the header declaration requires an additional comment character within the string, e.g. `#&fv#_core_nml`.

do_schmidt Logical for whether to use Schmidt advection. Set to .true. if using stretched grid; otherwise this entry is commented out.

stretch_fac Stretched grid factor, equal to the ratio of grid resolution in targeted high resolution region to the configured run resolution. This is commented out if not using stretched grid.

target_lat Target latitude of high resolution region if using stretched grid. This is commented out if not using stretched grid.

target_lon Target longitude of high resolution region if using stretched grid. This is commented out if not using stretched grid.

9.2.8 logging.yml

The `logging.yml` file is the configuration file for the pFlogger logging package used in GCHP. This package is a Fortran logger written and maintained by NASA Goddard. The pFlogger package is based on python logging and contains functions and classes that enable flexible event logging within GCHP components, including MAPL ExtData which handles input read.

GCHP logging is not the same as GEOS-Chem and HEMCO prints that go to the main GCHP log. It is hierarchical based on the severity of the event, with the level of severity per component used as criteria to print to the log file. The logging messages are sent to a separate file from the main GCHP log. The filename is specified in `logging.yml` as `allPEs.log` by default in the definition of the `mpi_shared` file handler.

Like the python logger, there are five levels of severity used to trigger messages. These are as follows, in order of most to least severe:

1. CRITICAL
2. ERROR
3. WARNING
4. INFO
5. DEBUG

These levels are hierarchical, meaning each level triggers writing messages for all events with greater or equal severity. For example, if you specify `CRITICAL` you will get only messages triggered with that criteria since it is the most severe level. If you instead specify `WARNING` then you will trigger all events categorized as `WARNING`, `ERROR`, and `CRITICAL`.

Different GCHP components can have different levels of severity. These components are listed in the `loggers` section of the file. This helps hone in on problems you are experiencing in a specific component by allowing you to increase logger messages for one component only. This is particularly useful for debugging the component called `CAP.EXTDATA` in `logging.yml` which corresponds to the MAPL component that handles reading and regridding input files. When you experience a problem reading input files we recommend that you set the logger level for this component to `DEBUG`.

In addition to setting severity level per component you can also specify severity level based on processor. There are two options: root thread only and all threads. The root thread only option is `root_level` in the configuration file and will only trigger messages if the event is executed by the root processor. Using this option keeps the log file size down and can make reading the file easier. We recommend setting this option to `DEBUG` when investigating problems with input files.

The all threads option will log events for all processors. Each message will be prefixed by the processor number, e.g. `0000` for the root thread, `0001` for the next, and so on. Using this option can make the file size very large and difficult to read. However, you can grep the file for a processor number to isolate events of just one thread of interest, such as the one that appears in error message traceback.

For more information on the GCHP logger, including more advanced features, see documentation at https://github.com/Goddard-Fortran-Ecosystem/pFlogger/.

9.2.9 HISTORY.rc

`HISTORY.rc` is the file that configures GCHP's output. It has the following format

```
EXPID:  OutputDir/GCHP
EXPDSC: GEOS-Chem_devel
CoresPerNode: 30
VERSION: 1

<DEFINE GRID LABELS>

<DEFINE ACTIVE COLLECTIONS>

<DEFINE COLLECTIONS>
```

EXPID This is the file prefix for all collections. `OutputDir/GCHP` means that collections will be to a direcotry named `OutputDir/`, and the file names will start with *GCHP*.

CoresPerNode The number of cores per node for your GCHP simulation.

EXPDSC Leave this as it is.

VERSION Leave this as it is.

The format and description of *<DEFINE GRID LABELS>*, *<DEFINE ACTIVE COLLECTIONS>*, and and *<DEFINE COLLECTIONS>* sections are given below.

Defining Grid Labels

You can specify custom grids for your output. For example, a regional 0.05°x0.05° grid covering North America. This way your collections are regridded online. There are two advantages to doing this:

1. It eliminates the need to regrid your simulation data in a post-processing step.
2. It saves disk space if you are interested in regional output.

Beware that outputting data on a different grid assumes the data is independent of horizontal cell size. The regridding routines are area-conserving and thus regridded values will only make sense for data that is area-independent. Examples of data units that are area-independent are mixing ratios (e.g. kg/kg or mol/mol) and emissions rates per area (e.g. kg/m2/s). Examples of data units that are NOT area-independent are kg/s and m2, or any other unit that implicitly is per grid cell area. This sort of unit is most common in the meteorology diagnostics, such as Met_AREAM2 and Met_AD. The values of these arrays will be incorrect in non-native grid output.

You can define as many grids as you want. A collection can define `grid_label` to select a custom grid. If a collection does not define `grid_label` the simulation's grid is assumed.

Below is the format for the `<DEFINE GRID LABELS>` section in `HISTORY.rc`.

```
GRID_LABELS:   MY_FIRST_GRID       # My custom grid for C96 output
               MY_SECOND_GRID      # My custom grid for global 0.5x0.625 output
               MY_THIRD_GRID       # My custom grid for regional 0.05x0.05 output
::
   MY_FIRST_GRID.GRID_TYPE:   Cubed-Sphere
   MY_FIRST_GRID.IM_WORLD:    96
   MY_FIRST_GRID.JM_WORLD:    576      # 576=6x96

   MY_SECOND_GRID.GRID_TYPE:  LatLon
   MY_SECOND_GRID.IM_WORLD:   360
   MY_SECOND_GRID.JM_WORLD:   181
```

(continues on next page)

(continued from previous page)

```
   MY_SECOND_GRID.POLE:       PC       # pole-centered
   MY_SECOND_GRID.DATELINE:   DC       # dateline-centered

   MY_THIRD_GRID.GRID_TYPE:   LatLon
   MY_THIRD_GRID.IM_WORLD:    80
   MY_THIRD_GRID.JM_WORLD:    40
   MY_THIRD_GRID.POLE:        XY
   MY_THIRD_GRID.DATELINE:    XY
   MY_THIRD_GRID.LON_RANGE:    0 80    # regional boundaries
   MY_THIRD_GRID.LAT_RANGE:  -30 10
```

SPEC NAMES

GRID_TYPE The type of grid. Valid options are `Cubed-Sphere` or `LatLon`.

IM_WORLD The number of grid boxes in the i-dimension. For a `LatLon` grid this is the number of longitude grid-boxes. For a `Cubed-Sphere` grid this is the cubed-sphere size (e.g., 48 for C48).

JM_WORLD The number of grid boxes in the j-dimension. For a `LatLon` grid this is the number of latitude grid-boxes. For a `Cubed-Sphere` grid this is six times the cubed-sphere size (e.g., 288 for C48).

POLE Required if the grid type is `LatLon`. `POLE` defines the latitude coordinates of the grid. For global lat-lon grids the valid options are `PC` (pole-centered) or `PE` (polar-edge). Here, "center" or "edge" refers to whether the grid has boxes that are centered on the poles, or whether the grid has boxes with edges at the poles. For regional grids `POLE` should be set to `XY` and the grid will have boxes with edges at the regional boundaries.

DATELINE Required if the grid type is `LatLon`. `DATELINE` defines the longitude coordinates of the grid. For global lat-lon grids the valid options are `DC` (dateline-centered), `DE` (dateline-edge), `GC` (grenwich-centered), or `GE` (grenwich-edge). If `DC` or `DE`, then the longitude coordinates will span (-180°, 180°). If `GC` or `GE`, then the longitude coordinates will span (0°, 360°). Similar to `POLE`, "center" or "edge" refer to whether the grid has boxes that are centered at -180° or 0°, or whether the grid has boxes with edges at -180° or 0°. For regional grids `DATELINE` should be set to *XY* and the grid will have boxes with edges at the regional boundaries.

LON_RANGE Required for regional `LatLon` grids. `LON_RANGE` defines the longitude bounds of the regional grid.

LAT_RANGE Required for regional `LatLon` grids. `LAT_RANGE` defines the latitude bounds of the regional grid.

Defining Active Collections

Collections are activated by defining them in the `COLLECTIONS` list. For instructions on defining collections, see *Defining Collections*.

Below is the format for the `<DEFINE ACTIVE COLLECTIONS>` section of `HISTORY.rc`.

```
COLLECTIONS:   'MyCollection1',
               'MyCollection2',
::
```

This example activates collections named "MyCollection1" and "MyCollection2".

Defining Collections

A collection is

```
MyCollection1.template:    '%y4%m2%d2_%h2%n2z.nc4',
MyCollection1.format:      'CFIO',
MyCollection1.frequency:   010000
MyCollection1.duration:    240000
MyCollection1.mode:        'time-averaged'
MyCollection1.fields:      'SpeciesConc_O3  ',  'GCHPchem',
                           'SpeciesConc_NO  ',  'GCHPchem',
                           'SpeciesConc_NO2 ',  'GCHPchem',
                           'Met_BXHEIGHT    ',  'GCHPchem',
                           'Met_AIRDEN      ',  'GCHPchem',
                           'Met_AD          ',  'GCHPchem',
::
<DEFINE MORE COLLECTIONS ...>
```

Output file configuration

template This is the file name suffix for the collection. The path to the collection's files is obtained by concatenating `EXPID` with the collection name and the value of `template`.

format Defines the file format of the collection. Valid values are `'CFIO'` for CF compliant NetCDF (recommended), or `'flat'` for GrADS style flat files.

duration Defines the frequency at which files are generated. The format is `HHMMSS`. For example, `1680000` means that a file is generated every 168 hours (7 days).

monthly *[optional]* Set to `1` for monthly output. One file per month is generated. If `mode` is `time-averaged`, the variables in the collection are 1-month time averages.

`duration` and `frequency` are not required if `monthly: 1`.

timeStampStart *[optional]* Only used if `mode` is `'time-averaged'`. If `.true.` the file is timestamped according to the start of the accumulation interval (which depends on `frequency`, `ref_date`, and `ref_time`). If `.false.` the file is timestamped according to the middle of the accumulation interval. If `timeStampStart` is not set then the default value is false.

Sampling configuration

mode Defines the sampling method. Valid values are `'time-averaged'` or `'instantaneous'`.

frequency Defines the time frequency of collection's data. Said another way, this defines the time separation (time step) of the time coordinate for the collection. The format is `HHMMSS`. For example, `010000` means that the collection's time coordinate will have a 1-hour time step. If `frequency` is less than `duration` multiple time steps are written to each file.

acc_interval *[optional]* Only valid if `mode` is `'time-averaged'`. This specifies the length of the time average. By default it is equal to `frequency`.

ref_date *[optional]* The reference date from which the frequency is based. The format is `YYYYMMDD`. For example, a frequency of `1680000` (7 days) with a reference date of *20210101* means that the time coordinate will be weeks since 2021-01-01. The default value is the simulation's start date.

ref_time *[optional]* The reference time from which the frequency is based. The format is `HHMMSS`. The default value is `000000`. See `ref_date`.

fields Defines the list of fields that this collection should use. The format (per-field) is `'FieldName', 'GridCompName',`. For example, `'SpeciesConc_O3', 'GCHPchem',` specifies that this collection should include the *SpeciesConc_O3* field from the *GCHPchem* gridded component.

Fields from multiple gridded components can be included in the same collection. However, a collection must not mix fields that are defined at the center of vertical levels and the edges of vertical levels (e.g., *Met_PMID* and *Met_PEDGE* cannot be included in the same collection).

Variables can be renamed in the output by adding `'your_custom_name',` at the end. For example, `'SpeciesConc_O3', 'GCHPchem', 'ozone_concentration',` would rename the SpeciesConc_O3 field to "ozone_concentration" in the output file.

Output grid configuration

grid_label *[optional]* Defines the grid that this collection should be output on. The lable must match on of the grid labels defined in *<DEFINE GRID LABELS>*. If `grid_label` isn't set then the collection uses the simulation's horizontal grid.

conservative *[optional]* Defines whether or not regridding to the output grid should use ESMF's first-order conservative method. Valid values are `0` or `1`. It is recommended you set this to `1` if you are using `grid_label`. The default value is `0`.

levels *[optional]* Defines the model levels that this collection should use (i.e., a subset of the simulation levels). The format is a space-separated list of values. The lowest layer is 1 and the highest layer is 72. For example, `1 2 5` would select the first, second, and fifth level of the simulation.

track_file *[optional]* Defines the path to a 1D track file along which the collection is sampled. See *Output Along a Track* for more info.

recycle_track *[optional]* Only valid if a `track_file` is defined. Specifies that the track file should be reused every day. If `.true.` the dates in the track file are automatically forced to the simulation's current date. The default value is false.

Other configuration

end_date *[optional]* A date at which the collection is deactivated (turned off). By default there is no end date.

end_time *[optional]* Time at which the collection is deactivated (turned off) on the `end_date`.

Example `HISTORY.rc` configuration

Below is an example `HISTORY.rc` that configures two output collection

1. 30-min instantaneous concentrations of O3, NO, NO2, and some meteorological parameters for the lowest 10 model levels on a 0.1°x0.1° covering the US. Each file contains one day of data.
2. 24-hour time averages of O3, NO, and NO2 concentrations, NO emissions, and some meteorological parameters. The horizontal grid is the simulation's grid. All vertical levels are use. Each file contains one week worth of data, and files are generated relative to 2017-01-01.

```
EXPID:  OutputDir/GCHP
EXPDSC: GEOS-Chem_devel
CoresPerNode: 6
VERSION: 1

GRID_LABELS: RegionalGrid_US
::
   RegionalGrid_US.GRID_TYPE: LatLon
   RegionalGrid_US.IM_WORLD:   640
   RegionalGrid_US.JM_WORLD:   290
   RegionalGrid_US.POLE:        XY
   RegionalGrid_US.DATELINE:    XY
   RegionalGrid_US.LON_RANGE: -127 -63
```

(continues on next page)

(continued from previous page)

```
   RegionalGrid_US.LAT_RANGE:   23  52

COLLECTIONS: 'Inst30minGases',
      'DailyAvgGasesAndNOEmissions',
::
Inst30minGases.template:    '%y4%m2%d2_%h2%n2z.nc4',
Inst30minGases.format:      'CFIO',
Inst30minGases.frequency:   003000
Inst30minGases.duration:    240000
Inst30minGases.mode:        'instantaneous'
Inst30minGases.grid_label:  RegionalGrid_US
Inst30minGases.levels:      1 2 3 4 5 6 7 8 9 10 11 12 13 14
Inst30minGases.fields:     'SpeciesConc_O3  ',  'GCHPchem',
                           'SpeciesConc_NO  ',  'GCHPchem',
                           'SpeciesConc_NO2 ',  'GCHPchem',
                           'Met_BXHEIGHT    ',  'GCHPchem',
                           'Met_AIRDEN      ',  'GCHPchem',
                           'Met_AD          ',  'GCHPchem',
                           'Met_PS1WET      ',  'GCHPchem',
::
DailyAvgGasesAndNOEmissions.template:     '%y4%m2%d2_%h2%n2z.nc4',
DailyAvgGasesAndNOEmissions.format:       'CFIO',
DailyAvgGasesAndNOEmissions.ref_date:     20170101
DailyAvgGasesAndNOEmissions.frequency:    240000
DailyAvgGasesAndNOEmissions.duration:    1680000
DailyAvgGasesAndNOEmissions.mode:         'time-averaged'
DailyAvgGasesAndNOEmissions.fields:       'SpeciesConc_O3  ',  'GCHPchem',
                                          'SpeciesConc_NO  ',  'GCHPchem',
                                          'SpeciesConc_NO2 ',  'GCHPchem',
                                          'EmisNO_Total    ',  'GCHPchem',
                                          'EmisNO_Aircraft ',  'GCHPchem',
                                          'EmisNO_Anthro   ',  'GCHPchem',
                                          'EmisNO_BioBurn  ',  'GCHPchem',
                                          'EmisNO_Lightning',  'GCHPchem',
                                          'EmisNO_Ship     ',  'GCHPchem',
                                          'EmisNO_Soil     ',  'GCHPchem',
                                          'EmisNO2_Anthro  ',  'GCHPchem',
                                          'EmisNO2_Ship    ',  'GCHPchem',
                                          'EmisO3_Ship     ',  'GCHPchem',
                                          'Met_BXHEIGHT    ',  'GCHPchem',
                                          'Met_AIRDEN      ',  'GCHPchem',
                                          'Met_AD          ',  'GCHPchem',
::
```

9.2.10 HEMCO_Diagn.rc

Like in GEOS-Chem Classic, the `HEMCO_Diagn.rc` file is used to map between HEMCO containers and output file diagnostic names. However, while all uncommented diagnostics listed in `HEMCO_Diagn.rc` are output as HEMCO diagnostics in GEOS-Chem Classic, only the subset also listed in `HISTORY.rc` are output in GCHP. See the HEMCO documentation for an overview of the file.

CHAPTER

TEN

CONFIGURE A RUN

As noted earlier, the many configuration files in GCHP can be overwhelming but you should be able to accomplish most if not all of what you wish to configure from one place in `setCommonRunSettings.sh`. Use this section to learn what to change in the run directory based on what you would like to do.

Table of contents

* *Prevent overwriting diagnostic files*

Note: If there is topic not covered on this page that you would like to see added please create an issue on the GCHP issues page with your request.

10.1 Compute resources

10.1.1 Set number of nodes and cores

To change the number of nodes and cores for your run you must update settings in two places: (1) `setCommonRunSettings.sh`, and (2) your run script. The `setCommonRunSettings.sh` file contains detailed instructions on how to set resource parameter options and what they mean. Look for the Compute Resources section in the script. Update your resource request in your run script to match the resources set in `setCommonRunSettings.sh`.

It is important to be smart about your resource allocation. To do this it is useful to understand how GCHP works with respect to distribution of nodes and cores across the grid. At least one unique core is assigned to each face on the cubed sphere, resulting in a constraint of at least six cores to run GCHP. The same number of cores must be assigned to each face, resulting in another constraint of total number of cores being a multiple of six. Communication between the cores occurs only during transport processes.

While any number of cores is valid as long as it is a multiple of six (although there is an upper limit per resolution), you will typically start to see negative effects due to excessive communication if a core is handling less than around one hundred grid cells or a cluster of grid cells that are not approximately square. You can determine how many grid cells are handled per core by analyzing your grid resolution and resource allocation. For example, if running at C24 with six cores each face is handled by one core (6 faces / 6 cores) and contains 576 cells (24x24). Each core therefore processes 576 cells. Since each core handles one face, each core communicates with four other cores (four surrounding faces). Maximizing squareness of grid cells per core is done automatically within `setCommonRunSettings.sh` if variable `AutoUpdate_NXNY` is set to `ON` in the "DOMAIN DECOMPOSITON" section of the file.

10.1.2 Change domain stack size

For runs at very high resolution or small number of processors you may run into a domains stack size error. This is caused by exceeding the domains stack size memory limit set at run-time. The error will be apparent from the message in your log file. If this occurs you can increase the domains stack size in file `input.nml`.

10.2 Basic run settings

10.2.1 Set cubed-sphere grid resolution

GCHP uses a cubed sphere grid rather than the traditional lat-lon grid used in GEOS-Chem Classic. While regular lat-lon grids are typically designated as Lat Lon (e.g. 45), cubed sphere grids are designated by the side-length of the cube. In GCHP we specify this as CX (e.g. C24 or C180). The simple rule of thumb for determining the roughly equivalent lat-lon resolution for a given cubed sphere resolution is to divide the side length by 90. Using this rule you can quickly match C24 with about 4x5, C90 with 1 degree, C360 with quarter degree, and so on.

To change your grid resolution in the run directory edit `CS_RES` in the "GRID RESOLUTION" section of `setCommonRunSettings.sh`. The paramter should be an integer value of the cube side length you wish to use. To use a uniform global grid resolution make sure `STRETCH_GRID` is set to `OFF` in the "STRETCHED GRID" section of the file. To use a stretched grid rather than a globally uniform grid see the section on this page for setting stretched grid parameters.

10.2.2 Set stretched grid parameters

GCHP has the capability to run with a stretched grid, meaning one portion of the globe is stretched to fine resolution. Set stretched grid parameter in `setCommonRunSettings.sh` section "STRETCHED GRID". See instructions in that section of the file. For more detailed information see the stretched grid section of the Supplemental Guides section of the GCHP ReadTheDocs.

10.2.3 Turn on/off model components

You can toggle most primary GEOS-Chem components that are set in `geoschem_config.yml` from the "GEOS-CHEM COMPONENTS" section of `setCommonRunSettings.sh`. The settings in that file will update `geoschem_config.yml` automatically so be sure to check that the settings there are as you intend. For emissions you should directly edit `HEMCO_Config.rc`.

10.2.4 Change model timesteps

Model timesteps, including chemistry, dynamic, and RRTMG, are configured within the "TIMESTEPS" section of `setCommonRunSettings.sh`. By default, the RRTMG timestep is set to 3 hours. All other GCHP timesteps are automatically set based on grid resolution. Chemistry and dynamic timesteps are 20 and 10 minutes respectively for grid resolutions coarser than C180, and 10 and 5 minutes for C180 and higher. Meteorology read frequency for PS2, SPHU2, and TMPU2 are automatically updated in `ExtData.rc` accordingly. To change the default timesteps settings edit the "TIMESTEPS" section of `setCommonRunSettings.sh`.

10.2.5 Set simulation start date and duration

Unlike GEOS-Chem Classic, GCHP uses a start date and run duration rather than start and end dates. Set simulation start date in `cap_restart` using string format `YYYYMMDD HHmmSS`. Set simulation duration in section "SIMULATION DURATION" in `setCommonRunSettings.sh` using the same format as start date. For example, a 1-year run starting 15 January 2019 would have `20190115 000000` in `cap_restart` and `00010000 000000` in `setCommonRunSettings.sh`.

Under the hood `cap_restart` is used directly by the MAPL software in GCHP, and `setCommonRunSettings.sh` auto-updates the run duration in GCHP config file `CAP.rc`. Please be aware that MAPL overwrites `cap_restart` at the end of the simulation to contain the new start date (end of last run) so be sure to check it every time you run GCHP.

If you poke around the GCHP configuration files you may notice that file `CAP.rc` contains entries for `BEG_DATE` and `END_DATE`. You can ignore these fields for most cases. `BEG_DATE` is not used for start date if `cap_restart` is present. However, it must be prior to your start date for use in GEOS-Chem's "ELAPSED_TIME" variable. We set it to year 1960 to be safe. `BEG_DATE` can also be ignored as long as it is the same as or later than your start date plus run duration. For safety we set it to year 2200. The only time you would need to adjust these settings is for simulations way in the past or way into the future.

10.3 Inputs

10.3.1 Change restart file

All GCHP run directories come with symbolic links to initial restart files for commonly used cubed sphere resolutions. These are located in the `Restarts` directory in the run directory. All initial restart files contain start date and grid resolution in the filename using the start date in `cap_restart`. Prior to running GCHP, either you or your run script will execute `setRestartLink.sh` to create a symbolic link `gchp_restart.nc4` to point to the appropriate restart file given configured start date and grid resolution. `gchp_restart.nc4` will always be used as the restart file for all runs since it is specified as the restart file in `GCHP.rc`.

If you want to change the restart file then you should put the restart file you want to use in the `Restarts` directory using the expected filename format with the start date you configure in `cap_restart` and the grid resolution you configure in `setCommonRunSettings.sh`. The expected format is `GEOSChem.Restarts.YYYYMMDD_HHmmz.cN.nc4`. Running `setRestartLink.sh` will update `gchp_restart.nc4` to use it.

If you do not want to rename your restart file then you can create a symbolic link in the `Restarts` folder that points to it.

Please note that unlike GC-Classic, GCHP does not use a separate HEMCO restart file. All HEMCO restart variables are included in the main GCHP restart.

10.3.2 Enable restart file to have missing species

Most simulations by default do not allow missing species in the restart file. The model will exit with an error if species are not found. However, there is a switch in `setCommonRunSetting.sh` to disable this behavior. This toggle is located in the section on infrequently changed settings under the header `REQUIRE ALL SPECIES IN INITIAL RESTART FILE`. Setting the switch to `NO` will use background values set in `species_database.yml` as initial values for species that are missing.

10.3.3 Turn on/off emissions inventories

Because file I/O impacts GCHP performance it is a good idea to turn off file read of emissions that you do not need. You can turn individual emissions inventories on or off the same way you would in GEOS-Chem Classic, by setting the inventories to true or false at the top of configuration file `HEMCO_Config.rc`. All emissions that are turned off in this way will be ignored when GCHP uses `ExtData.rc` to read files, thereby speeding up the model.

For emissions that do not have an on/off toggle at the top of the file, you can prevent GCHP from reading them by commenting them out in `HEMCO_Config.rc`. No updates to `ExtData.rc` would be necessary. If you alternatively comment out the emissions in `ExtData.rc` but not `HEMCO_Config.rc` then GCHP will fail with an error when looking for the file information.

Another option to skip file read for certain files is to replace the file path in `ExtData.rc` with `/dev/null`. However, if you want to turn these inputs back on at a later time you should preserve the original path by commenting out the original line.

10.3.4 Change input meteorology

Input meteorology source and grid resolution are set in config file `ExtData.rc` during run directory creation. You will be prompted to choose between MERRA2 and GEOS-FP, and grid resolution is automatically set to the native grid lat-lon resolution. If you would like to change the meteorology inputs, for example using a different grid resolution, then you would need to change the met-field entries in run directory file `ExtData.rc` after creating a run directory. Simply open the file, search for the meteorology section, and edit file paths as needed. Please note that while MAPL will automatically regrid met-fields to the run resolution you specify in `setCommonRunSettings.sh`, you will achieve best performance using native resolution inputs.

10.3.5 Add new emissions files

There are two steps for adding new emissions inventories to GCHP. They are (1) add the inventory information to `HEMCO_Config.rc`, and (2) add the inventory information to `ExtData.rc`.

To add inventory information to `HEMCO_Config.rc`, follow the same rules as you would for adding a new emission inventory to GEOS-Chem Classic. Note that not all information in `HEMCO_Config.rc` is used by GCHP. This is because HEMCO is only used by GCHP to handle emissions after they are read, e.g. scaling and applying hierarchy. All functions related to HEMCO file read are skipped. This means that you could put garbage for the file path and units in `HEMCO_Config.rc` without running into problems with GCHP, as long as the syntax is what HEMCO expects. However, we recommend that you fill in `HEMCO_Config.rc` in the same way you would for GEOS-Chem Classic for consistency and also to avoid potential format check errors.

To add inventory information to `ExtData.rc` follow the guidelines listed at the top of the file and use existing inventories as examples. Make sure that you stay consistent with the information you put into `HEMCO_Config.rc`. You can ignore all entries in `HEMCO_Config.rc` that are copies of another entry (i.e. mostly filled with dashes). Putting these in `ExtData.rc` would result in reading the same variable in the same file twice.

A few common errors encountered when adding new input emissions files to GCHP are:

1. Your input file contains integer values. Beware that the MAPL I/O component in GCHP does not read or write integers. If your data contains integers then you should reprocess the file to contain floating point values instead.
2. Your data latitude and longitude dimensions are in the wrong order. Lat must always come before lon in your inputs arrays, a requirement true for both GCHP and GEOS-Chem Classic.
3. Your 3D input data are mapped to the wrong levels in GEOS-Chem (silent error). If you read in 3D data and assign the resulting import to a GEOS-Chem state variable such as `State_Chm` or `State_Met`, then you must flip the vertical axis during the assignment. See files `Includes_Before_Run.H` and setting `State_Chm%Species` in `Chem_GridCompMod.F90` for examples.
4. You have a typo in either `HEMCO_Config.rc` or `ExtData.rc`. Errors in `HEMCO_Config.rc` typically result in the model crashing right away. Errors in `ExtData.rc` typically result in a problem later on during ExtData read. Always try a short run with all debug prints enabled when first implementing new emissions. See the debugging section of the user manual for more information. Another useful strategy is to find config file entries for similar input files and compare them against the entry for your new file. Directly comparing the file metadata may also lead to insights into the problem.

10.4 Outputs

10.4.1 Output diagnostics data on a lat-lon grid

See documentation in the `HISTORY.rc` config file for instructions on how to output diagnostic collection on lat-lon grids, as well as the configuration files section at the top of this page for more information on that file. If outputting on a lat-lon grid you may also output regional data instead of global. Make sure that whatever grid you choose is listed under `GRID_LABELS` and is not commented out in `HISTORY.rc`.

10.4.2 Output restart files at regular frequency

The MAPL component in GCHP has the option to output restart files (also called checkpoint files) prior to run end. These periodic restart files are output to the main level of the run directory with filename `gcchem_internal_checkpoint.YYYYMMDD_HHssz.nc4`.

Outputting restart files beyond the end of the run is a good idea if you plan on doing a long simulation and you are not splitting your run into multiple jobs. If the run crashes unexpectedly then you can restart mid-run rather than start over from the beginning. Update settings for checkpoint restart outputs in `setCommonRunSettings.sh` section "MID-RUN CHECKPOINT FILES". Instructions for configuring restart frequency are included in the file.

10.4.3 Turn on/off diagnostics

To turn diagnostic collections on or off, comment ("#") collection names in the "COLLECTIONS" list at the top of file `HISTORY.rc`. Collections cannot be turned on/off from `setCommonRunSettings.sh`.

10.4.4 Set diagnostic frequency, duration, and mode

All diagnostic collections that come with the run directory have frequency and duration auto-set within `setCommonRunSettings.sh`. The file contains a list of time-averaged collections and instantaneous collections, and allows setting a frequency and duration to apply to all collections listed for each. Time-avraged collections also have a monthly mean option (see separate section on this page about monthly mean). To avoid auto-update of a certain collection, remove it from the list in `setCommonRunSettings.sh`, or set "AutUpdate_Diagnostics" to `OFF`. See section "DIAGNOSTICS" within `setCommonRunSettings.sh` for examples.

10.4.5 Add a new diagnostics collection

Adding a new diagnostics collection in GCHP is the same as for GEOS-Chem Classic netcdf diagnostics. You must add your collection to the collection list in `HISTORY.rc` and then define it further down in the file. Any 2D or 3D arrays that are stored within GEOS-Chem objects `State_Met`, `State_Chm`, or `State_Diag`, may be included as fields in a collection. `State_Met` variables must be preceded by "Met_", `State_Chm` variables must be preceded by "Chem_", and `State_Diag` variables should not have a prefix. Collections may have a combination of 2D and 3D variables, but all 3D variables must have the same number of levels. See the `HISTORY.rc` file for examples.

10.4.6 Generate monthly mean diagnostics

You can toggle monthly mean diagnostics on/off from within `setCommonRunSettings.sh` in the "DIAGNOSTICS" section if you also set auto-update of diagnostics it that file to on. All time-averaged diagnostic collections will then automatically be configured to compute monthly mean. Alternatively, you can edit `HISTORY.rc` directly and set the "monthly" field to value 1 for each collection you wish to output monthly diagnostics for.

10.4.7 Prevent overwriting diagnostic files

By default all GCHP run directories are configured to allow overwriting diagnostics files present in `OutputDir` over the course a simulation. You may disable this feature by setting `Allow_Overwrite=.false.` at the top of configuration file `HISTORY.rc`.

CHAPTER

ELEVEN

OUTPUT FILES

A successful GCHP run produces three categories of output files: diagnostics, restarts (also called checkpoints), and logs. Diagnostic and restart files are always in netCDF4 format, and logs are always ascii viewable with any text editor. Diagnostic files are output to the `OutputDir` directory in the run directory. The end-of-run restart file is output to the `Restarts` directory. All other output files, including periodic checkpoints if enabled, are saved to the main level of the run directory.

Note: It is important to be aware that GCHP 3D data files in this version of GCHP have two different vertical dimension conventions. Restart files and Emissions diagnostic files are defined with top-of-atmospheric level equal to 1. All other data files, meaning all diagnostic files that are not Emissions collections, are defined with surface level equal to 1. This means files may be vertically flipped relative to each other. This should be taken into account when doing data visualization and analysis using these files.

11.1 File descriptions

Below is a summary of all GCHP output files that you may encounter depending on your run directory configuration.

`gchp.YYYYMMSS_HHmmSSz.log`
: Standard output log file of GCHP, including both GEOS-Chem and HEMCO. The date in the filename is the start date of the simulation. Using this file is technically optional since it appears only in the run script. However, the advantage of sending GCHP standard output to this file is that the logs of consecutive runs will not be over-written due to the date in the filename. Note that the file contains HEMCO log information as well as GEOS-Chem. Unlike in GEOS-Chem Classic there is no `HEMCO.log` in GCHP.

`batch` `job file`**, e.g.** `slurm-jobid.out`
: If you use a job scheduler to submit GCHP as a batch job then you will have a job log file. This file will contain output from your job script unless sent to a different file. If your run crashes then the MPI error messages and error traceback will also appear in this file.

`allPES.log`
: GCHP logging output based on configuration in `logging.yml`. Treat this file as a debugging tool to help diagnose problems in MAPL, particularly the ExtData component of the model which handles input reading and regridding.

`logfile.000000.out`
: Log file for advection. It includes information such as the domain stack size, stretched grid factors, and FV3 parameters used in the run.

`cap_restart`
: This file is both input and output. As an input file it contains the simulation start date. After a successful run the

content of the file is updated to the simulation end date. As an output file it is therefore the input file for the next run if running GCHP simulations consecutively in time.

Restarts/GEOSChem.Restart.YYYYMMDD_HHmmz.cN.nc4
GCHP restart file output at the end of the run. This file is actually the GCHP end-of-run checkpoint file that is moved and renamed as part of the run script. Unless including the code to do that in your run script you will instead get `gcchem_internal_checkpoint` in the main run directory. Moving and renaming is a better option because (1) it includes the datetime to prevent overwriting upon consecutive runs, (2) it enables using the `gchp_restart.nc4` symbolic link in the main run directory to automatically point to the correct restart file based on start date and grid resolution, and (3) it minimizes clutter in the run directory. Please note that the vertical level dimension in all GCHP restart files is positive down, meaning level 1 is top-of-atmosphere.

gcchem_internal_checkpoint.YYYYMMDD_HHmmz.nc4
Optional restart files output mid-run. In order to generate these you must configure the run directory to output with a specific frequency that is less than the duration of your run. Note that unlike the end-of-run restart file, these files are not copied to `Restarts` in your run script and are not renamed.

OutputDir/GEOSChem.HistoryCollectionName.YYYYMMDD_HHmmz.nc4
GCHP diagnostic data files. Each file contains the collection name configured in `HISTORY.rc` and the datetime of the first data in the file. For time-averaged data files the datetime is the start of the averaging period. Please note that the vertical level dimension in GCHP diagnostics files is collection-dependent. Data are positive down, meaning level 1 is top-of-atmosphere, for the Emissions collection. All other collections are positive up, meaning level 1 is surface.

HistoryCollectionName.rcx
Summary of settings in `HISTORY.rc` per collection.

EGRESS
This file is empty and can be ignored. It is an artifact of the MAPL software used in GCHP.

warnings_and_errors.log
This file is empty and can be ignored. It is an artifact of configuration in `logging.yml`.

11.2 Memory

Memory statistics are printed to the GCHP log each model timestep. As discussed in the run directory configuration section of this user guide, this includes percentage of memory committed, percentage of memory used, total used memory (MB), and total swap memory (MB) by default.

To inspect the memory usage of GCHP you can grep the output log file for string `Date:` and `Mem/Swap`. For example, `grep "Date:|Mem/Swap" gchp.log`. The end of the line containing date and time shows memory committed and used. For example, `42.8% :  40.4% Mem Comm:Used` indicates 42.8% of memory available is committed and 40.4% of memory is actually used. The total memory used is in the next line, for example `Mem/Swap Used (MB) at MAPL_Cap:TimeLoop= 1.104E+05 0.000E+00`. The first value is the total memory used in MB, and the second line is swap (virtual) memory used. In this example GCHP is using around 110 gigabytes of memory with zero swap.

These memory statistics are useful for assessing how much memory GCHP is using and whether the memory usage grows over time. If the memory usage goes up throughout a run then it is an indication of a memory leak in the model. The memory debugging option is useful for isolating the memory leak by determining if there if it is in GEOS-Chem or advection.

11.3 Timing

Timing of GCHP components is done using MAPL timers. A summary of all timing is printed to the GCHP log at the end of a run. Configuring timers from the run directory is not currently possible but will be an option in a future version. Until then a complete summary of timing will always be printed to the end of the log for a successful GCHP run. You can use this information to help diagnose timing issues in the model, such as extra slow file read due to system problems.

The timing output written by MAPL is somewhat cryptic but you can use this guide to decipher it. Timing is broken in up into several sections.

1. `GCHPctmEnv`, the environment component that facilitates exchange between GEOS-Chem and FV3 advection
2. `GCHPchem`, the GEOS-Chem component containing chemistry, mixing, convection, emissions and deposition
3. `DYNAMICS`, the FV3 advection component
4. `GCHP`, the parent component of GCHPctmEnv, GCHPchem, and DYNAMICS, and sibling component to HIST and EXTDATA
5. `HIST`, the MAPL History component for writing diagnostics
6. `EXTDATA`, the MAPL ExtData component for processing inputs, including reading and regridding
7. Total model and MPI communicator run times broken into user, system, and total times
8. Full summary of all major model components, including core routines SetService, Initialize, Run, and Finalize
9. Model throughput in units of days per day

Each of the six gridded component sections contains two sub-sections. The first subsection shows timing statistics for core gridded component processes and their child functions. These statistics include number of execution cycles as well as inclusive and exclusive total time and percent time. `Inclusive` refers to the time spent in that function including called child functions. `Exclusive` refers to the time spent in that function excluding called child functions.

The second subsection shows from left to right minimum, mean, and maximum processor times for the gridded component and its MAPL timers. If you are interested in timing for a specific part of GEOS-Chem then use the timers in this section for `GCHPchem`, specifically the ones that start with prefix `GC_`. For chemistry you should look at timer `GC_CHEM` which includes the calls to compute overhead ozone, set H2O, and calling the chemistry driver routine.

Beware that the timers can be difficult to interpret because the component times do not always add up to the total run time. This is likely due to load imbalance where processors wait (timed in MAPL) while other processors complete (timed in other processes). You can get a sense of how large the wait time is by comparing the `Exclusive` time to the `Inclusive` time. If the former is smaller than the latter then the bulk of time is spent in a sub-process and the `Exclusive` time may be at least partially due to wait time.

If you are interested in changing the definitions of GCHP timers, or adding a new one, you will need to edit the source code. Toggling `GC_` timers on and off are mostly in file `geos-chem/Interfaces/GCHP/gchp_chunk_mod.F90`, but also in `geos-chem/Interfaces/GCHP/Chem_GridCompMod.F90`, using MAPL subroutines `MAPL_TimerOn` and `MAPL_TimerOff`. When in doubt about what a timer is measuring it is best to check the source code to see what calls it is wrapping.

CHAPTER

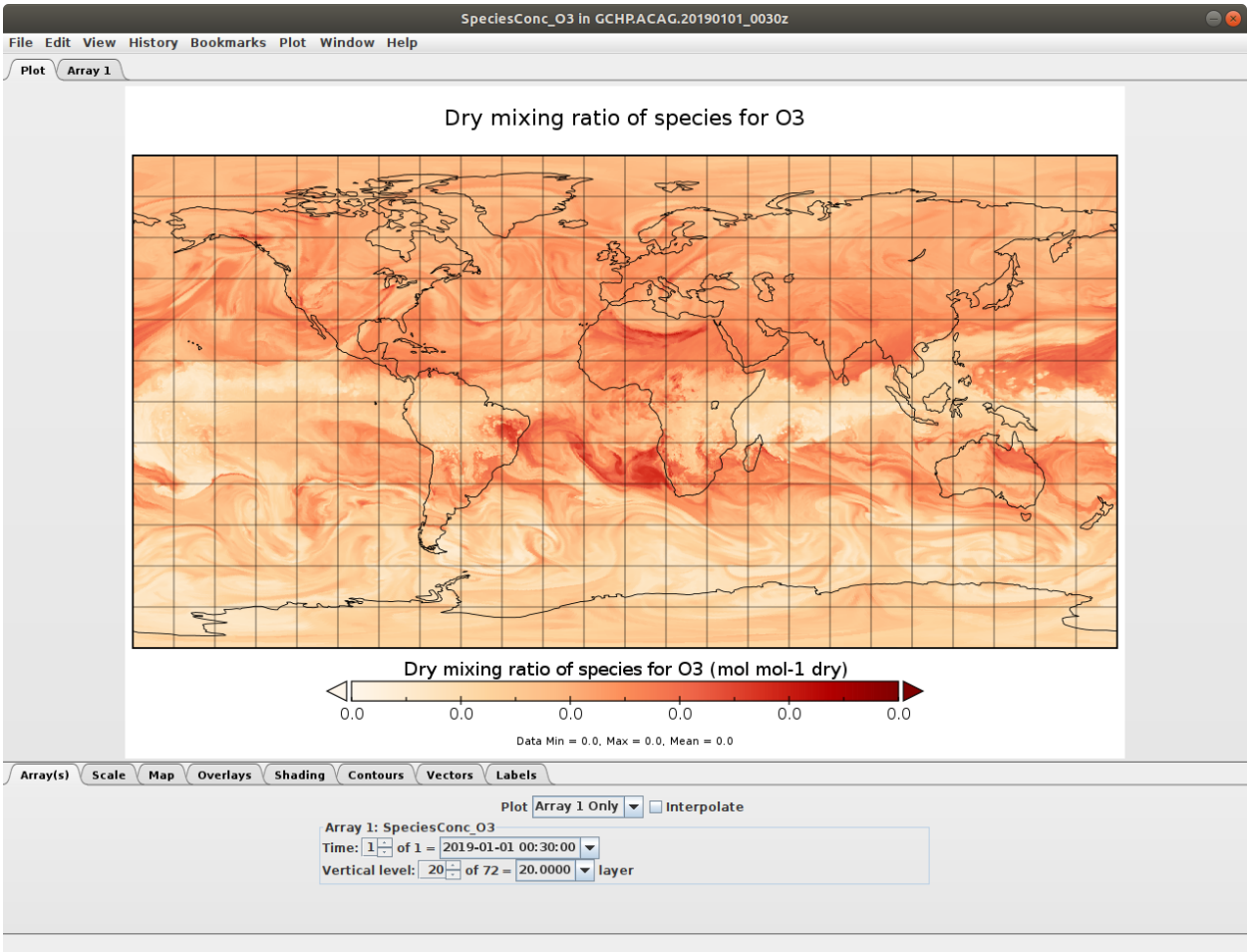
TWELVE

PLOT OUTPUT DATA

With the exception of the restart file, all GCHP output netCDF files may be viewed with Panoply software freely available from NASA GISS. In addition, python works very well with all GCHP output.

12.1 Panoply

Panoply is useful for quick and easy viewing of GCHP output. Panoply is a grahpical program for plotting geo-referenced data like GCHP's output. It is an intuitive program and it is easy to set up.



You can read more about Panoply, including how to install it, here.

Some suggestions

- If you can mount your cluster's filesystem as a Network File System (NFS) on your local machine, you can install Panoply on your local machine and view your GCHP data through the NFS.
- If your cluster supports a graphical interface, you could install Panoply (administrative priviledges not necessary, provided Java is installed) yourself.
- Alternatively, you could install Panoply on your local machine and use **`scp`** or similar to transfer files back and forth when you want to view them.

Note: To get rid of the missing value bands along face edges, **uncheck 'Interpolate'** (turn interpolation off) in the *Array(s)* tab.

12.2 Python

To make a basic plot of GCHP data using Python you will need the following libraries:

- cartopy >= 0.19 (0.18 won't work – see cartopy#1622)
- xarray
- netcdf4

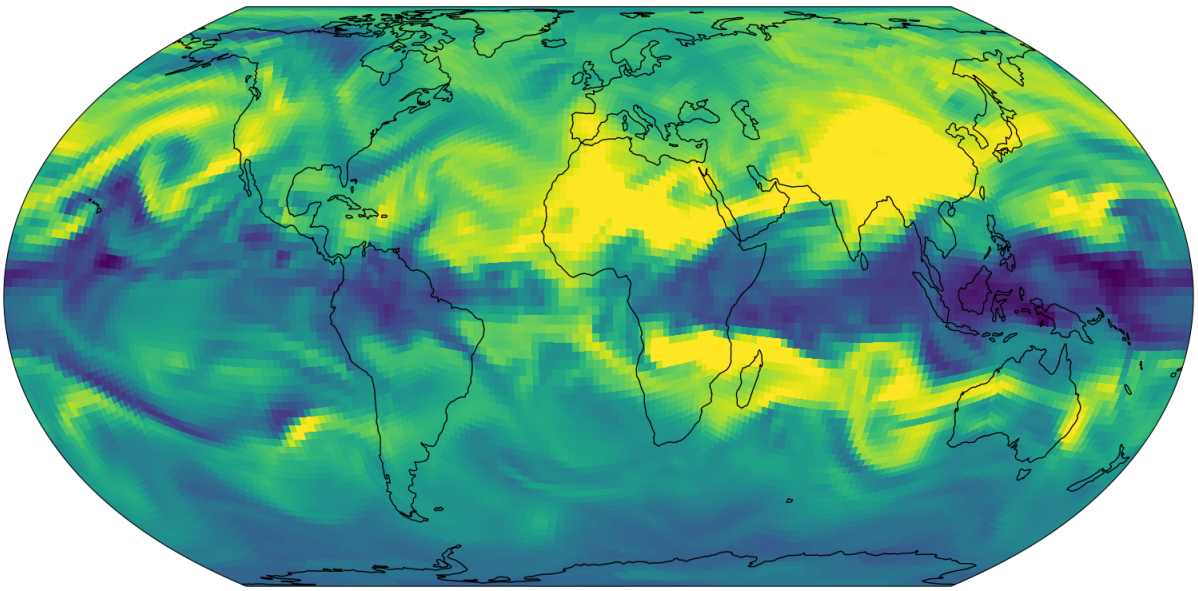
If you use conda you can install these packages like so

```
$ conda activate your-environment-name
$ conda install cartopy>=0.19 xarray netcdf4 -c conda-forge
```

Here is a basic example of plotting cubed-sphere data:

- Sample data: `GCHP.SpeciesConc.20210508_0000z.nc4`

```
import matplotlib.pyplot as plt
import cartopy.crs as ccrs  # cartopy must be >=0.19
import xarray as xr

ds = xr.open_dataset('GCHP.SpeciesConc.20210508_0000z.nc4')  # see note below for␣
↪download instructions

plt.figure()
ax = plt.axes(projection=ccrs.EqualEarth())
ax.coastlines()
ax.set_global()

norm = plt.Normalize(1e-8, 7e-8)

for face in range(6):
    x = ds.corner_lons.isel(nf=face)
    y = ds.corner_lats.isel(nf=face)
    v = ds.SpeciesConc_O3.isel(time=0, lev=23, nf=face)
    ax.pcolormesh(x, y, v, norm=norm, transform=ccrs.PlateCarree())

plt.show()
```



Note: The grid-box corners should be used with `pcolormesh()` because the grid-boxes are not regular (it's a curvilinear grid). This is why we use `corner_lats` and `corner_lons` in the example above.

You may also use the GCPy python toolkit to work with GCHP files. For more information see https://github.com/geoschem/gcpy/.

CHAPTER

THIRTEEN

DEBUGGING

This page provides strategies for investigating errors encountered while using GCHP.

13.1 Configure errors

The most basic configuration problem occurs if you forget to run `git submodule update --init --recursive` after cloning the GCHP repository. Check that you did this correctly. Other configuration problems usually have to do with libraries. Check that you have libraries loaded and that they meet the requirements for GCHP. Also check the logs printed to the build directory, in particular `CMakeCache.txt`. That file lists the directories of the libraries that are used. Check that these paths are what you intend to use. Sometimes on compute clusters there can be multiple instances of the same library loaded, such as when using a spack-built library when the cluster already has a different version of the same library. Check the library paths carefully to look for inconsistencies.

13.2 Build-time errors

Usually build-time errors are self-explanatory, with an error message indicating the file, line number, and reason for the error. Sometimes you need to do some digging in the build log to find where the error is. Searching for string " error " (note the space before and after) usually hones in on the problem fast. Read the error message carefully and then find the file and line number specified. If it is not clear what the error is even from the error message then you can try doing a string search on the GCHP GitHub issues page, or on the web in general. If the error is occuring with an out-of-the-box GCHP version then the issue is likely a library. Check that your libraries meet the requirements of GCHP as specified on ReadTheDocs. Also check your ESMF version and make sure you build ESMF using the same libraries with which you are building GCHP.

13.3 Run-time errors

The first step in debugging run-time errors is always to look at the logs. First check the `gchp.*.log` (* is the start time of the run) to see how far the run got. It is possible the error was trapped by HEMCO or GEOS-Chem in which case there will likey be error messages explaining the problem. Also check the standard error log. If running on a job scheduler this would be a separate file from the main GCHP log file. The error should include a traceback of the error, meaning filenames and line numbers where the error occurred, moving up the call stack from deepest to highest. Go to the first file listed and find the line number. Also read the error message in the traceback. Try to determine if the error is in GEOS-Chem, HEMCO, MAPL, or somewhere else. If the error is in MAPL then you should check output file `allPEs.log`. This log provides basic information on the MAPL run, which includes general GCHP infrastructure setup as well as model I/O.

13.3.1 Recompile with debug flags

If you are running into a segmentation fault or the model appears to hang within GEOS-Chem then you should try building with GEOS-Chem debug flags turned on. Recompile using debug flags by setting `-DCMAKE_BUILD_TYPE=Debug` during the configure step. See the section of the user guide on compiling GCHP for more guidance on how to do this. Once you rebuild and run there will be more information in the logs if the problem is an out-of-bounds error or floating point exception.

13.3.2 Enable maximum print output for GEOS-Chem and HEMCO

To more information about the execution within GEOS-Chem and HEMCO you can enable additional prints to the main GCHP log within `geoschem_config.yml` and `HEMCO_Config.rc`.

1. Activate GEOS-Chem verbose output by editing `geoschem_config.yml` as shown below. This will tell GEOS-Chem to send extra printout to the `gchp.YYYYMMDD_hhmmz.log` file.

```
#============================================================================
# Simulation settings
#============================================================================
simulation:
  # ... etc not shown ...
  verbose:
    activate: false   <=== Change this to true
    on_cores: root       # Allowed values: root all
```

2. Activate HEMCO verbose output by editing `HEMCO_Config.rc` as shown below. This will tell HEMCO to send extra printout to the `gchp.YYYYMMDD_hhmmz.log` file.

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

# ... etc not shown ...
Verbose:                    false   <=== Change this to true
```

13.3.3 Enable ESMF error log output

If the error is in MAPL then check if the call where the error occurs contains "ESMF". If the error is occuring in a call to ESMF then you should enable ESMF error log files in GCHP. Look for file `ESMF.rc` in your run directory. Open it and set the `logKindFlag` parameter to `ESMF_LOGKIND_MULTI_ON_ERROR` and run again. You should then get ESMF error log files upon rerun. There will be one log file per processor. More often than not the ESMF error message will appear in every file.

13.3.4 Enable maximum print output for MAPL

If you see `ExtData` in the error traceback then the problem has to do with input files. It is common to run into errors when adding new input files because of strict rules for import files within MAPL. If there is not enough information in `allPEs.log` to determine what the input file problem is then you should enable additional MAPL prints and rerun. This is mostly recommended for input file issues because MAPL ExtData is where most of the debug logging statements are currently implemented. However, problems elsewhere in MAPL might have useful debugging error messages as well. You can also go into the code and add your own by searching for examples with string `lgr%debug`. Contact the GEOS-Chem Support Team if you need help deciphering the resulting log output.

1. Activate the `CAP.EXTDATA` and `MAPL` debug loggers by editing the `logging.yml` configuration file as shown below. This will send all MAPL debug-level logging prints to the `allPEs.log` file.

```
loggers:

   # ... etc not shown ...

   MAPL:
       handlers: [mpi_shared]
       level: WARNING     <=== Change this to DEBUG
       root_level: INFO   <=== Change this to DEBUG

   CAP.EXTDATA:
       handlers: [mpi_shared]
       level: WARNING     <=== Change this to DEBUG
       root_level: INFO   <=== Change this to DEBUG
```

13.3.5 Read the code

If log error messages are not helpful in determining the problem then you may be able to solve it by reading the code. Follow the traceback to find the file and line number where the code crashed. You can find the location of files in GCHP by using the unix find command from the top-level source code directory, e.g. `find . -name aerosol_mod.F90` Once you find the file and the line where the model fails, read the code above it to try to get a sense of the context of where it crashed. This will give clues as to why it had a problem and may give you ideas of what to do to try to fix it. You can also add your own debug code, recompile, and run.

13.3.6 Inspecting memory

Memory statistics are printed to the GCHP log each model timestep by default. This includes percentage of memory committed, percentage of memory used, total used memory (MB), and total swap memory (MB). This information is always printed and is not configurable from the run directory. However, additional memory prints may be enabled by changing the value set for variable `MEMORY_DEBUG_LEVEL` in run directory file `GCHP.rc`. Setting this to a value greater than zero will print out total used memory and swap memory before and after run methods for gridded components GCHPctmEnv, FV3 advection, and GEOS-Chem. Within GEOS-Chem, total and swap memory will also be printed before and after subroutines to run GEOS-Chem, perform chemistry, and apply emissions. For more information about inspecting memory see the output files section of this user guide.

13.3.7 Inspecting timing

Model timing information is printed out at the end of each GCHP run. Check the end of the GCHP log for a breakdown of component timing.

CHAPTER

FOURTEEN

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.

14.1 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.

14.2 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 dependencis.

Depending on your system's setup, there are a few different ways in which you can activate these software pacakges in your computational environment, as shown below.

14.2.1 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).

14.2.2 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
```

14.2.3 3. Check if libaries 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.

14.2.4 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*.

CHAPTER

FIFTEEN

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

15.1 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*

15.2 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 complilers
```

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}
```

15.3 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
```

15.4 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>
```

15.5 Build GEOS-Chem dependencies and useful tools

Once the compiiler 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
```

15.6 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
```

15.7 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!

CHAPTER

SIXTEEN

SET UP AWS PARALLELCLUSTER

Important: AWS ParallelCluster and FSx for Lustre costs hundreds or thousands of dollars per month to use. See FSx for Lustre Pricing and EC2 Pricing for details.

AWS ParallelCluster is a service that lets you create your own HPC cluster. Using GCHP on AWS ParallelCluster is similar to using GCHP on any other HPC. We offer up-to-date Amazon Machine Images (AMIs) with GCHP's dependencies built and GCHP compiled through AMI list. These images contain pre-built GCHP source code and the tools for creating a GCHP run directory. This page has instructions on using the AMIs to create your own ParallelCluster. You can also choose to set up AWS ParallelCluster for running GCHP simulations yourself, and the other GCHP documentation like *Build GCHP's dependencies*, *Download the model*, *Compile*, *Download Input Data*, and *Run the model* is appropriate for using GCHP on AWS ParallelCluster.

The workflow for getting started with GCHP simulations using AWS ParallelCluster based on our public AMIs is

1. Create an FSx for Lustre file system for input data (*described on this page*)
2. Configure AWS CLI (*described on this page*)
3. Configure AWS ParallelCluster (*described on this page*)
4. Create AWS ParallelCluster with GCHP public AMIs (*described on this page*)
5. Follow the normal GCHP User Guide
    a. *Create a Run Directory*
    b. *Download Input Data*
6. Running GCHP on ParallelCluster (*described on this page*)

These instructions were written using AWS ParallelCluster 3.7.0.

16.1 1. Create an FSx for Lustre file system

Start by creating an FSx for Lustre file system. This is persistent storage that will be mounted to your AWS ParallelCluster cluster. This file system will be used for storing GEOS-Chem input data and for housing your GEOS-Chem run directories.

Refer to the official FSx for Lustre Instructions for instructions on creating the file system. Only Step 1, *Create your Amazon FSx for Lustre file system*, is necessary. Step 2, *Install the Lustre client*, and subsequent steps have instructions for mounting your file system to EC2 instances, but AWS ParallelCluster automates this for us.

In subsequent steps you will need the following information about your FSx for Lustre file system:

- its ID (`fs-XXXXXXXXXXXXXXXXX`)

- its subnet (`subnet-YYYYYYYYYYYYYYYYY`)
- its security group that has the inbound network rules (`sg-ZZZZZZZZZZZZZZZZZ`).

Once you have created the file system, proceed with *2. AWS CLI Installation and First-Time Setup*.

16.2 2. AWS CLI Installation and First-Time Setup

Next you need to make sure you have the AWS CLI installed and configured. The AWS CLI is a terminal command, `aws`, for working with AWS services. If you have already installed and configured the AWS CLI previously, continue to *3. Create your AWS ParallelCluster*.

Install the `aws` command: Official AWS CLI Install Instructions. Once you have installed the `aws` command, you need to configure it with the credentials for your AWS account:

```
$ aws configure
```

For instructions on `aws configure`, refer to the Official AWS Instructions or this YouTube tutorial.

16.3 3. Create your AWS ParallelCluster

Note: You should also refer to the offical AWS documentation on Configuring AWS ParallelCluster. Those instructions will have the latest information on using AWS ParallelCluster. The instructions on this page are meant to supplement the official instructions, and point out the important parts of the configuration for use with GCHP.

Next, install AWS ParallelCluster with `pip`. This requires Python 3.

```
$ pip install aws-parallelcluster
```

Now you should have the `pcluster` command. You will use this command to performs actions like: creating a cluster, shutting your cluster down (temporarily), destroying a cluster, etc.

Create a cluster config file by running the **`pcluster configure`** command:

```
$ pcluster configure --config cluster-config.yaml
```

For instructions on `pcluster configure`, refer to the official instructions Configuring AWS ParallelCluster.

The following settings are recommended:

- Scheduler: slurm
- Operating System: alinux2
- Head node instance type: c5n.large
- Number of queues: 1
- Compute instance type: c5n.18xlarge
- Maximum instance count: Your choice. This is the maximum number execution nodes that can run concurrently. Execution nodes automatically spinup and shutdown according when there are jobs in your queue.

Now you should have a file name `cluster-config.yaml`. This is the configuration file with setting for a cluster.

Before starting your cluster with the **pcluster create-cluster** command, you can modify `cluster-config.yaml` to create cluster based on our AMIs. We provide the available AMI ID through AMI list.

You also need to modify `cluster-config.yaml` so that your FSx for Lustre file system is mounted to your cluster. Use the following `cluster-config.yaml` as a template for these changes.

```
Region: us-east-1  # [replace with] the region with your FSx for Lustre file system
Image:
  Os: alinux2
  CustomAmi: ami-AAAAAAAAAAAAAAAAA # [replace with] the AMI ID you want to use
HeadNode:
  InstanceType: c5n.large  # smallest c5n node to minimize costs when head-node is up
  Networking:
    SubnetId: subnet-YYYYYYYYYYYYYYYYY  # [replace with] the subnet of your FSx for␣
↪Lustre file system
    AdditionalSecurityGroups:
      - sg-ZZZZZZZZZZZZZZZZZ  # [replace with] the security group with inbound rules␣
↪for your FSx for Lustre file system
  LocalStorage:
    RootVolume:
      VolumeType: io2
  Ssh:
    KeyName: AAAAAAAAAA  # [replace with] the name of your ssh key name for AWS CLI
SharedStorage:
  - MountDir: /fsx  # [replace with] where you want to mount your FSx for Lustre file␣
↪system
    Name: FSxExtData
    StorageType: FsxLustre
    FsxLustreSettings:
      FileSystemId: fs-XXXXXXXXXXXXXXXXX  # [replace with] the ID of your FSx for␣
↪Lustre file system
Scheduling:
  Scheduler: slurm
  SlurmQueues:
  - Name: main
    ComputeResources:
    - Name: c5n18xlarge
      InstanceType: c5n.18xlarge
      MinCount: 0
      MaxCount: 10  # max number of concurrent exec-nodes
      DisableSimultaneousMultithreading: true  # disable hyperthreading (recommended)
      Efa:
        Enabled: true
    Networking:
      SubnetIds:
      - subnet-YYYYYYYYYYYYYYYYY  # [replace with] the subnet of your FSx for Lustre␣
↪file system (same as above)
      AdditionalSecurityGroups:
        - sg-ZZZZZZZZZZZZZZZZZ  # [replace with] the security group with inbound␣
↪rules for your FSx for Lustre file system
      PlacementGroup:
        Enabled: true
    ComputeSettings:
      LocalStorage:
        RootVolume:
          VolumeType: io2
```

When you are ready, run the **pcluster create-cluster** command.

```
$ pcluster create-cluster --cluster-name pcluster --cluster-configuration cluster-
↪config.yaml
```

It may take several minutes up to an hour for your cluster's status to change to `CREATE_COMPLETE`. You can check the status of you cluster with the following command.

```
$ pcluster describe-cluster --cluster-name pcluster
```

Once your cluster's status is `CREATE_COMPLETE`, run the **`pcluster ssh`** command to ssh into it.

```
$ pcluster ssh --cluster-name pcluster -i ~/path/to/keyfile.pem
```

At this point, your cluster is set up and you can use it like any other HPC. Now you can create a run directory by running the `createRunDir.sh` command. Your next steps will be following the normal instructions found in the User Guide.

16.4 4. Running GCHP on ParallelCluster

AWS ParallelCluster supports Slurm and AWS Batch job schedulers. Your cluster is set to use Slurm scheduler according to the configuration file. It might require the root permission to run Slurm commands or restart Slurm. Before you submit your job, you can start a shell as superuser by running `sudo -s`.

You can follow *Run the model* to run GCHP with Slurm scheduler.

CHAPTER

SEVENTEEN

CACHE INPUT DATA ON FAST DRIVES

This page describes how to set up a cache of GEOS-Chem input data. This is useful if you want to temporarily transfer a simulation's input data to a performant hard drive. This can improve the speed of your GCHP simulation by reducing the time spent reading input data. Caching input data is also useful if the file system that stores your GEOS-Chem input data repository has issues that are causing simulations to crash (i.e., you can transfer the data for your simulation to more stable hard drives).

17.1 Install the bashdatacatalog

Install the bashdatacatalog with the following command. Follow the prompts and restart your console.

```
gcuser:~$ bash <(curl -s https://raw.githubusercontent.com/LiamBindle/bashdatacatalog/
↪main/install.sh)
```

Note: You can rerun this command to upgrade to the latest version.

17.2 Set Up the ExtDataCache Directory

Next, we are going to set up the `ExtDataCache` directory. You should put this directory in the appropriate path so that desired hard drives are used. For example, if you have performance hard drives at `/scratch/`, create a directory like `/scratch/ExtDataCache/`. We are going to use `ExtDataCache/` to temporarily store the input data for simulations.

In the future, the idea is that you will copy the prerequisite input data to `ExtDataCache/` before you run a simulation. Since `ExtDataCache/` is temporary data, you can delete it periodically to "purge" it. Alternatively, you can use bashdatacatalog commands to selectively remove files. If you are running long simulations, you can keep a few years of data in `ExtDataCache/`, sort of like a moving window tracking the progress of your simulation.

Create a subdirectory in `ExtDataCache/` to store catalog files. You need a set of four catalog files for each simulation:

- MeteorologicalInputs.csv – Specifies the simulation's meteorological input data
- ChemistryInputs.csv – Specifies the simulation's chemistry input data
- EmissionsInputs.csv – Specifies the simulation's emissions input data
- InitialConditions.csv – Specifies the default restart files for the simulation

A good directory structure for catalog files is `ExtDataCache/CatalogFiles/SIMULATION_ID` where `SIMULATION_ID` is a placeholder for a unique identifier for your simulation. These instructions will put a demo set of catalog files in `ExtDataCache/CatalogFiles/DemoSimulation`:

```
gcuser:~$ cd /scratch
gcuser:/scratch$ mkdir ExtDataCache  # for storing input data for simulations
gcuser:/scratch$ mkdir ExtDataCache/CatalogFiles  # for storing catalog files
gcuser:/scratch$ mkdir ExtDataCache/CatalogFiles/DemoSimulation  # for storing␣
↪catalog files for a specific simulation
```

Next, download the catalog files for the appropriate version of GEOS-Chem. You can find the GEOS-Chem catalog files here.

```
gcuser:/scratch$ cd ExtDataCache/CatalogFiles/DemoSimulation
gcuser:/scratch/ExtDataCache/CatalogFiles/DemoSimulation$
gcuser:/scratch/ExtDataCache/CatalogFiles/DemoSimulation$ wget http://geoschemdata.
↪wustl.edu/ExtData/DataCatalogs/MeteorologicalInputs.csv
gcuser:/scratch/ExtDataCache/CatalogFiles/DemoSimulation$ wget http://geoschemdata.
↪wustl.edu/ExtData/DataCatalogs/13.3/ChemistryInputs.csv
gcuser:/scratch/ExtDataCache/CatalogFiles/DemoSimulation$ wget http://geoschemdata.
↪wustl.edu/ExtData/DataCatalogs/13.3/EmissionsInputs.csv
gcuser:/scratch/ExtDataCache/CatalogFiles/DemoSimulation$ wget http://geoschemdata.
↪wustl.edu/ExtData/DataCatalogs/13.3/InitialConditions.csv
```

Edit the catalog files according to your simulation configuration. You can enable/disable data collections by editing column 3 (`1` to enable a collection, `0` to disable a collection). If you are not sure if your simulation needs a collection, it is better to err on the side of inclusion. The meteorological data collections are the largest by volume. Only one meteorological data collection in `MeteorologicalInputs.csv` needs to be enabled.

17.3 Update the Collection URLs

The default collection URLs in the catalog files point to http://geoschemdata.wustl.edu/ExtData. To copy data from your primary ExtData repository, edit column 2 of the catalog files. For example, if your primary ExtData repository is at `/storage/ExtData` you would replace `http://geoschemdata.wustl.edu/ExtData` with `file:///storage/ExtData` in column 2 of the catalog files. Below is a **sed** command that will do the replacement.

```
gcuser:/scratch/ExtDataCache/CatalogFiles/DemoSimulation$ export FIND_STR="http://
↪geoschemdata.wustl.edu/ExtData"
gcuser:/scratch/ExtDataCache/CatalogFiles/DemoSimulation$ export REPLACE_STR="file:///
↪storage/ExtData"   # replace '/storage/ExtData' with the path to your ExtData
gcuser:/scratch/ExtDataCache/CatalogFiles/DemoSimulation$ sed -i "s#${FIND_STR}#$
↪{REPLACE_STR}#g" *.csv  # do url find/replace
```

17.4 Copy Data to ExtDataCache

Navigate to `ExtDataCache/`. One you are there, run **bashdatacatalog-fetch** to fetch metadata from ExtData. The arguments to **bashdatacatalog-fetch** are catalog files. This metadata includes the file list for each data collection, and the details to classify each file as a temporal or static file.

```
gcuser:/scratch/ExtDataCache/CatalogFiles/DemoSimulation$ cd ../..
gcuser:/scratch/ExtDataCache$ bashdatacatalog-fetch CatalogFiles/DemoSimulation/*.csv
```

Now you can run **bashdatacatalog-list** commands to generate file lists. The output of **bashdatacatalog-list** is controlled using flags. For example, add the `-s` to list "static" files (input files that are always required regardless of the simulation period). You can list "temporal" files with the `-t` flag. You can filter temporal files according to a date range with the `-r START,END` argument. You can filter out files that exist using the `-m` flag (lists files that are missing). You can specify different file list formats using the *-f FORMAT* argument. Below is a command that lists all the files in ExtDataCache that are missing for a simulation starting on 2017-01-01 and ending on 2017-12-31.

```
gcuser:/scratch/ExtDataCache$ bashdatacatalog-list -stm -r 2016-12-31,2018-01-01␣
↪CatalogFiles/DemoSimulation/*.csv
```

Note: You need to subtract/add one day to the period of your simulation. The example above uses `-r 2016-12-31,2018-01-01` because the simulation period is 2017-01-01 to 2017-12-31.

To copy the missing files to ExtDataCache, you can use the argument `-f xargs-curl` to specify the output list should be formatted as input to `xargs curl`. You can use a command similar to the one below to copy all the missing files for your simulation to ExtDataCache.

```
gcuser:/scratch/ExtDataCache$ bashdatacatalog-list -stm -r 2016-12-31,2018-01-01 -f␣
↪xargs-curl CatalogFiles/DemoSimulation/*.csv | xargs -P 4 curl
```

Note: The `-P 4` argument to **xargs** allows for 4 parallel copies at a time.

17.5 Update Run Directory to use ExtDataCache

To update a run directory to use ExtDataCache, you can run the following commands. Make sure to set `FIND_PATH` to ExtData and `REPLACE_PATH` to ExtDataCache.

```
gcuser:/scratch/ExtDataCache$ cd /MyRunDirectory  # cd to your run directory
gcuser:/MyRunDirectory$ export FIND_PATH=/storage/ExtData          # replace path to␣
↪your primary ExtData
gcuser:/MyRunDirectory$ export REPLACE_PATH=/scratch/ExtDataCache # replace with the␣
↪path to your ExtDataCache
gcuser:/MyRunDirectory$ function swap_extdata_link { ln -sfn $(readlink $1 | sed "s#$
↪{FIND_PATH}/*#${REPLACE_PATH}/#") $1; }
gcuser:/MyRunDirectory$ swap_extdata_link ChemDir
gcuser:/MyRunDirectory$ swap_extdata_link HcoDir
gcuser:/MyRunDirectory$ swap_extdata_link MetDir
gcuser:/MyRunDirectory$ sed -i "s#${FIND_PATH}#${REPLACE_PATH}#g" HEMCO_Config.rc␣
↪geoschem_config.yml
```

Now your GCHP simulation will use input data from ExtDataCache.

17.6 See Also

- bashdatacatalog - Instructions for GEOS-Chem Users
- bashdatacatalog - List of useful commands
- GEOS-Chem Input Data Catalogs

CHAPTER

EIGHTEEN

USE GCHP CONTAINERS

Containers are an effective method of packaging and delivering GCHP's source code and requisite libraries. We offer up-to-date Docker images for GCHP through Docker Hub. These images contain pre-built GCHP source code and the tools for creating a GCHP run directory. The instructions below show how to create a run directory and run GCHP using Singularity , which can be installed using instructions at the previous link or through Spack. Singularity is a container software that is preferred over Docker for many HPC applications due to security issues. Singularity can automatically convert and use Docker images. You can choose to use Docker or Singularity depending on the support of the cluster.

The workflow for running GCHP using containers is

1. Pull an image (*described on this page*)
2. Create a run directory (*use pre-built tools* or follow *Create a Run Directory*)
3. Download input data (*described on this page* and *Download Input Data*)
4. Running GCHP (*use pre-built tools* or follow *Run the model*)

18.1 Software requirements

There are only two software requirements for running GCHP using a Singularity container:

- Singularity itself
- An MPI implementation that matches the type and major/minor version of the MPI implementation inside of the container

18.2 Performance

Because we do not include optimized infiniband libraries within the provided Docker images, container-based GCHP is currently not as fast as other setups.

Container-based benchmarks deployed on Harvard's Cannon cluster using up to 360 cores at c90 (~1x1.25) resolution averaged 15% slower than equivalent non-container runs. Performance may worsen at a higher core count and resolution. If this performance hit is not a concern, these containers are the quickest way to setup and run GCHP.

18.3 Pulling an image and creating run directory using Singularity

Available GCHP images are listed on Docker Hub. The following command pulls the image of GCHP 14.2.0 and converts it to a Singularity image named *gchp.sif* in your current directory.

```
$ singularity pull gchp.sif docker://geoschem/gchp:14.2.0
```

If you do not already have GCHP data directories, create a directory where you will later store data files. We will call this directory `DATA_DIR` and your run directory destination `WORK_DIR` in these instructions. Make sure to replace these names with your actual directory paths when executing commands from these instructions.

The following command executes GCHP's run directory creation script. Within the container, your `DATA_DIR` and `WORK_DIR` directories are visible as `/ExtData` and `/workdir`. Use `/ExtData` and `/workdir` when asked to specify your ExtData location and run directory target folder, respectively, in the run directory creation prompts.

```
$ singularity exec -B DATA_DIR:/ExtData -B WORK_DIR:/workdir gchp.sif /bin/bash -c ". ␣
↪~/.bashrc && /opt/geos-chem/bin/createRunDir.sh"
```

Once the run directory is created, it will be available at `WORK_DIR` on your host machine. **`cd`** to `WORK_DIR`.

18.4 Setting up and running GCHP using Singularity

To avoid having to specify the locations of your data and run directories (RUN_DIR) each time you execute a command in the singularity container, we will add these to an environment file called `~/.container_run.rc` and point the `gchp.env` symlink to this environment file. We will also load MPI in this environment file (edit the first line below as appropriate to your system).

```
$ echo "module load openmpi/4.0.3" > ~/.container_run.rc
$ echo "export SINGULARITY_BINDPATH=\"DATA_DIR:/ExtData,RUN_DIR:/rundir\"" >> ~/.
↪container_run.rc
$ ./setEnvironmentLink.sh ~/.container_run.rc
$ source gchp.env
```

We will now move the pre-built `gchp` executable and example run scripts to the run directory.

```
$ rm runScriptSamples # remove broken link
$ singularity exec ../gchp.sif cp /opt/geos-chem/bin/gchp /rundir
$ singularity exec ../gchp.sif cp -rf /gc-src/run/runScriptSamples/ /rundir
```

Before running GCHP in the container, we need to create an execution script to tell the container to load its internal environment before running GCHP. We'll call this script `internal_exec`.

```
$ echo -e "if [ -e \"/init.rc\" ] ; then\n\t. /init.rc\nfi" > ./internal_exec # no␣
↪need for versions after 13.4.1
$ echo "cd /rundir" >> ./internal_exec
$ echo "./gchp" >> ./internal_exec
$ chmod +x ./internal_exec
```

The last change you need to make to run GCHP in a container is to edit your run script (whether from `runScriptSamples/` or otherwise). Replace the typical execution line in the script (where **`mpirun`** or **`srun`** is called) with the following:

```
$ time mpirun singularity exec ../gchp.sif /rundir/internal_exec >> ${log}
```

You can now setup your run configuration as normal using `setCommonRunSettings.sh` and tweak Slurm parameters in your run script.

If you already have GCHP data directories, congratulations! You've completed all the steps you need to run GCHP in a container. If you still need to download data directories, read on.

18.5 Downloading data directories using GEOS-Chem Classic's dry-run option

GCHP does not currently support automated download of requisite data directories, unlike GEOS-Chem Classic. Luckily we can use a GC Classic container to execute a dry-run that matches the parameters of our GCHP run to download data files.

```
$ #get GC Classic image from https://hub.docker.com/r/geoschem/gcclassic
$ singularity pull gcc.sif docker://geoschem/gcclassic:13.0.0-alpha.13-7-ge472b62
$ #create a GC Classic run directory (GC_CLASSIC_RUNDIR) in WORK_DIR that matches
$ #your GCHP rundir (72-level, standard vs. benchmark vs. transport tracers, etc.)
$ singularity exec -B WORK_DIR:/workdir gcc.sif /opt/geos-chem/bin/createRunDir.sh
$ cd GC_CLASSIC_RUNDIR
$ #get pre-compiled GC Classic executable
$ singularity exec -B .:/classic_rundir ../gcc.sif cp /opt/geos-chem/bin/gcclassic /
↪classic_rundir
```

Make sure to tweak dates of run in geoschem_config.yml as needed, following info here.

```
$ #create an internal execute script for your container
$ echo ". /init.rc" > ./internal_exec
$ echo "cd /classic_rundir" >> ./internal_exec
$ echo "./gcclassic --dryrun" >> ./internal_exec
$ chmod +x ./internal_exec
$ #run the model, outputting requisite file info to log.dryrun
$ singularity exec -B .:/classic_rundir ../gcc.sif /classic_rundir/internal_exec >␣
↪log.dryrun
```

Follow instructions here for downloading your relevant data. Note that you will still need a restart file for your GCHP run which will not be automatically retrieved by this download script.

CHAPTER

NINETEEN

STRETCHED-GRID SIMULATION

Note: Stretched-grid simulations are described in [[Bindle et al., 2021]]. This paper also discusses related tpics of consideration and offers guidance for choosing appropriate stretching parameters.

19.1 Overview

A stretched-grid is a cubed-sphere grid that is "stretched" to enhance its resolution in a region. To set up a stretched-grid simulation you need to do the following:

1. Choose stretching parameters, including stretch factor and target latitude and longitude.
2. Create a stretched grid restart file for your simulation using your chosen stretch parameters.
3. Configure the GCHP run directory to specify stretched grid parameters in `setCommonRunSettings.sh` and use your stretched grid restart file.

19.1.1 Choose stretching parameters

The *target face* is the face of a stretched-grid that shrinks so that the grid resolution is finer. The target face is centered on a target point, and the degree of stretching is controlled by a parameter called the stretch-factor. Relative to a normal cubed-sphere, the resolution of the target face is refined by approximately the stretch-factor. For example, a C60 stretched-grid with a stretch-factor of 3.0 has approximately C180 (~50 km) resolution in the target face. The enhancement-factor is approximate because (1) the stretching gradually changes with distance from the target point, and (2) gnominic cubed-sphere grids are quasi-uniform with grid-boxes at face edges being ~1.5x shorter than at face centers.

You can choose a stretch-factor and target point using the interactive figure below. You can reposition the target face by changing the target longitude and target latitude. The domain of refinement can be increased or decreased by changing the stretch-factor. Choose parameters so that the target face roughly covers the refion that you want to refine.

Note: The interactive figure above can be a bit fiddly. Refresh the page if the view gets messed up. If the figure above is not showing up properly, please open an issue.

Next you need to choose a cubed-sphere size. The cubed-sphere size must be an even integer (e.g., C90, C92, C94, etc.). Remember that the resolution of the target face is enhanced by approximately the stretch-factor.

19.1.2 Create a restart file

A simulation restart file must have the same grid as the simulation. For example, a C180 simulation requires a restart file with a C180 grid. Likewise, a stretched-grid simulation needs a restart file with the same stretched-grid (i.e., an identical cubed-sphere size, stretch-factor, target longitude, and target latitude).

You can regrid an existing restart file to a stretched-grid using the GEOS-Chem python package GCPy. See the Regridding section of the GCPy documentation for instructions. Once you have created a restart file for your simulation, you can move on to updating your simulation's configuration files.

Note: A stretched grid restart file is available for download if you would like to quickly get set up to run a stretched grid simulation. See the GEOSCHEM_RESTARTS/GC_14.0.0 directory in the GEOS-Chem data repository.

Configure run directory

Modify the section of `setCommonRunSettings.sh` that controls the simulation grid. Turn `STRETCH_GRID` to `ON` and update `CS_RES`, `STRETCH_FACTOR`, `TARGET_LAT`, and `TARGET_LON` for your specific grid.

```
#------------------------------------------------
#   GRID RESOLUTION
#------------------------------------------------
# Integer representing number of grid cells per cubed-sphere face side
CS_RES=24

#------------------------------------------------
#   STRETCHED GRID
#------------------------------------------------
# Turn stretched grid ON/OFF. Follow these rules if ON:
#    (1) Minimum STRETCH_FACTOR value is 1.0001
#    (2) TARGET_LAT and TARGET_LON are floats containing decimal
#    (3) TARGET_LON in range [0,360)
STRETCH_GRID=OFF
STRETCH_FACTOR=3.0
TARGET_LAT=40.0
TARGET_LON=260.0
```

Execute **`./setCommonRunSettings.sh`** to update your run directory's configuration files.

```
$ ./setCommonRunSettings.sh
```

You will also need to configure the run directory to use the stretched grid restart file.

1. Update `cap_restart` to match the date of your restart file. This will also be the start date of the run.
2. Copy or symbolically link to your restart file in the `Restarts` subdirectory with the proper filename format. The format includes global resolution but not stretched grid resolution. To avoid confusion about what grid the file contains you can symbolically link to a file with stretched grid parameters in its filename.
3. Run `setRestartLink.sh` to set symbolic link `gchp_restart.nc4` to point to your restart file based on start date in `cap_restart` and global grid resolution in `setCommonRunSettings.sh`. This is also included as pre-run step in all example run scripts provided in `runScriptSamples`.

19.2 Tutorial: Eastern United States

This tutorial walks you through setting up and running a stretched-grid simulation for ozone in the eastern United States. The grid parameters for this tutorial are:

Parameter	Value
Stretch-factor	3.6
Cubed-sphere size	C60
Target latitude	37° N
Target longitude	275° E

These parameters are chosen so that the target face covers the eastern United States. Some back-of-the-envelope resolution calculations are:

$$\text{average resolution of target face} = \mathrm{R_{tf}} \approx \frac{10000\ \mathrm{km}}{\mathrm{N} \times \mathrm{S}} = 46\ \mathrm{km}$$

$$\text{coarsest resolution in target face (at the center)} \approx \mathrm{R_{tf}} \times 1.2 = 56\ \mathrm{km}$$

$$\text{finest resolution in target face (at the edges)} \approx \mathrm{R_{tf}} \div 1.2 = 39\ \mathrm{km}$$

$$\text{coarsest resolution globally (at target antipode)} \approx \mathrm{R_{tf}} \times \mathrm{S}^2 \times 1.2 = 720\ \mathrm{km}$$

where N is the cubed-sphere size and S is the stretch-factor. The actual values of these, calculated from the grid-box areas, are 46 km, 51 km, 42 km, and 664 km respectively.

Note: This tutorial uses a relatively large stretch-factor. A smaller stretch-factor, such as 2.0 rather than 3.6, would have a broader refinement and smaller range resolutions.

19.2.1 Requirements

Before continuing with the tutorial check that you have all pre-requisites:

- You are able to run global GCHP simulations using MERRA2 data for July 2019
- You have the latest version of GEOS-Chem python package GCPy
- You have python package cartopy with version >= 0.19

19.2.2 Create run directory

Create a standard full chemistry run directory that uses MERRA2 meteorology. The rest of the tutorial assume that your current working directory is your run directory.

19.2.3 Create restart file

You will need to create a restart file with a horizontal resolution that matches your chosen stretched-grid resolution. Unlike other input data, GCHP ingests the restart file with no online regridding. Using a restart file with a horizontal grid that does not match the run grid will result in a run-time error. To create a restart file for a stretched-grid simulation you can regrid a restart file with a uniform grid using GCPy. Follow instructions on how to create a GCHP stretched grid restart file in the GCPy documentation. For this tutorial regrid the c48 fullchem restart file for July 1, 2019 that comes with a GCHP fullchem run directory (`GEOSChem.Restart.20190701_0000z.c48.nc4`). Grid resolution is 60, stretch factor is 3.6, target longitude is 275, and target latitude is 37. Name the output file `initial_GEOSChem_rst.EasternUS_SG_fullchem.c60.s3.6_37N_275E.nc`.

19.2.4 Configure run directory

Make the following modifications to `setCommonRunSettings.sh`:

- Change the simulation's duration to 7 days
- Turn on auto-update of diagnostics
- Set diagnostic frequency to 24 hours (daily)
- Set diagnostic duration to 24 hours (daily)
- Update the compute resources as you like. This simulation's computational demands are about 50% more than a C48 or 2°x2.5° simulation.
- Change global grid resolution to 60
- Change `STRETCH_GRID` to `ON`
- Change `STRETCH_FACTOR` to `3.6`
- Change `TARGET_LAT` to `37.0`
- Change `TARGET_LON` to `275.0`

Note: In our tests this simulation took approximately 7 hours to run using 30 cores on 1 node. For comparison, it took 2 hours to run using 180 cores across 6 notes. You may choose your compute resources based on how long you are willing to wait for your run to end.

Next, execute `setCommonRunSettings.sh` to apply the updates to the various configuration files:

```
$ ./setCommonRunSettings.sh
```

Before running GCHP you also need to configure the model to use your stretched-grid restart file. Move or copy your restart file to the `Restarts` subdirectory. Then change the symbolic link `GEOSChem.Restart.20190701_0000z.c48.nc4` to point to your stretched-grid restart file while keeping the name of the link the same.

```
$ ln -nsf initial_GEOSChem_rst.EasternUS_SG_fullchem.c60.s3.6_37N_275E.nc GEOSChem.
↪Restart.20190701_0000z.c48.nc4
```

You could also rename your restart file to this format but this would remove valuable information about the content of the file from the filename. Symbolically linking is a better way to preserve the information to avoid errors. You can check that you did this correctly by running `setRestartLink.sh` in the run directory.

19.2.5 Run GCHP

To run GCHP you can use the example run script for running interactively located at `runScriptSamples/gchp.local.run` as long as you have enough resources available locally, e.g. 30 cores on 1 node. Copy it to the main level of your run directory and then execute it. If you want to use more resources you can submit as a batch job to your scheduler.

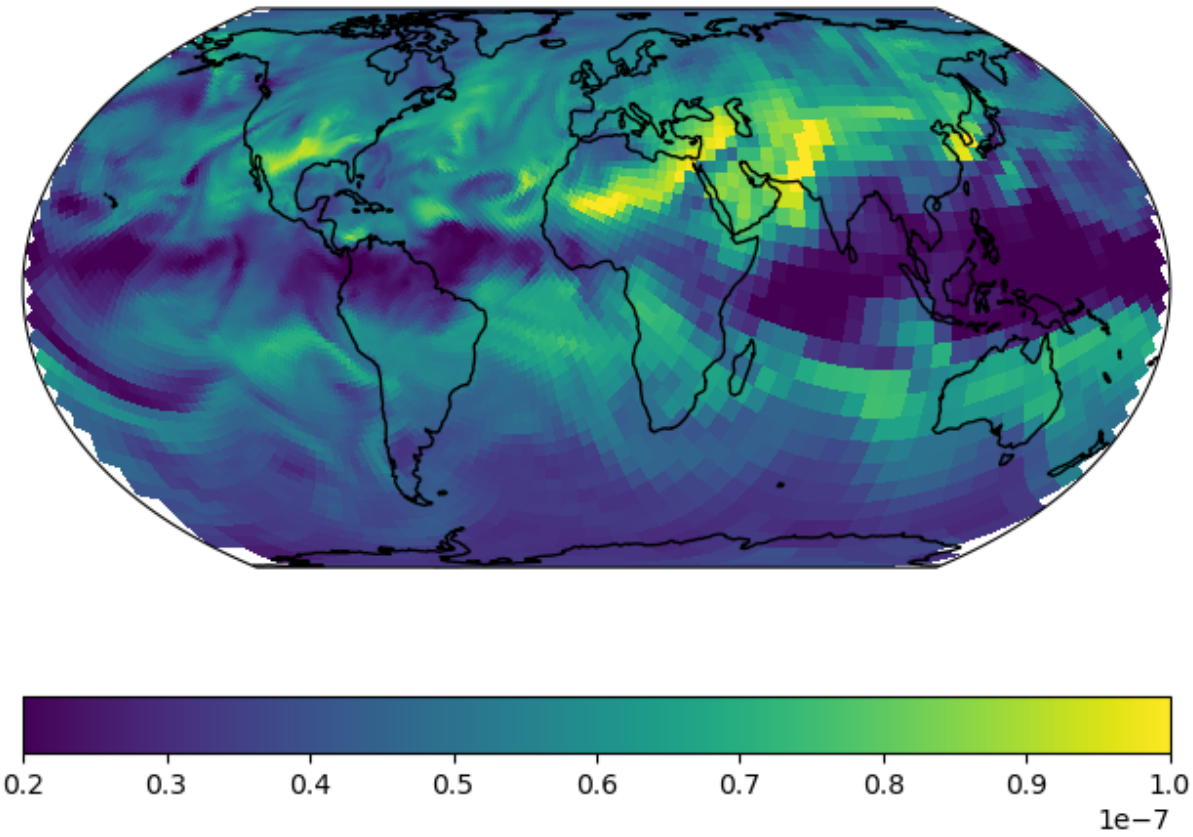
```
$ ./gchp.local.run
```

Log output of the run will be sent to log file `gchp.20190701_0000z.log`. Check that your run was successful by inspecting the log and looking for output in the `OutputDir` subdirectory.

19.2.6 Plot the output

Plotting stretched grid is simple using Python. Below is an example plotting ozone at model level 22. All libraries are available if using a python environment compatible with GCPy.

```
import matplotlib.pyplot as plt
import cartopy.crs as ccrs
import xarray as xr

# Load 24-hr average concentrations for 2019-07-01
ds = xr.open_dataset('GCHP.DefaultCollection.20190701_0000z.nc4')

# Get Ozone at level 22
ozone_data = ds['SpeciesConcVV_O3'].isel(time=0, lev=22).squeeze()

# Setup axes
ax = plt.axes(projection=ccrs.EqualEarth())
ax.set_global()
ax.coastlines()

# Plot data on each face
for face_idx in range(6):
    x = ds.corner_lons.isel(nf=face_idx)
    y = ds.corner_lats.isel(nf=face_idx)
    v = ozone_data.isel(nf=face_idx)
    pcm = plt.pcolormesh(
        x, y, v,
        transform=ccrs.PlateCarree(),
        vmin=20e-9, vmax=100e-9
    )

plt.colorbar(pcm, orientation='horizontal')
plt.show()
```


0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
1.0
1e−7

CHAPTER

TWENTY

OUTPUT ALONG A TRACK

HISTORY collections can define a `track_file` that specifies a 1D timeseries of coordinates that the model is sampled at. The collection output has the same coordinates as the track file. This feature can be used to sample GCHP along a satellite track or a flight path. A track file is a NetCDF file with the following format

```
$ ncdump -h example_track.nc
netcdf example_track.nc {
dimensions:
   time = 1234 ;
variables:
   float time(time) ;
      time:_FillValue = NaNf ;
      time:long_name = "time" ;
      time:units = "hours since 2020-06-01 00:00:00" ;
   float longitude(time) ;
      longitude:_FillValue = NaNf ;
      longitude:long_name = "longitude" ;
      longitude:units = "degrees_east" ;
   float latitude(time) ;
      latitude:_FillValue = NaNf ;
      latitude:long_name = "latitude" ;
      latitude:units = "degrees_north" ;
}
```

Important: Longitudes must be between 0 and 360.

Important: When using `recycle_track`, the time offsets must be between 0 and 24 hours.

To configure 1D output, you can add the following attributes to any collection in `HISTORY.rc`.

track_file Path to a track file. The associated collection will be sampled from the model along this track. A track file is a 1-dimensional timeseries of latitudes and longitudes that the model is be sampled at (nearest neighbor).

recycle_track Either `.false.` (default) or `.true.`. When enabled, HISTORY replaces the date of the `time` coordinate in the track file with the simulation's current day. This lets you use the same track file for every day of your simulation.

Note: 1D output only works for instantaneous sampling.

The `frequency` attribute is ignored when `track_file` is used.

20.1 Creating a satellite track file

GCPy includes a command line tool, **gcpy.raveller_1D**, for generating track files for polar orbiting satellites. These track files will sample model grid-boxes at the times that correspond to the satellite's overpass time. You can also use this tool to "unravel" the resulting 1D output back to a cubed-sphere grid. Below is an example of using **gcpy.raveller_1D** to create a track file for a C180 simulation for TROPOMI, which is in ascending sun-synchronous orbit with 14 orbits per day and an overpass time of 13:30. Please see the GCPy documentation for this program's exact usage, and for installation instructions.

```
$ python -m gcpy.raveller_1D create_track --cs_res 24 --overpass_time 13:30 --
↪direction ascending --orbits_per_day 14 -o tropomi_overpass_c24.nc
```

The resulting track file, `tropomi_overpass_c24.nc`, looks like so

```
$ ncdump -h tropomi_overpass_c24.nc
netcdf tropomi_overpass_c24 {
dimensions:
   time = 3456 ;
variables:
   float time(time) ;
      time:_FillValue = NaNf ;
      time:long_name = "time" ;
      time:units = "hours since 1900-01-01 00:00:00" ;
   float longitude(time) ;
      longitude:_FillValue = NaNf ;
      longitude:long_name = "longitude" ;
      longitude:units = "degrees_east" ;
   float latitude(time) ;
      latitude:_FillValue = NaNf ;
      latitude:long_name = "latitude" ;
      latitude:units = "degrees_north" ;
   float nf(time) ;
      nf:_FillValue = NaNf ;
   float Ydim(time) ;
      Ydim:_FillValue = NaNf ;
   float Xdim(time) ;
      Xdim:_FillValue = NaNf ;
}
```

Note: Track files do not require the `nf`, `Ydim`, `Xdim` variables. The are used for post-process "ravelling" with **gcpy.raveller_1D** (changing the 1D output's coordinates to a cubed-sphere grid).

Note: With `recycle_track`, HISTORY replaces the reference date (e.g., 1900-01-01) with the simulation's current date, so you can use any reference date.

20.2 Updating HISTORY

Open `HISTORY.rc` and add the `track_file` and `recycle_track` attributes to your desired colleciton. For example, the following is a custom collection that samples NO2 along the `tropomi_overpass_c24.nc`.

```
  TROPOMI_NO2.template:       '%y4%m2%d2_%h2%n2z.nc4',
  TROPOMI_NO2.format:         'CFIO',
  TROPOMI_NO2.duration:       240000
  TROPOMI_NO2.track_file:     tropomi_overpass_c24.nc
  TROPOMI_NO2.recycle_track:  .true.
  TROPOMI_NO2.mode:           'instantaneous'
  TROPOMI_NO2.fields:         'SpeciesConc_NO2            ', 'GCHPchem',
::
```

20.3 Unravelling 1D overpass timeseries

To covert the 1D timeseries back to a cubed-sphere grid, you can use **`gcpy.raveller_1D`**. Below is an example of changing the 1D output back to model grid. Again, see the GCPy documentation for this program's exact usage, and for installation instructions.

```
$ python -m gcpy.raveller_1D unravel --track tropomi_overpass_c24.nc -i OutputDir/
↪GCHP.TROPOMI_NO2.20180101_1330z.nc4 -o OutputDir/GCHP.TROPOMI_NO2.20180101_1330z.
↪OVERPASS.nc4
```

The resulting dataset, `GCHP.TROPOMI_NO2.20180101_1330z.OVERPASS.nc4`, are simulated concentration on the model grid, sampled at the times that correspond to TROPOMI's overpass.

CHAPTER

TWENTYONE

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.

CHAPTER

TWENTYTWO

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 procesing.

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_O3: {ds['SpeciesConc_O3'].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_O3: {np.sum(ds['SpeciesConc_O3'].values)}")
print(f"Max of SpeciesRst_O3: {np.max(ds['SpeciesConc_O3'].values)}")
print(f"Min of SpeciesRst_O3: {np.min(ds['SpeciesConc_O3'].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: 1.012129541777 54e-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 shiftime,-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 *nco*) to concatenate the individual netCDF files into a single netCDF file.

Let's assume we want to combine 12 monthy 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 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 *nco* 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 argumetnt 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**.

CHAPTER

TWENTYTHREE

PREPARE COARDS-COMPLIANT NETCDF FILES

On this page we discuss how you can generate netCDF data files in the proper format for HEMCO and and GEOS-Chem.

23.1 The COARDS netCDF standard

The Harmonized Emissions Compionent (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 be 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 n

ames 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 folllowing 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 emisisons 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 *lat* 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, PARAMTER   :: 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 maniuplate data in netCDF files.

CHAPTER

TWENTYFOUR

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.

24.1 Aerosols

24.1.1 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.

24.2 Chemistry

24.2.1 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.

24.2.2 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
```

24.2.3 HO2 heterogeneous chemistry reaction probability

You may update the value of γ_{HO2} (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
```

24.2.4 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.

24.3 Diagnostics

24.3.1 GEOS-Chem and HEMCO diagnostics

Please see our Diagnostics reference chapter for an overview of how to archive diagnostics from GEOS-Chem and HEMCO.

24.3.2 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.

24.4 Emissions

24.4.1 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** ccalculations 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 meteorlogical 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.

24.4.2 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
```

24.5 Photolysis

24.5.1 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.

24.6 Wet deposition

24.6.1 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
```

CHAPTER

TWENTYFIVE

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.

25.1 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.

25.2 Compile-time errors

In this section we discuss some compilation warnings that you may encounter when building GEOS-Chem.

25.2.1 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*.

25.2.2 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*.

25.2.3 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

25.3 Run-time errors

25.3.1 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.

25.3.2 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 (`Hnew`) to zero before calling the Rosenbrock integrator again. In many instances, this is sufficient for the chemistry to converge to a soluiton.

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 `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.

25.3.3 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 `EFYO`. 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.

25.3.4 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.

25.3.5 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.

25.3.6 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. into the same 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 te 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 . . .
```

25.3.7 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
 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 depoosition.
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 concetnrations are going negative.

25.3.8 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
```

25.3.9 Excessive fall velocity error

```
GEOS-CHEM ERROR:  Excessive fall velocity?
STOP at  CALC_FALLVEL, UCX_mod
```

Problem: The fall velocity (in stratopsheric 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.

25.4 File I/O errors

25.4.1 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.

25.4.2 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.

25.4.3 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 occured at the original source of te data.

25.5 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.

25.5.1 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 wil 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.

25.5.2 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.

25.5.3 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.

25.5.4 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.

25.6 Less commmon errors

The errors listed below, which occur infrequently, are related to invalid memory operations. These can especially occur with `POINTER`-based variables.

25.6.1 Bus Error

Problem: GEOS-Chem or HEMCO is trying to reference memory that cannot possibly be there. The website StackOverflow.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.

25.6.2 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*.

25.6.3 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.

25.6.4 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.

25.6.5 Munmap_chunk: invalid pointer

```
** glibc detected *** PROGRAM_NAME: munmap_chunk(): invalid pointer:␣
↪0x00000000059aac30 ***
```

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.

25.6.6 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`.

CHAPTER

TWENTYSIX

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.

26.1 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.

26.2 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

26.3 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.

26.4 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.

26.5 Send debug printout to the log files

If your **GEOS-Chem** simulation stopped with an error, but you cannot tell where, turn on the 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.

26.6 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).

26.7 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.

26.8 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.

26.9 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.

26.10 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`.

26.11 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 )
   ```

26.12 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.

26.13 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.

CHAPTER

TWENTYSEVEN

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

27.1 Species properties defined

The following sections contain a detailed description of GEOS-Chem species properties.

27.1.1 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.

27.1.2 Identification

Name
Species short name (e.g. `ISOP`).

Formula
Species chemical formula (e.g. `CH2=C(CH3)CH=CH2`). 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`).

27.1.3 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 depositon 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.

27.1.4 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
Specfies minimum dry deposition velocities ($cm\ s^{-1}$) for sulfate species (SO2, SO4, MSA, NH3, NH4, 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.

27.1.5 Wet deposition properties

WD_Is_H2SO4
Indicates that the species is H2SO4 (true), or isn't (false). This allows the wet deposition code to perform special calculations when computing H2SO4 rainout and washout.

WD_Is_HNO3
Indicates that the species is HNO3 (true), or isn't (false). This allows the wet deposition code to perform special calculations when computing HNO3. rainout and washout.

WD_Is_SO2
Indicates that the species is SO2 (true), or isn't (false). This allows the wet deposition code to perform special calculations when computing SO2 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 μm.

WD_LiqAndGas
Indicates that the 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 volatization 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.

27.1.6 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 horizonatal 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
```

27.1.7 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 $10^{-20} vol\ vol^{-1}$ 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.

27.2 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.

27.2.1 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
`ModelId`	Model species index
`AdvectId`	Advected species index
`AerosolId`	Aerosol species index
`DryAltId`	Dry dep species at altitude Id
`DryDepId`	Dry deposition species index
`GasSpcId`	Gas-phase species index
`HygGrthId`	Hygroscopic growth species index
`KppVarId`	KPP variable species index
`KppFixId`	KPP fixed spcecies index
`KppSpcId`	KPP species index
`PhotolId`	Photolyis species index
`RadNuclId`	Radionuclide index
`TracerId`	Transport tracer index
`WetDepId`	Wet deposition index

27.2.2 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
```

27.2.3 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
```

27.2.4 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
```

27.2.5 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
```

CHAPTER

TWENTYEIGHT

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.

28.1 Using KPP: Quick start

28.1.1 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
```

28.1.2 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`.

28.1.3 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.

28.1.4 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 exectuable. 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.

28.2 Using KPP: Reference section

28.2.1 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.

28.2.2 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 O3 + NO produces NO2 and O2, and its Arrhenius parameters are A = 3.0x10^-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 $O3 + NO = NO2 + O2$ 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 $O3 + NO = NO2 + O2$ 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 O3 + 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`.

28.2.3 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}
```

28.2.4 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 = 2.000O :                            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 $NO_3 = NO2 + O$:

```
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 ast `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.

28.2.5 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 + HOCl +␣
 ↪ClNO2 + 2ClNO3 + 2Cl2O2 + 2OClO + 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 + HOCl +␣
 ↪ClNO2 + 2ClNO3 + 2Cl2O2 + 2OClO + 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 65332 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 avaialble stack memory (which will result in the
//       infamous "Segmentation fault 11" error).  If you are stll 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.

28.2.6 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	#INTEGRATOR	#AUTOREDUCE
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.

CHAPTER

TWENTYNINE

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

CHAPTER

THIRTY

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.

30.1 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.

30.2 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.

30.3 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).

30.4 How to submit changes

Please see **Contributing Guidelines**.

30.5 How to request an enhancement

Please see **Contributing Guidelines**.

CHAPTER

THIRTYONE

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 GCHP source code, to track issues, user questions, and feature requests, and to accept pull requests: https://github.com/geoschem/GCHP. Please help out as you can in response to issues and user questions.

GCHP Classic documentation can be found at gchp.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 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 migth 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**.

CHAPTER

THIRTYTWO

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), wh ich Sphinx extends for software documentation. The source for the documentation is the `docs/source` directory in top-level of the source code.

32.1 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`.

32.2 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.

32.3 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) occuring in the text should use the `:file:` role.

Program names (e.g. **`cmake`**) occuring in the text should use the `:program:` role.

OS-level commands (e.g. **`rm`**) occuring in the text should use the `:command:` role.

Environment variables occuring in the text should use the `:envvar:` role.

Inline code or code variables occuring 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.

CHAPTER

THIRTYTHREE

GIT SUBMODULES

33.1 Forking submodules

This sections describes updating git submodules to use your own forks. You can update submodule so that they use your forks at any time. It is recommended you only update the submodules that you need to, and that you leave submodules that you don't need to modify pointing to the GEOS-Chem repositories.

The rest of this section assumes you are in the top-level of GCHP, i.e.,

```
$ cd GCHP    # navigate to top-level of GCHP
```

First, identify the submodules that you need to modify. The `.gitmodules` file has the paths and URLs to the submodules. You can see it with the following command

```
$ cat .gitmodules
[submodule "src/MAPL"]
   path = src/MAPL
   url = https://github.com/sdeastham/MAPL
[submodule "src/GMAO_Shared"]
   path = src/GMAO_Shared
   url = https://github.com/geoschem/GMAO_Shared
[submodule "ESMA_cmake"]
   path = ESMA_cmake
   url = https://github.com/geoschem/ESMA_cmake
[submodule "src/gFTL-shared"]
   path = src/gFTL-shared
   url = https://github.com/geoschem/gFTL-shared.git
[submodule "src/FMS"]
   path = src/FMS
   url = https://github.com/geoschem/FMS.git
[submodule "src/GCHP_GridComp/FVdycoreCubed_GridComp"]
   path = src/GCHP_GridComp/FVdycoreCubed_GridComp
   url = https://github.com/sdeastham/FVdycoreCubed_GridComp.git
[submodule "src/GCHP_GridComp/GEOSChem_GridComp/geos-chem"]
   path = src/GCHP_GridComp/GEOSChem_GridComp/geos-chem
   url = https://github.com/sdeastham/geos-chem.git
[submodule "src/GCHP_GridComp/HEMCO_GridComp/HEMCO"]
   path = src/GCHP_GridComp/HEMCO_GridComp/HEMCO
   url = https://github.com/geoschem/HEMCO.git
```

Once you know which submodules you need to update, fork each of them on GitHub.

Once you have your own forks for the submodules that you are going to modify, update the submodule URLs in `.gitmodules`

```
$ git config -f .gitmodules -e    # opens editor, update URLs for your forks
```

Synchronize your submodules

```
$ git submodule sync
```

Add and commit the update to `.gitmodules`.

```
$ git add .gitmodules
$ git commit -m "Updated submodules to use my own forks"
```

Now, when you push to your GCHP fork, you should see the submodules point to your submodule forks.

CHAPTER

THIRTYFOUR

TERMINOLOGY

absolute path The full path to a file, e.g., `/example/foo/bar.txt`. An absolute path should always start with `/`. As opposed to a *relative path*.

build See *compile*.

build directory A directory where build configuration settings are stored, and where intermediate build files like object files, module files, and libraries are stored.

checkpoint file See *restart file*.

compile Generating an executable program from source code (which is in a plain-text format).

dependencies The software libraries that are needed to compile GCHP. These include HDF5, NetCDF, and ESMF. See *Software Requirements* for a complete list.

environment The software packages and software configuration that are active in your current *terminal* or *script*. In Linux, the `$HOME/.bashrc` script performs automatic configuration when your terminal starts. You can manually configure your environment by running commands like **`source path_to_a_script`** or with tools like TCL or LMod for modulefiles. Software containers are effectively a prepackaged operating system + software + environment.

gridded component A formal model component. MAPL organizes model components with a tree structure, and facilitates component interconnections.

HISTORY The MAPL *gridded component* that handles model output. All GCHP output diagnostics are facilitated by HISTORY.

relative path The path to a file relative to the current working directory. For example, the relative path to `/example/foo/bar.txt` if your current working directory is `/example` is `foo/bar.txt`. As opposed to an *absolute path*.

restart file A NetCDF file with initial conditions for a simulation. Also called a *checkpoint file* in GCHP.

run directory The working directory for a GEOS-Chem simulation. A run directory houses the simulation's configuration files, the output directory (`OutputDir`), and input files/links such as *restart files* or input data directories.

script A file that scripts a sequence of commands. Typically a bash that is written to execute a sequence of commands.

software environment See *environment*.

stretched-grid A cubed-sphere grid that is "stretched" to enhance the grid resolution in a region.

target face The face of a stretched-grid that is refined. The target face is centered on the target point.

terminal A command-line.

CHAPTER

THIRTYFIVE

GCHP VERSION HISTORY

For a list of updates by GCHP version, please see:

- CHANGELOG.md for the GEOS-Chem science codebase
- CHANGELOG.md for the GCHP wrapper
- CHANGELOG.md for HEMCO

CHAPTER

THIRTYSIX

UPLOAD TO SPACK

This page describes how to upload recipe changes to Spack. Common recipe changes include updating available versions of GCHP and changing version requirements for dependencies.

1. Create a fork of https://github.com/spack/spack.git and clone your fork.
2. Change your `SPACK_ROOT` environment variable to point to the root directory of your fork clone.
3. Create a descriptive branch name in the clone of your fork and checkout that branch.
4. Make any changes to `$SPACK_ROOT/var/spack/repos/builtin/packages/package_name/` as desired.
5. Install Flake8 and mypy using `conda install flake8` and `conda install mypy` if you don't already have these packages.
6. Run Spack's style tests using `spack style`, which will conduct tests in `$SPACK_ROOT` using Flake8 and mypy.
7. (Optional) Run Spack's unit tests using `spack unit-test`. These tests may take a long time to run. The unit tests will always be run when you submit your PR, and the unit tests primarily test core Spack features unrelated to specific packages, so you don't usually need to run these manually.
8. Prefix your commit messages with the package name, e.g. `gchp: added version 13.1.0`.
9. Push your commits to your fork.
10. Create a PR targetted to the `develop` branch of the original Spack repository, prefixing the PR title with the package name, e.g. `gchp: added version 13.1.0`.

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.

[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.

[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., 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., Eastham4, 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.

[Martin et al., 2022] Martin, R. V., Eastham, S. D., Bindle, L., Lundgren, E. W., Clune, T. L., Keller, C. A., Downs, W., Zhang, D., Lucchesi, R. A., Sulprizio, M. P., Yantosca, R. M., Li, Y., Estrada, L., Putman, W. M., Auer, B. M., Trayanov, A. L., Pawson, S., and Jacob, D. J. Improved Advection, Resolution, Performance,

and Community Access in the New Generation (Version 13) of the High Performance GEOS-Chem Global Atmospheric Chemistry Model (GCHP). *Geosci. Model Dev. Discuss.*, 2022:1–30, 2022. doi:10.5194/gmd-2022-42.

[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.

[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., 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.

INDEX

Symbols

A

B

C

D

E

F

G

H

I

L

M

N

O

P

R

S

T

U

W

X