GCHP

Release 14.0.0

GEOS-Chem Support Team

Feb 17, 2023

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 *GCHP's 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:

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

Checkout the GEOS-Chem version that you want to use:

```
gcuser:~/GCHP$ git checkout 14.0.0
```

Initialize and update all the submodules:

```
gcuser:~/GCHP$ git submodule update --init --recursive
```

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

Create a build directory and **cd** into it. A good name for this directory is `build/`, and you may put this directory anywhere on your system, such as in the top-level of the source code or in your run directory. The GCHP build directory will require 1.4G of storage space. In this guide we put it in 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 common 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 making use of the `CodeDir` symbolic link in the run directory:

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

1.4 4. Compile and install

Compiling GCHP takes about 20 minutes, but it can vary depending on your system. Next, compile GCHP in parallel using as many cores as are available:

```
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**.
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**.

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 18.0.5, or
  - GNU compilers version 8.3
- MPI (Message Passing Interface)
  - OpenMPI 3.0, or
  - IntelMPI, or
  - MVAPICH2, or
  - MPICH, or
  - other MPI libraries might work too
- HDF5
- NetCDF (with C, C++, and Fortran support)
- ESMF version 8.0.0

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 *Build Dependencies*.

2.2 Hardware Requirements

These are GCHP's hardware requirements. Note that high-end HPC infrastructure is not required to use GCHP effectively. Gigabit Ethernet and 2 nodes is enough for returns on performance compared to GEOS-Chem Classic.

2.2.1 Recommended Minimum Requirements

These recommended minimums are adequate to effectively use GCHP in scientific applications:

- 2 nodes, preferably 24 cores per node
- Gigabit Ethernet (GbE) interconnect or better
- 100 GB memory per node
- 1 TB of storage

2.2.2 Bare Minimum Requirements

These bare minimum requirements are sufficient for running GCHP at C24. The are adequate for trying GCHP out, and for learning purposes.

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

2.2.3 Big Compute Recommendations

These hardware recommendations are for users that are interested in tackling large bleeding-edge computational problems:

- A high-performance-computing cluster (or a cloud-HPC service like AWS)
  - 1–50 nodes
  - >24 cores per node (the more the better), preferably Intel Xeon
  - High throughput and low-latency interconnect, preferably InfiniBand if using 500 cores
- Lots of storage. Several TB is sufficient, but tens or hundreds of TB is better.

2.2.4 General Hardware and Software Recommendations

- Hyper-threading may improve simulation throughput, particularly at low core counts
- MPI process should be bound sequentially across cores and nodes (e.g., a simulation with 48-processes with 24 processes per node should bind rank 0 to CPU L#0, rank 1 to CPU L#1, etc. on the first node, and rank 24 to CPU L#0, rank 1 to CPU L#1, etc. 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 setting `WRITE_RESTART_BY_OSERVER: YES` in config file `GCHP.rc`. This enables the MAPL o-server functionality for writing restart files, thereby speeding up the mdoel. This is set automatically when executing `setCommonRunSettings.sh`.

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 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`**. To finish retrieving the GCHP source code, initialize and update the submodules:

```
gcuser:~$ cd GCHP
gcuser:~/GCHP$ git submodule update --init --recursive
```

By default, the source code will be on the `main` branch which is always the last offocial 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. Checkout the release that you want to work with, and update the submodules:

```
gcuser:~/GCHP$ git checkout 13.3.4
gcuser:~/GCHP$ git submodule update --init --recursive
```

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 13.3.4
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/disable the RRTMG component.

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 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
```

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.

```
------------------------------------------------------------
Choose meteorology source:
------------------------------------------------------------
  1. MERRA2 (Recommended)
  2. GEOS-FP
```

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:
------------------------------------------------------------
```

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).

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.

The following two parameters are set at the top of the file:

Ext_AllowExtrat Logical toggle to use data from nearest year available. This is set to true for GCHP. Note that GEOS-Chem Classic accomplishes the same effect but with more flexibility in `HEMCO_Config.rc`. That functionality of `HEMCO_Config.rc` is ignored in GCHP.

DEBUG_LEVEL Turns MAPL ExtData debug prints on/off. This is set to 0 in GCHP (off), but may be set to 1 to enable. Beware that turning on ExtData debug prints greatly slows down the model, and prints are only done from the root thread. Use this when debugging problems with input files.

The rest of the file contains space-delimited lines, one for each variable imported to the model from an external file. Columns are as follows in order as they appear left to right in the file:

Export Name Name of imported met field (e.g. ALBD) or HEMCO emissions container name (e.g. GEIA_NH3_ANTH).

Units Unit string nested within single quotes. '1' indicates there is no unit conversion from the native units in the netCDF file.

Clim Enter Y if the file is a 12 month climatology, otherwise enter N. If you specify it is a climatology ExtData the data can be on either one file or 12 files if they are templated appropriately with one per month.

Conservative Enter Y the data should be regridded in a mass conserving fashion through a tile file. `F;{VALUE}` can also be used for fractional regridding. Otherwise enter N to use the non-conervative bilinear regridding.

Refresh Time Template Possible values include:

- -: The field will only be updated once the first time ExtData runs
- 0: Update the variable at every step. ExtData will do a linear interpolation to the current time using the available data.
- %y4-%m2-%h2T%h2:%n2:00: Set the recurring time to update the file. The file will be updated when the evaluated template changes. For example, a template in the form %y4-%m2-%d2T12:00:00 will cause the variable to be updated at the start of a new day (i.e. when the clock hits 2007-08-02T00:00:00 it will update the variable but the time it will use for reading and interpolation is 2007-08-02T12:00:00).

Offset Factor Factor the variable will be shifted by. Use none for no shifting.

Scale Factor Factor the variable will be scaled by. Use none for no scaling.

External File Variable The name of the variable in the netCDF data file, e.g. ALBEDO in met fields.

External File Template Path to the netCDF data file. If not using the data, specify `/dev/null` to reduce processing time. If there are no tokens in the template name ExtData will assume that all the data is on one file. Note that if the data on file is at a different resolution that the application grid, the underlying I/O library ExtData uses will regrid the data to the application grid.

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 `geoschem_config.yml` file wiki page 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 you 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.

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
   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
   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

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 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.

10.3.3 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.

10.3.4 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.

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.

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

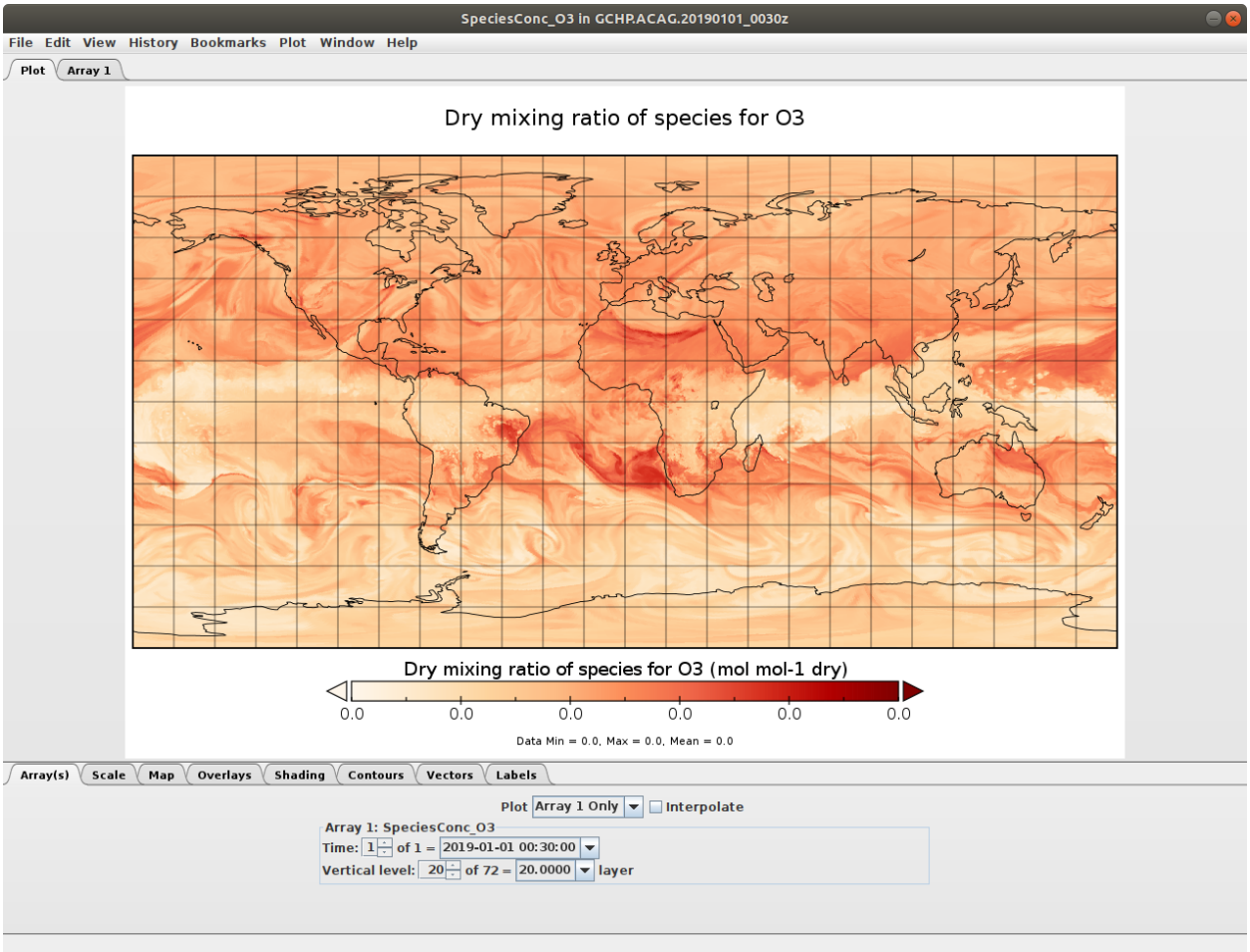
TWELVE

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

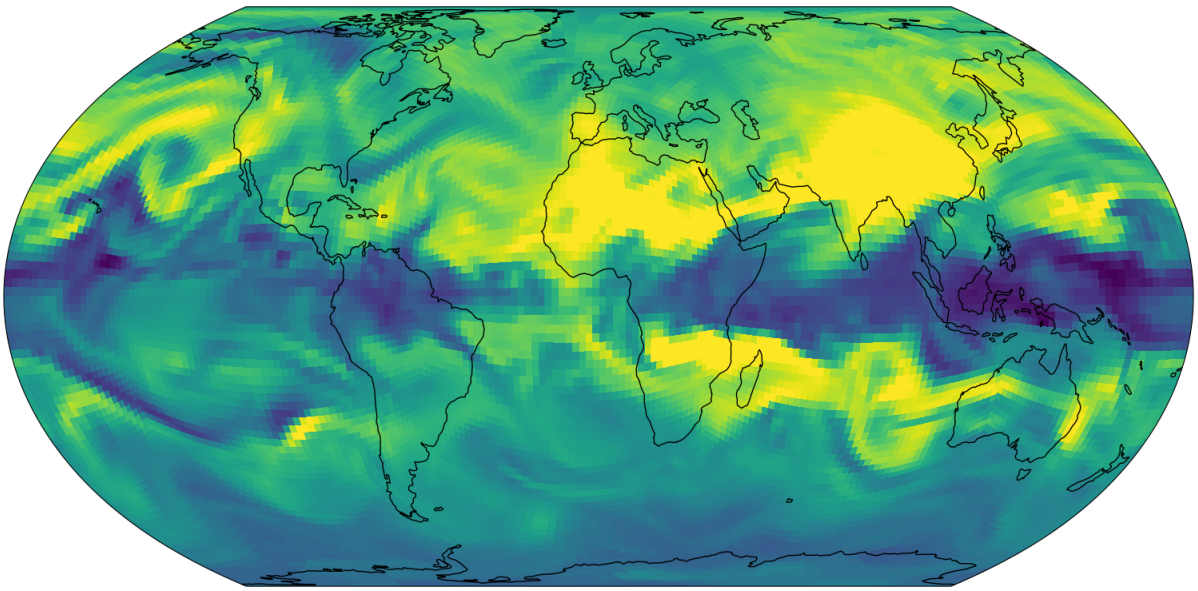
If you use conda you can install these packages like so

```
$ 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.

Table of contents

13.1 Configure errors

Coming soon

13.2 Build-time errors

Coming soon

13.3 Run-time errors

13.3.1 Recompile with debug flags

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 there may be more information in the logs when you run again.

13.3.2 Enable maximum print output

Besides compiling with `CMAKE_BUILD_TYPE=Debug`, there are a few run-time settings you can configure to boost your chance of successful debugging. All of them involve sending additional print statements to the log files.

1. Set Turn on debug printout? in `geoschem_config.yml` to T to turn on extra GEOS-Chem print statements in the main log file.
2. Set the Verbose and Warnings settings in `HEMCO_Config.rc` to maximum values of 3 to send the maximum number of prints to `HEMCO.log`.
3. Set `CAP.EXTDATA` and `MAPL` options `root_level` in `logging.yml` to `DEBUG` to send root thread MAPL prints to `allPEs.log`.
4. Set `CAP.EXTDATA` and `MAPL` option `level` in `logging.yml` to `DEBUG` to send all thread MAPL ExtData (input) prints to `allPEs.log`.

None of these options require recompiling. Be aware that all of them will slow down your simulation. Be sure to set them back to the default values after you are finished debugging.

13.3.3 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.

CHAPTER

FOURTEEN

BUILD DEPENDENCIES

This page has instructions for building GCHP's *dependencies*. These are the software libraries that are needed to compile and execute the GCHP program. These instructions are meant for users that are working on a cluster where GCHP's *Software Requirements* are not already available.

Note: This is not the only way to build the GCHP 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.

The general workflow is the following:

1. Install Spack and perform first-time setup
2. Install the recommended compiler
3. Build GCHP's dependencies
4. Generate a load script (a script that loads the GCHP dependencies in your environment)

14.1 1. Install Spack and do first-time setup

Decide where you want to install Spack. A few details you should consider are:

- this directory will be ~5-20 GB (keep in mind that some clusters limit `$HOME` to a few GB)
- this directory cannot be moved (needs redo if you need to move it in the future)
- if other people are going to use these dependencies, this directory should be in a shared location

Once you choose an install location, proceed with the commands below. You can copy-paste these commands, but lookout for lines marked with a `# (modifiable) ...` comment as they might require modification.

Important: All commands in this tutorial are executed in the same directory.

Install spack and perform the following first-time setup.

```
$ 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  # find software that is already available
```

Next, download a copy of the GCHP source code. The GCHP source code has a `spack/` subdirectory with important Spack settings. The GCHP version should not matter, but it is good practice to use the latest version.

```
$ git clone https://github.com/geoschem/GCHP.git  # we need the GCHP/spack␣
↪subdirectory
```

14.2 2. Install the recommended compiler

Next, install the recommended compiler, `intel-oneapi-compilers`. Note the `-C GCHP/spack` argument—this specifies custom Spack setting for GCHP.

```
$ spack -C GCHP/spack install intel-oneapi-compilers  # install the recommended␣
↪compiler
```

This should take a few minutes. Once the package is installed, add it as a compiler.

```
$ spack compiler add $(spack location -i intel-oneapi-compilers)/compiler/latest/
↪linux/bin/intel64  # register the compiler with spack
```

Note: You can run the command `spack find` to list all the packages that are installed.

You can run the command `spack compiler list` to list the registered compilers. After the **`spack compiler add`** command above, you should see a compiler named `intel@XXXX.XX`, where `XXXX.XX` is the compiler version.

14.3 3. Build GCHP's dependencies

The next step is building the GCHP dependencies. This will be done a **`spack install`** command, which has the following syntax.

```
spack <scope-arguments> install <install-spec>
```

`<scope-arguments>` is a placeholder for arguments like `-C GCHP/spack`, which configures recommended Spack settings for use with GCHP. `<install-spec>` is a placeholder for arguments that specify what package to install.

To install the GCHP dependencies, choose one of the following for `<install-spec>`:

- `esmf%intel ^intel-oneapi-mpi` - **(Recommended)** Default GCHP dependencies, using Intel compilers and Intel MPI.
- `esmf%intel ^openmpi` - Default GCHP dependencies, using Intel compilers and OpenMPI.

For `<scope-arguments>`, you should always include `-C GCHP/spack`. This configures settings for the GCHP dependencies. Note that `GCHP/spack` has subdirectories with platform-specific settings for certain platforms (e.g., AWS ParallelCluster). Check to see if any subdirectories look relevant to you.

The remainder of these instructions use AWS ParallelCluster as an example, so the commands use `-C GCHP/spack -C GCHP/spack/aws-parallelcluster-3.0.1` for `<scope-arguments>`. If no subdirectories are relevant to you, just use `-C GCHP/spack`.

Note: You can see that packages that will be installed with the **`spack spec`** command. For example,

```
$ scope_args="-C GCHP/spack -C GCHP/spack/aws-parallelcluster-3.0.1"  # (modifiable)␣
↪see description of <scope-arguments>
$ install_spec="esmf%intel ^intel-oneapi-mpi"  # (modifiable) see description of
↪<install-spec>
$ spack ${scope_args} spec -I ${install_spec}
Input spec
--------------------------------
 -   esmf%intel

Concretized
--------------------------------
 -   esmf@8.0.1%intel@2021.5.0~debug~external-lapack+mpi+netcdf~pio~pnetcdf~xerces␣
↪arch=linux-amzn2-x86_64
 -       ^intel-oneapi-mpi@2021.5.1%gcc@7.3.1+external-libfabric~ilp64 arch=linux-
↪amzn2-x86_64
 -           ^libfabric@1.13.0%gcc@7.3.1~debug~kdreg fabrics=efa,mrail,rxd,rxm,shm,
↪sockets,tcp,udp arch=linux-amzn2-x86_64
 -       ^libxml2@2.9.12%intel@2021.5.0~python arch=linux-amzn2-x86_64
 -           ^libiconv@1.16%intel@2021.5.0 libs=shared,static arch=linux-amzn2-x86_64
 -           ^pkgconf@1.8.0%intel@2021.5.0 arch=linux-amzn2-x86_64
 -           ^xz@5.2.5%intel@2021.5.0~pic libs=shared,static arch=linux-amzn2-x86_64
 -           ^zlib@1.2.11%intel@2021.5.0+optimize+pic+shared arch=linux-amzn2-x86_64
 -       ^netcdf-c@4.8.1%intel@2021.5.0~dap~fsync~hdf4~jna~mpi~parallel-
↪netcdf+pic+shared arch=linux-amzn2-x86_64
 -           ^hdf5@1.12.1%intel@2021.5.0~cxx~fortran+hl~ipo~java~mpi+shared~szip~
↪threadsafe+tools api=default build_type=RelWithDebInfo patches=ee351eb arch=linux-
↪amzn2-x86_64
 -               ^cmake@3.22.2%intel@2021.5.0~doc~ncurses+openssl+ownlibs~qt build_
↪type=Release arch=linux-amzn2-x86_64
 -                   ^openssl@1.0.2k-fips%intel@2021.5.0~docs certs=system arch=linux-
↪amzn2-x86_64
 -           ^m4@1.4.16%intel@2021.5.0+sigsegv arch=linux-amzn2-x86_64
 -       ^netcdf-fortran@4.5.3%intel@2021.5.0~doc+pic+shared arch=linux-amzn2-x86_64
```

The **`spack spec`** command is not necessary, but it can be helpful to see exactly what packages will be installed.

The following commands build the GCHP dependencies. Note that this may take several hours.

```
$ scope_args="-C GCHP/spack -C GCHP/spack/aws-parallelcluster-3.0.1" # (modifiable)␣
↪see description of <scope-arguments>
$ install_spec="esmf%intel ^intel-oneapi-mpi"  # (modifiable) see description of
↪<install-spec>
$ spack ${scope_args} install ${install_spec}
```

14.4 4. Generate a load script

The last step is generating a script that loads the these dependencies. This is a file that you will `source` before you build or run GCHP. The following commands generate a script called `geoschem_deps-YYYY.MM` where `YYYY.MM` is the current year and month.

```
$ load_script_name="geoschem_deps-$(date +%Y.%m)"  # (modifiable) rename if you want␣
↪to
$ spack ${scope_args} module tcl refresh -y  # regenerate all the modulefiles
```

(continues on next page)

(continued from previous page)

```
$ spack ${scope_args} module tcl loads -r -p $(pwd)/spack/share/spack/modules/linux-*-
↪x86_64/ intel-oneapi-compilers cmake > ${load_script_name}
$ spack ${scope_args} module tcl loads -r -p $(pwd)/spack/share/spack/modules/linux-*-
↪x86_64/ ${install_spec} >> ${load_script_name}
```

For me, this generated a load script named `geoschem_deps-2022.03`. In terminals or scripts you can load the GCHP dependencies by running:

```
$ source /YOUR_PATH_TO/geoschem_deps-2022.03  # loads the the dependencies (replace␣
↪YOUR_PATH_TO)
```

You can copy or move the load script to other directories. At this point, you can remove the `GCHP` directory as it is not needed. The `spack` directory needs to remain.

CHAPTER

FIFTEEN

SET UP AWS PARALLELCLUSTER

Important: AWS ParallelCluster and FSx for Lustre costs several hundred dollars per month to use. See FSx for Lustre Pricing and EC2 Pricing for details.

This page has instructions on setting up AWS ParallelCluster for running GCHP simulations. 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, so these instructions focus on AWS ParallelCluster setup, and the other GCHP documentation like *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 is

1. Create an FSx for Lustre file system (*described on this page*)
2. Configure AWS CLI (*described on this page*)
3. Configure AWS ParallelCluster (*described on this page*)
4. *Build GCHP's dependencies* on your AWS ParallelCluster
5. Follow the normal GCHP User Guide
    a. *Download the model*
    b. *Compile*
    c. *Create a Run Directory*
    d. *Download Input Data*
    e. *Run the model*

These instructions were written using AWS ParallelCluster 3.0.1.

15.1 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*.

15.2 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.

15.3 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
```

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 the configuration file with setting for a cluster. Before starting your cluster with the **`pcluster create-cluster`** command, you 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
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 30 minutes or 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. Your next steps will be *Build Dependencies* followed by the normal instructions found in the User Guide.

CHAPTER

SIXTEEN

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).

16.1 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.

16.2 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.

16.3 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
```

16.4 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.

16.5 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.

16.6 See Also

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

CHAPTER

SEVENTEEN

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.

17.1 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

The current images use OpenMPI 4.0.1 internally, which has been confirmed to work with external installations of OpenMPI 4.0.2-4.0.5.

17.2 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.

17.3 Setting up and running GCHP using Singularity

Available GCHP images are listed on Docker Hub. The following command pulls the image of GCHP 13.0.2 and converts it to a Singularity image named *gchp.sif* in your current directory.

```
$ singularity pull gchp.sif docker://geoschem/gchp:13.0.2
```

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 /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*.

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
$ ./setEnvironment.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 ". /init.rc" > ./internal_exec
$ 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.

17.4 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

EIGHTEEN

STRETCHED-GRID SIMULATION

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

18.1 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.

18.1.1 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.). Remeber that the resolution of the target face is enhanced by approximately the stretch-factor.

18.1.2 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 with GCPy's **`gcpy.file_regrid`** program. Below is an example of regridding a C90 cubed-sphere restart file to a C48 stretched-grid with a stretch factor of 3, a target longitude of 260.0, and a target latitude of 40.0. See the GCPy documentation for this program's exact usage, and for installation instructions.

```
$ python -m gcpy.file_regrid                                    \
               -i GEOSChem.Restart.20190701_0000z.c90.nc4    \
               --dim_format_in checkpoint                       \
               -o sg_restart_c48_3_260_40.nc                    \
               --cs_res_out 48                                  \
               --sg_params_out 3.0 260.0 40.0                   \
               --dim_format_out checkpoint
```

Description of arguments:

`-i` `GEOSChem.Restart.20190701_0000z.c90.nc`
: Specifies the input restart file is `GEOSChem.Restart.20190701_0000z.c90.nc4` (in the current working directory).

`--dim_format_in` `checkpoint`
: Specifies that the input file is in the "checkpoint" format. GCHP restart files use the "checkpoint" format.

`-o` `sg_restart_c48_3_260_40.nc`
: Specifies that the output file should be named `sg_restart_c48_3_260_40.nc`.

`--cs_res_out` `48`
: Specifies that the output grid has a cubed-sphere size 48 (C48).

`--sg_params_out` `3.0 260.0 40.0`
: Specifies that the output grid's stretched-grid parameters in the order stretch factor (3.0), target longitude (260.0), target latitude (40.0).

`--dim_format_out` `checkpoint`
: Specifies that the output file should be in the "checkpoint" format. GCHP restart files must be in the "checkpoint" format.

Once you have created a restart file for your simulation, you can move on to updating your simulation's configuration files.

18.1.3 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
```

(continues on next page)

(continued from previous page)

```
#-----------------------------------------------
# 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 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. Update `cap_restart` to match the date of your restart file. This will also be the start date of the run. 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 so it is a good idea to symbolically link to the original if you want to preserve the original file's specification of stretched grid in its name. 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 a pre-run step in all example run scripts provided in `runScriptSamples`.

18.2 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.

18.2.1 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 python packages GCPy >= 1.0.0 and cartopy >= 0.19

18.2.2 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.

18.2.3 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. Using one of the initial restart files that comes with the GCHP run directory is handy.

```
$ python -m gcpy.file_regrid                              \
     -i GEOSChem.Restart.20190701_0000z.c48.nc4      \
     --dim_format_in checkpoint                           \
     --dim_format_out checkpoint                          \
     --cs_res_out 60                                      \
     --sg_params_out 3.6 275 37                           \
     -o initial_GEOSChem_rst.EasternUS_SG_fullchem.nc
```

This creates `initial_GEOSChem_rst.EasternUS_SG_fullchem.nc`, which is the new restart file for your simulation.

Note: Regridding a C48 files using GCPy takes about a minute to run. If you regrid an even larger restart file (e.g., C180) it may take significantly longer.

18.2.4 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 $1.5\times$ that of a C48 or 2°x2.5° simulation.
- Change global grid resolutio 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. 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.

18.2.5 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 schedule.

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

Log output of the run should be printed to both screen and log file `gchp.20190701_000000z.log`. Check that your run was successful by inspecting the log and looking for output in the `OutputDir` subdirectory.

18.2.6 Plot the output

Append grid-box corners:

```
$ python -m gcpy.append_grid_corners \
     --sg_params 3.6 275 37 \
     OutputDir/GCHP.SpeciesConc.20190707_1200z.nc4
```

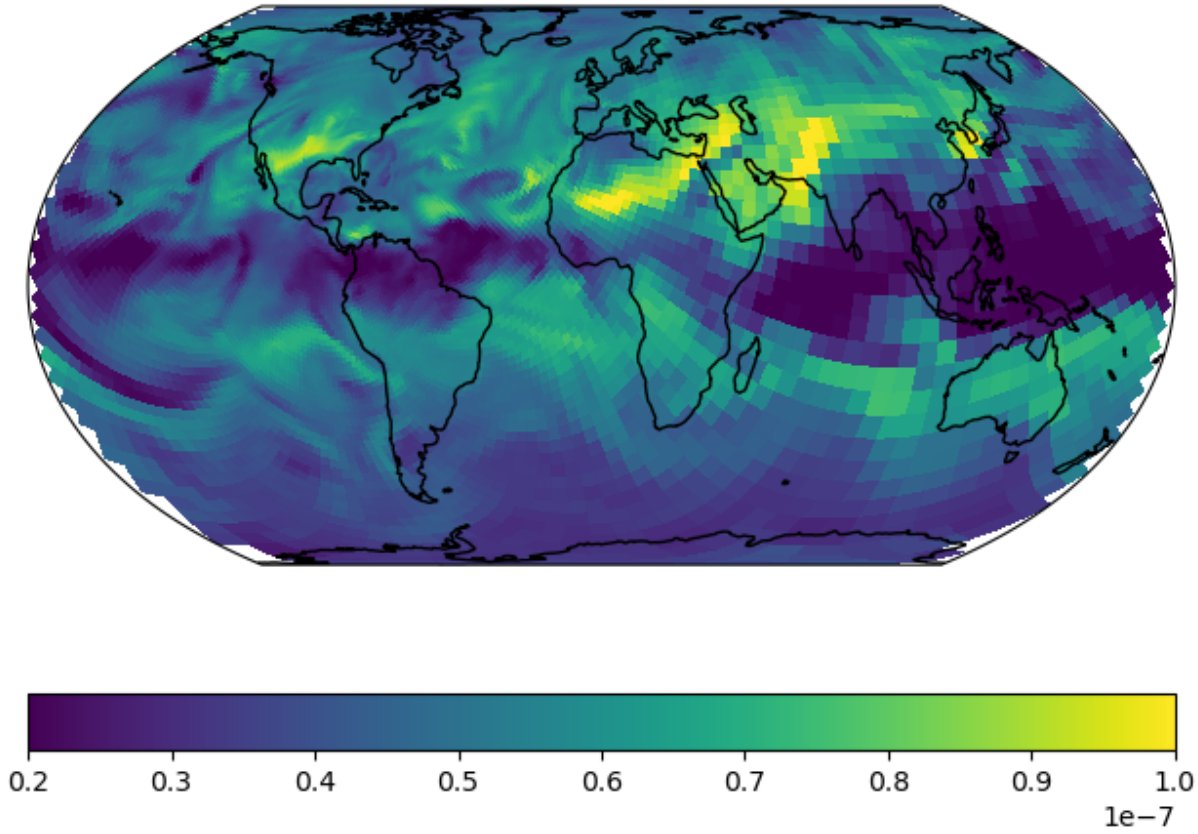
Plot ozone at model level 22:

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

# Load 24-hr average concentrations for 2019-07-07
ds = xr.open_dataset('GCHP.SpeciesConc.20190707_1200z.nc4')

# Get Ozone at level 22
ozone_data = ds['SpeciesConc_O3'].isel(time=0, lev=22).squeeze()

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

(continues on next page)

(continued from previous page)

```
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()
```



CHAPTER

NINETEEN

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.

19.1 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.

19.2 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',
::
```

19.3 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

TWENTY

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.

20.1 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.

20.2 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

TWENTYONE

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.

21.1 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.

21.2 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

21.3 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.

21.4 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.

21.5 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.

21.6 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).

21.7 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.

21.8 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.

21.9 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.

21.10 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`.

21.11 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 )
   ```

21.12 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.

21.13 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

TWENTYTWO

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

22.1 Species properties defined

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

22.1.1 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.

22.1.2 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`).

22.1.3 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.

Radius
Radius (m) of the species. Typically defined only for aerosols.

22.1.4 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.

22.1.5 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.

22.1.6 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.

22.2 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.

22.2.1 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 spceies index
`KppSpcId`	KPP species index
`PhotolId`	Photolyis species index
`RadNuclId`	Radionuclide index
`WetDepId`	Wet deposition index

22.2.2 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
```

22.2.3 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
```

22.2.4 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
```

(continues on next page)

(continued from previous page)

```
   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' )

! 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.
```

(continues on next page)

(continued from previous page)

```
  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
```

22.2.5 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
```

(continues on next page)

(continued from previous page)

```
      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

      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

TWENTYTHREE

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:

23.1 Using KPP: Quick start

23.1.1 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
```

23.1.2 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`.

23.1.3 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 172 reactions
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.map             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.map` 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.

23.1.4 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:
   * CUSTOMMECH:  fullchem  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.

23.2 Using KPP: Reference section

23.2.1 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.

23.2.2 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...
//
```

(continues on next page)

(continued from previous page)

```
O3 + NO = NO2 + O2 :                    GCARR(3.00E-12, 0.0, -1500.0);
O3 + OH = HO2 + O2 :                    GCARR(1.70E-12, 0.0, -940.0);
O3 + HO2 = OH + O2 + O2 :               GCARR(1.00E-14, 0.0, -490.0);
O3 + NO2 = O2 + NO3 :                   GCARR(1.20E-13, 0.0, -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`.

23.2.3 Heterogeneous reactions

TODO Remove reference to HET array

List heterogeneous reactions after all of the gas-phase reactions in `custom.eqn`, according to the format below:

```
//
// Heterogeneous reactions
//
HO2 = O2 :                                    HET(ind_HO2,1);
↪{2013/03/22; Paulot2009; FP,EAM,JMAO,MJE}
NO2 = 0.500HNO3 + 0.500HNO2 :                 HET(ind_NO2,1);
NO3 = HNO3 :                                  HET(ind_NO3,1);
NO3 = NIT :                                   HET(ind_NO3,2);
↪{2018/03/16; XW}
... etc ...
```

Implementing new heterogeneous chemistry requires an additional step. For the reaction in question, a reaction should be added as usual, but this time the rate function should be given as an entry in the `HET` array. A simple example is uptake of HO2, specified as

```
HO2 = O2 : HET(ind_HO2,1);
```

Note that the product in this case, O2, is actually a fixed species, so no O2 will actually be produced. O2 is used in this case only as a dummy product to satisfy the KPP requirement that all reactions have at least one product. Here, `HET` is simply an array of pre-calculated rate constants. The rate constants in `HET` are actually calculated in `gckpp_HetRates.F90`.

To implement an additional heterogeneous reaction, the rate calculation must be added to this file. The following example illustrates a (fictional) heterogeneous mechanism which converts the species XYZ into CH2O. This reaction is assumed to take place on the surface of all aerosols, but not cloud droplets (this requires additional steps not shown here). Three steps would be required:

1. Add a new line to the `custom.eqn` file, such as `XYZ = CH2O : HET(ind_XYZ,1);`
2. Add a new function to `gckpp_HetRates.F90` designed to calculate the heterogeneous reaction rate. As a simple example, we can copy the function `HETNO3` and rename it `HETXYZ`. This function accepts two arguments: molecular mass of the impinging gas-phase species, in this case XYZ, and the reaction's "sticking coefficient" - the probability that an incoming molecule will stick to the surface and undergo the reaction in question. In the case of `HETNO3`, it is assumed that all aerosols will have the same sticking coefficient, and the function returns a first-order rate constant based on the total available aerosol surface area and the frequency of collisions
3. Add a new line to the function `SET_HET` in `gckpp_HetRates.F90` which calls the new function with the appropriate arguments and passes the calculated constant to `HET`. Example: assuming a molar mass of 93 g/mol, and a sticking coefficient of 0.2, we would write `HET(ind_XYZ, 1) = HETXYZ(93.0_fp, 0.2_fp)`

The function `HETXYZ` can then be specialized to distinguish between aerosol types, or extended to provide a second-order reaction rate, or whatever the user desires.

23.2.4 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 ...
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 :file:`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.

23.2.5 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 `gckpp.kpp` file:

```
#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.

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 KPP.

```
// - 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.
//
//   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.

23.2.6 Changing the numerical integrator

Several global options for **KPP** are listed at the top of the `gckpp.kpp` file:

```
#MINVERSION   2.5.0
#INTEGRATOR   rosenbrock
#LANGUAGE     Fortran90
#UPPERCASEF90 on
#DRIVER       none
#HESSIAN      off
#MEX          off
#STOICMAT     off
```

The #INTEGRATOR tag specifies the choice of numerical integrator that you wish to use with your chemical mechanism. The Rosenbrock solver is used by default for the GEOS-Chem **fullchem** and **Hg** mechanisms. But 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 2.5.0. 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

TWENTYFOUR

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/c/geoschem

Table 2: **User manuals for GEOS-Chem and related software**

Software	Documentation
GEOS-Chem Classic	geos-chem.readthedocs.io
GCHP	gchp.readthedocs.io
HEMCO	hemco.readthedocs.io
GEOS-Chem on the cloud	geos-chem-cloud.readthedocs.io
WRF-GC (GEOS-Chem in WRF)	wrf.geos-chem.org
GCPy (Python toolkit)	gcpy.readthedocs.io
KPP (The Kinetic PreProcessor)	kpp.readthedocs.io
IMI (Integrated Methane Inversion)	imi.readthedocs.io
CHEEREIO (Data assimilation & emissions inversions)	cheereio.readthedocs.io

CHAPTER

TWENTYFIVE

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.

25.1 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.

25.2 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.

25.3 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).

25.4 How to submit changes

Please see "Contributing Guidelines".

25.5 How to request an enhancement

Please see "Contributing Guidelines".

CHAPTER

TWENTYSIX

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.

26.1 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.

26.2 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.

26.3 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

26.3.1 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.

26.3.2 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.

26.3.3 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.

26.3.4 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.

26.4 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.

26.5 How can I report a bug?

Please see "Support Guidelines".

26.6 Where can I ask for help?

Please see "Support Guidelines".

CHAPTER

TWENTYSEVEN

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.

27.1 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`.

27.2 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.

27.3 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

TWENTYEIGHT

GIT SUBMODULES

28.1 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

TWENTYNINE

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

THIRTY

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

THIRTYONE

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., 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.

[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