GCHP

Release 13.0.0-alpha.10

GEOS-Chem Support Team

Dec 17, 2020

GETTING STARTED

Important: This is a prerelease of the GEOS-Chem High Performance user guide. These pages are the most up-to-date and accurate instructions for GCHP, but they are still a work in progress.

Contributions (e.g., suggestions, edits, revisions) would be greatly appreciated. See *editing this guide* and our *contributing guidelines*. If you find a something hard to understand—let us know!

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

CHAPTER

ONE

QUICK START

This quickstart guide assumes your environment satisfies *GCHP's requirements*. This means you should load a compute environment such that programs like **`cmake`** and **`mpirun`** are available, before continuing. You can find more detailed instructions 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 13.0.0-beta.1
```

Note: Version 13 is not officially released yet. Until then, the most recent commit to `main` is the most stable version of GCHP. Therefore, we recommend you checkout `main`, rather than a version like `13.0.0-beta.1`. E.g.:

```
$ git checkout main    # recommended until version 13 is officially released
```

Once version 13 is released, we will resume recommending users checkout a specific version.

Initialize and update all submodules:

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

1.2 2. Create a run directory

Navigate to the `run/` subdirectory. Create a run directory by running `./createRunDir.sh` and answering 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 a good place for it is in the top-level of the source code:

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

Initialize your build directory by running **cmake** and passing it the path to your source code:

```
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 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.

1.4 4. Compile and install

Compiling GCHP takes about 20 minutes, but it can varry depending on your system. Next, compile GCHP:

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

Next, 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**. See
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
```

Most simulation settings are configured in `./runConfig.sh`. You should review this file as it explains how to configure most simulation settings. Note that `./runConfig.sh` is actually a helper script that updates other configuration files. Therefore, you need to run it to actually apply the updates:

```
$ vim runConfig.sh                # edit simulation settings here
$ ./runConfig.sh                  # applies the updated 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 you use to launch the **`gchp`** executable that is in your run directory. You'll need 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 will write a script that runs GCHP, and then you will submit that script to your scheduler.

Note: When GCHP runs, partially or to completion, it generates several files including `cap_restart` and `gcchem_internal_checkpoint`. Subsequent runs won't overwrite these files, and instead the run will exit with an error. Because of this it is common to do

```
$ rm -f cap_restart gcchem_internal_checkpoint
```

before starting a GCHP simulation.

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 use GCHP you will need a compute environment with the following:

- 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
- NetCDF (with C, C++, and Fortran support)
- ESMF version 8.0.0

Your system administrator can tell you what software is available on your cluster and how you access it. If you need to install any of these software yourself, you can do that manually (build from source), but a faster and easier way to do it is with Spack. See our guide on *installing GCHP's dependencies with Spack*.

CHAPTER

THREE

KEY REFERENCES

- GEOS-Chem was first described in [Bey et al., 2001].
- HEMCO is described in [Keller et al., 2014].
- 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., 2020].

References

CHAPTER

FOUR

DOWNLOADING GCHP

Clone the GCHP repository from GitHub:

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

Next, update the submodules. These are other repositories that are nested inside the GCHP repository. Initialize the submodules:

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

By default, your source code will be on the `main` branch. It is a good idea to checkout an official release rather than use the `main` branch. You can find the list of GCHP releases here. Checkout the version that you want to work with, and update your submodules:

```
gcuser:~/Code.GCHP$ git checkout 13.0.0-beta.1
gcuser:~/Code.GCHP$ git submodule update --init --recursive
```

Note: Version 13 is not officially released yet. Until then, the most recent commit to `main` is the most stable version of GCHP. Therefore, we recommend you checkout `main`, rather than a version like `13.0.0-beta.1`. E.g.:

```
$ git checkout main    # recommended until version 13 is officially released
```

Once version 13 is released, we will resume recommending users checkout a specific version.

CHAPTER

FIVE

COMPILING GCHP

Note: This user guide assumes you have loaded a computing environment that satisfies *GCHP's software requirements*.

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

There are two steps to build GCHP. The first step is configuring your build. To configure your build you use **`cmake`** to configure build settings. Build settings cover options like enabling or disabling components like RRTMG, specifying run directories to install GCHP to, or whether GCHP should be compiled in Debug mode.

The second step is compiling. To compile GCHP you use **`make`**. This compiles GCHP according to your configuration from the first step.

5.1 Create a build directory

Create a build directory. This directory is going to be the working directory for your build. The configuration and compile steps generate a bunch of build files, and this directory is going to store those. You can think of a build directory as representing a GCHP build. It stores configuration settings, information about your system, and intermediate files from the compiler.

A build directory is self contained, so you can delete it at any point to erase the build and its configuration. You can have as many build directories as you would like. Most users only need one build directory, since they only build GCHP once; but, for example, if you were building GCHP with Intel and GNU compilers to compare performance, you would have two build directories: one for the Intel build, and one for the GNU build. You can name your build directories whatever you want, but a good choice is `build/`. There is one rule for build directories: **a build directory should be a new directory**.

Create a build directory and initialize it. You initialize a build directory by running **`cmake`** with the path to the GCHP source code. Here is an example of creating a build directory in the top-level of the GCHP source code:

```
gcuser:~$ cd ~/GCHP.Code
gcuser:~/GCHP.Code$ mkdir build                # create the build dir
gcuser:~/Code.GCHP$ cd build
gcuser:~/Code.GCHP/build$ cmake ~/Code.GCHP    # initialize the build
-- 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
```

(continues on next page)

(continued from previous page)

```
...
-- Configuring done
-- Generating done
-- Build files have been written to: /src/build
gcuser:~/Code.GCHP/build$
```

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 on your system. You can add custom paths to CMake's default list of search paths with the `CMAKE_PREFIX_PATH` variable.

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

```
gcuser:~/Code.GCHP/build$ cmake . -DCMAKE_PREFIX_PATH=/software/ESMF
```

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

5.2 Configure your build

Build settings are controlled by **`cmake`** commands with the following form:

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

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

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

GCHP has no required build settings. You can find the complete list of *GCHP's build settings here*. The most frequently used build 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 directories.

Note: You can even 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, for this tutorial 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 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
```

Optionally, you can use the `VERBOSE=1` argument to see the compiler commands.

This step creates `./bin/gchp` which is the compiled executable. You can copy this executable to your run directory manually, or you can do

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

which copies `./bin/gchp` (and some supplemental files) to the run directories specified in *RUNDIR*.

Now you have compiled GCHP, and you are ready to move on to creating a run directory!

5.4 Recompiling

You need to recompile GCHP if you update a build setting or make a modification to the source code. However, with CMake, you don't need to clean before recompiling. The build system automatically figure out which files need to be recompiled based on your modification. This is known as incremental compiling.

To recompile GCHP, simply do

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

and optionally, do **`make install`**.

5.5 GCHP build options

These are persistent build setting that are set with **`cmake`** commands with the following form

```
$ 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 One or more directories that are searched for external libraries like NetCDF or MPI. You can specify multiple paths 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

CREATING A RUN DIRECTORY

GCHP run directories are created from within the source code. A new run directory should be created for each different version of GEOS-Chem you use. Git version information is logged to file `rundir.version` within the run directory upon creation.

To create a run directory, navigate to the `run/` subdirectory of the source code and execute shell script `createRunDir.sh`.

```
gcuser:~$ cd Code.GCHP/run
gcuser:~/Code.GCHP/run$
```

During the course of script execution you will be asked a series of questions:

6.1 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:
------------------------------------------------------------
```

6.2 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
```

6.3 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
```

6.4 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:
------------------------------------------------------------
```

6.5 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:
------------------------------------------------------------
```

6.6 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

RUNNING GCHP

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 at the end to help prevent run errors. The GCHP "standard" simulation run directory is configured for a 1-hr simulation at c24 resolution and is a good first test case to check that GCHP runs on your system.

7.1 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. Source environment file symbolic link `gchp.env` to ensure run environment consistent with build
2. Source config file `runConfig.sh` to set run-time configuration
3. Delete any previous run output files that might interfere with the new run if present
4. Send standard output to run-time log file `gchp.log`
5. Rename the output restart file to include "restart" and datetime

7.1.1 Run interactively

Copy or adapt example run script gchp.local.run to run GCHP locally on your machine. Before running, open your run script and set nCores to the number of processors you plan to use. Make sure you have this number of processors available locally. It must be at least 6. Next, open file runConfig.sh and set NUM_CORES, NUM_NODES, and NUM_CORES_PER_NODE to be consistent with your run script.

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 log file `gchp.log`.

7.1.2 Run as batch job

Batch job run scripts will vary based on what job scheduler you have available. Most of the example run scripts are for use with SLURM, and the most basic example of these is `gchp.run`. You may copy any of the example run scripts to your run directory and adapt 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. For a basic six core standard simulation job on one node you should request at least 20 min and 32GB of memory. The more cores you request the faster GCHP will run.

To submit a batch job using SLURM:

```
$ sbatch gchp.run
```

To submit a batch job using Grid Engine:

```
$ qsub gchp.run
```

Standard output will be sent to log file `gchp.log` once the job is started unless you change that feature of the run script. Standard error will be sent to a file specific to your scheduler, e.g. `slurm-jobid.out` if using SLURM, unless you configure your run script to do otherwise.

If your computational cluster uses a different job scheduler, e.g. Grid Engine, LSF, or PBS, check with your IT staff or search the internet for how to configure and submit batch jobs. For each job scheduler, batch job configurable settings and acceptable formats are available on the internet and are often accessible from the command line. For example, type **`man sbatch`** to scroll through options for SLURM, including various ways of specifying number of cores, time and memory requested.

7.2 Verify a successful run

There are several ways to verify that your run was successful.

1. NetCDF files are present in the `OutputDir/` subdirectory
2. Standard output file `gchp.log` ends with `Model Throughput` timing information
3. The job scheduler 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
- `MAPL_Cap` errors typically indicate an error with your start time, end time, and/or duration set in `runConfig.sh`
- `MAPL_ExtData` errors often indicate an error with your input files specified in either `HEMCO_Config.rc` or `ExtData.rc`
- `MAPL_HistoryGridComp` errors are related to your configured output in `HISTORY.rc`

If you cannot figure out where the problem is please do not hesitate to create a GCHPctm GitHub issue.

7.3 Reuse a run directory

7.3.1 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 in quick succession 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
```

All files are archived to subfolders in the new directory. Which files are copied and to where are displayed on the screen. Diagnostic files in the `OutputDir/` directory are moved rather than copied so as not to duplicate large files. You will be prompted at the command line to accept this change prior to data move.

7.3.2 Clean a run directory

You should always clean your run directory prior to your next run. This avoids confusion about what output was generated when and with what settings. Under certain circumstances it also avoids having your new run crash. GCHP will crash if:

- Output file `cap_restart` is present and you did not change your start/end times
- Your last run failed in such a way that the restart file was not renamed in the post-run commands in the run script

The example run scripts include extra commands to clean the run directory of the two problematic files listed above. However, you may write your own run script and omit them in which case not cleaning the run directory prior to rerun will cause problems.

To make run directory cleaning simple is utility shell script `cleanRunDir.sh`. To clean the run directory simply execute this script.

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

All GCHP output files, including diagnostics files in `OutputDir/`, will then be deleted. Only restart files with names matching `gcchem*` are deleted. This preserve the initial restart symbolic links that come with the run directory.

7.4 Pre-run checklist

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

1. Your run directory contains the executable `gchp`.
2. All symbolic links in your run directory are valid (no broken links)
3. You have looked through and set all configurable settings in `runConfig.sh`
4. If running via a job scheduler: you have a run script and the resource allocation in `runConfig.sh` and your run script are consistent (# nodes and cores)

5. If running interactively: the resource allocation in `runConfig.sh` is available locally
6. If reusing a run directory (optional but recommended): you have archived your last run with `./archiveRun.sh` if you want to keep it and you have deleted old output files with `./cleanRunDir.sh`

7.5 Recommended MPI configuration

7.5.1 IntelMPI

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

7.5.2 OpenMPI

At high-core counts (e.g., > ~1000 cores) it's recommended to set `WRITE_RESTART_BY_OSERVER: YES` in `GCHP.rc`.

CHAPTER

EIGHT

RUN DIRECTORY CONFIGURATION

All GCHP run directories have default simulation-specific run-time settings that are set when you create a run directory. You will likely want to change these settings. This page goes over how to do this.

Table of contents

8.1 Configuration files

GCHP is controlled using a set of configuration files that are included in the GCHP run directory. Files include:

1. `CAP.rc`
2. `ExtData.rc`
3. `GCHP.rc`
4. `input.geos`
5. `HEMCO_Config.rc`
6. `HEMCO_Diagn.rc`
7. `input.nml`
8. `HISTORY.rc`

Several run-time settings must be set consistently across multiple files. Inconsistencies may result in your program crashing or yielding unexpected results. To avoid mistakes and make run configuration easier, bash shell script `runConfig.sh` is included in all run directories to set the most commonly changed config file settings from one location. Sourcing this script will update multiple config files to use values specified in file.

Sourcing `runConfig.sh` is done automatically prior to running GCHP if using any of the example run scripts, or you can do it at the command line. Information about what settings are changed and in what files are standard output of the script. To source the script, type the following:

```
$ source runConfig.sh
```

You may also use it in silent mode if you wish to update files but not display settings on the screen:

```
$ source runConfig.sh --silent
```

While using `runConfig.sh` to configure common settings makes run configure much simpler, it comes with a major caveat. If you manually edit a config file setting that is also set in `runConfig.sh` then your manual update will be overrided via string replacement. Please get very familiar with the options in `runConfig.sh` and be conscientious about not updating the same setting elsewhere.

You generally will not need to know more about the GCHP configuration files beyond what is listed on this page. However, for a comprehensive description of all configuration files used by GCHP see the last section of this user manual.

8.2 Common options

8.2.1 Compute configuration

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) `runConfig.sh`, and (2) your run script. The `runConfig.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 `runConfig.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 `runConfig.sh` if variable `NXNY_AUTO` is set to `ON`.

Further discussion about domain decomposition is in `runConfig.sh` section `Domain Decomposition`.

Split a simulation into multiple jobs

There is an option to split up a single simulation into separate serial jobs. To use this option, do the following:

1. Update `runConfig.sh` with your full simulation (all runs) start and end dates, and the duration per segment (single run). Also update the number of runs options to reflect to total number of jobs that will be submitted (`NUM_RUNS`). Carefully read the comments in `runConfig.sh` to ensure you understand how it works.
2. Optionally turn on monthly diagnostic (`Monthly_Diag`). Only turn on monthly diagnostics if your run duration is monthly.
3. Use `gchp.multirun.run` as your run script, or adapt it if your cluster does not use SLURM. It is located in the runScriptSamples subdirectory of your run directory. As with the regular `gchp.run`, you will need to update the file with compute resources consistent with `runConfig.sh`. **Note that you should not submit the run script directly**. It will be done automatically by the file described in the next step.
4. Use `gchp.multirun.sh` to submit your job, or adapt it if your cluster does not use SLURM. It is located in the `runScriptSamples/` subdirectory of your run directory. For example, to submit your series of jobs, type: `./gchp.multirun.sh`

There is much documentation in the headers of both `gchp.multirun.run` and `gchp.multirun.sh` that is worth reading and getting familiar with, although not entirely necessary to get the multi-run option working. If you have not done so already, it is worth trying out a simple multi-segmented run of short duration to demonstrate that the multi-segmented run configuration and scripts work on your system. For example, you could do a 3-hour simulation with 1-hour duration and number of runs equal to 3.

The multi-run script assumes use of SLURM, and a separate SLURM log file is created for each run. There is also log file called `multirun.log` with high-level information such as the start, end, duration, and job ids for all jobs

submitted. If a run fails then all scheduled jobs are cancelled and a message about this is sent to that log file. Inspect this and your other log files, as well as output in the `OutputDir/` directory prior to using for longer duration runs.

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 and 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`. The default is set to 20000000.

8.2.2 Basic run settings

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 the `CS_RES` integer parameter in `runConfig.sh` section `Internal Cubed Sphere Resolution` to the cube side length you wish to use. To use a uniform global grid resolution make sure that `STRETCH_GRID` is set to `OFF`.

Set stretching 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 `runConfig.sh` section Internal Cubed Sphere Resolution. See instructions in that section of the file.

Turn on/off model components

You can toggle all primary GEOS-Chem components, including type of mixing, from within `runConfig.sh`. The settings in that file will update `input.geos` automatically. Look for section `Turn Components On/Off`, and other settings in `input.geos`. Other settings in this section beyond component on/off toggles using CH4 emissions in UCX, and initializing stratospheric H2O in UCX.

Change model timestep

Model timesteps, both chemistry and dynamic, are configured within `runConfig.sh`. They are set to match GEOS-Chem Classic default values for low resolutions for comparison purposes but can be updated, with caution. Timesteps are automatically reduced for high resolution runs. Read the documentation in `runConfig.sh` section `Timesteps` for setting them.

Set simulation start and end dates

Set simulation start and end in `runConfig.sh` section `Simulation Start, End, Duration, # runs`. Read the comments in the file for a complete description of the options. Typically a "CAP" runtime error indicates a problem with start, end, and duration settings. If you encounter an error with the words "CAP" near it then double-check that these settings make sense.

8.2.3 Inputs

Change initial restart file

All GCHP run directories come with symbolic links to initial restart files for commonly used cubed sphere resolutions. The appropriate restart file is automatically chosen based on the cubed sphere resolution you set in `runConfig.sh`.

You may overwrite the default restart file with your own by specifying the restart filename in `runConfig.sh` section `Initial Restart File`. Beware that it is your responsibility to make sure it is the proper grid resolution.

Unlike GEOS-Chem Classic, HEMCO restart files are not used in GCHP. HEMCO restart variables may be included in the initial species restart file, or they may be excluded and HEMCO will start with default values. GCHP initial restart files that come with the run directories do not include HEMCO restart variables, but all output restart files do.

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 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.

Add new emissions files

There are two steps for adding new emissions inventories to GCHP:

1. Add the inventory information to `HEMCO_Config.rc`.
2. Add the inventory information to `ExtData.rc`.
3. To add 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.

Staying consistent with the information that you put into `HEMCO_Config.rc`, add the inventory information to `ExtData.rc` following the guidelines listed at the top of the file and using existing inventories as examples. You can ignore all entries in `HEMCO_Config.rc` that are copies of another entry since putting these in `ExtData.rc` would result in reading the same variable in the same file twice. HEMCO interprets the copied variables, denoted by having dashes in the `HEMCO_Config.rc` entry, separate from file read.

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`. Error 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 running with the MAPL debug flags on `runConfig.sh` (maximizes output to `gchp.log`) and Warnings and Verbose set to 3 in `HEMCO_Config.rc` (maximizes output to `HEMCO.log`) when encountering errors such as this. 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.

8.2.4 Outputs

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.

Output restart files at regular or irregular frequency

The MAPL component in GCHP has the option to output restart files (also called checkpoint files) prior to run end. The frequency of restart file write may be at regular time intervals (regular frequency) or at specific programmed times (irregular frequency). These periodic output restart files contain the date and time in their filenames.

Enabling this feature 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 `runConfig.sh` section `Output Restarts`. Instructions for configuring both regular and irregular frequency restart files are included in the file.

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 `runConfig.sh`.

Set diagnostic frequency, duration, and mode

All diagnostic collections that come with the run directory have frequency, duration, and mode auto-set within `runConfig.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. See section `Output Diagnostics` within `runConfig.sh`. To avoid auto-update of a certain collection, remove it from the list in `runConfig.sh`. If adding a new collection, you can add it to the file to enable auto-update of frequency, duration, and mode.

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. See the `HISTORY.rc` file for examples.

Once implemented, you can either incorporate the new collection settings into `runConfig.sh` for auto-update, or you can manually configure all settings in `HISTORY.rc`. See the `Output Diagnostics` section of `runConfig.sh` for more information.

Generate monthly mean diagnostics

There is an option to automatically generate monthly diagnostics by submitting month-long simulations as separate jobs. Splitting up the simulation into separate jobs is a requirement for monthly diagnostics because MAPL History requires a fixed number of hours set for diagnostic frequency and file duration. The monthly mean diagnostic option automatically updates `HISTORY.rc` diagnostic settings each month to reflect the number of days in that month taking into account leap years.

To use the monthly diagnostics option, first read and follow instructions for splitting a simulation into multiple jobs (see separate section on this page). Prior to submitting your run, enable monthly diagnostics in `runConfig.sh` by searching for variable "Monthly_Diag" and changing its value from 0 to 1. Be sure to always start your monthly diagnostic runs on the first day of the month.

8.2.5 Debugging

Enable maximum print output

Besides compiling with `CMAKE_BUILD_TYPE=Debug`, there are a few 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 input.geos to T to turn on extra GEOS-Chem print statements in the main log file.
2. Set `MAPL_EXTDATA_DEBUG_LEVEL` in `runConfig.sh` to 1 to turn on extra MAPL print statements in ExtData, the component that handles input.

3. 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`.

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.

CHAPTER

NINE

CONFIGURATION FILES

GCHP is controlled using a set of resource configuration files that are included in the GCHP run directory, most of which are denoted by suffix `.rc`. These files contain all run-time information required by GCHP. Files include:

- `CAP.rc`
- `ExtData.rc`
- `GCHP.rc`
- `input.geos`
- `HEMCO_Config.rc`
- `HEMCO_Diagn.rc`
- `input.nml`
- `HISTORY.rc`

Much of the labor of updating the configuration files has been eliminated by run directory shell script `runConfig.sh`. It is important to remember that sourcing `runConfig.sh` will overwrite settings in other configuration files, and you therefore should never manually update other configuration files unless you know the specific option is not available in `runConfig.sh`.

That being said, it is still worth understanding the contents of all configuration files and what all run options include. This page details the settings within all configuration files and what they are used for.

9.1 File descriptions

The following table lists the core functions of each of the configuration files in the GCHP run directory. See the individual subsections on each file for additional information.

`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`.

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

input.geos Primary config file for GEOS-Chem. Same function as in GEOS-Chem Classic except grid, simulation start/end, met field source, timers, and data directory information are ignored.

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`.

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 GCHPctm diagnostics listed in `HISTORY.rc` that start with Emis, Hco, or Inv must have a corresponding entry in `HEMCO_Diagn.rc`.

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

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

9.2 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).

9.3 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.

9.4 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.

9.5 input.geos

Information about the `input.geos` file is the same as for GEOS-Chem Classic with a few exceptions. See the `input.geos` file wiki page for an overview of the file.

The `input.geos` 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 `input.geos`.
- 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 `input.geos` file that are not relevant to GCHP are simply not included in the file that is copied to the GCHP run directory.

9.6 HEMCO_Config.rc

Like `input.geos`, 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`.

9.7 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.

9.8 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.

9.9 HISTORY.rc

The `HISTORY.rc` configuration file controls the diagnostic files output by GCHP. Information is organized into several sections:

1. Single parameters set at the top of the file
2. Grid label declaration list
3. Definition for each grid in grid label list

4. Variable collection declaration list
5. Definition for each collection in collection list.

Single parameters set at the top of `HISTORY.rc` are as follows and apply to all collections:

EXPID Filename prefix concatenated with each collection template string to define file path. It is set to OutputDir/GCHP so that all output diagnostic files are output to run subdirectory OutputDir and have filename begin with GCHP.

EXPDSC Export description included as attribute "Title" in output files

CoresPerNode Number of CPUs per node for your simulation.

VERSION Optional parameter included as attribute in output file.

The grid labels section of `HISTORY.rc` declares a list of descriptive grid strings followed by a definition for each declared grid label. Grids not in the grid label list may have definitions in the file; however, this will prevent them from being used in output collections. See the `HISTORY.rc` grid label section for syntax on declaring and defining grid labels.

Keywords that may be used for grid label definitions are in the table below. Note that this list is not exhaustive; MAPL may have additional keywords that may be used that have not yet been explored for use with GCHP.

GRID_TYPE Type of grid. May be Cubed-Sphere or LatLon.

IM_WORLD Side length of one cubed-sphere face, e.g. 24 if grid resolution is C24, or number of longitudinal grid boxes if lat-lon.

JM_WORLD Same as IM_WORLD but multiplied by 6 (number of faces), or number of latitudinal grid boxes if lat-lon.

POLE For lat-lon grids only. PC if latitudes are pole-centered and PE if latitudes are polar edge.

DATELINE For lat-lon grids only. DC if longitudes are dateline-centered and DE if longitudes are dateline-edge.

LAT_RANGE For lat-lon grids only. Regional grid latitudinal bounds.

LON_RANGE For lat-lon grids only. Regional grid longitudinal bounds.

LM Number of vertical levels.

The collections section of `HISTORY.rc` declares a list of descriptive strings that define unique collections of output variables and settings. As with grid labels, it is followed by a definition for each declared collection. Collections not in the collection list, or present but commented out, may have definitions in the file; however, this will prevent them from being output. See the `HISTORY.rc` collection section for syntax on declaring and defining output collections.

Keywords that may be used for collection definitions are in the table below. Note that this list is not exhaustive; MAPL may have additional keywords that may be used that have not yet been explored for use with GCHP.

{COLLECTION}.template The output filename suffix that is appended to global parameter EXPID to define full output file path. Including a date string, such as '%y4%m2%d2, will insert the simulation start day following GrADS conventions. The default template for GCHP is set to %y4%m2%d2_%h2%n2z.nc4.

{COLLECTION}.format Character string defining the file format. Options include CFIO (default) for netCDF-4 or flat for binary files. Always output as CFIO when using GCHP.

{COLLECTION}.grid_label Declared grid label for output grid. If excluded the collection will be output on the cubed-sphere grid at native resolution, e.g. C24 if you run GCHP with grid resolution C24.

{COLLECTION}.conservative For lat-lon grids only. Set to 1 to conservatively regrid from cubed-sphere to lat-lon upon output. Exclude or set to 0 to use bilinear interpolation instead (not recommended).

{COLLECTION}.frequency The frequency at which output values are computed and archived, in format HHMMSS. For example, 010000 will calculate diagnostics every hour. The method of calculation is determined by the mode keyword. Unlike GEOS-Chem Classic, you cannot specify number of days, months, or years.

{COLLECTION}.duration The frequency at which files are written, in format HHMMSS. For example, 240000 will output daily files. The number of times in the file are determined by the frequency keyword. Unlike GEOS-Chem Classic, you cannot specify number of days, months, or years.

{COLLECTION}.mode Method of diagnostic calculation. Options are either instantaneous or time-averaged.

{COLLECTION}.fields List of character string pairs including diagnostic name and gridded component. All GCHP diagnostics belong to the GCHPchem gridded component.

Diagnostic names have prefixes that determine where MAPL History will look for them. Prefixes Emis, Inv, and Hco are HEMCO diagnostics and must have definitions in config file `HEMCO_Diagn.rc`. Prefixes Chem and Met are GEOS-Chem state variables stored in objects State_Chm and State_Met respectively. Prefix SPC corresponds to internal state species, meaning the same arrays that are output to the restart file. Prefix GEOS is reserved for use in the GEOS model. All other diagnostic name prefixes are interpreted as variables stored in the GEOS-Chem State_Diag object.

CHAPTER

TEN

INSTALLING DEPENDENCIES WITH SPACK

10.1 Downloading and Installing Software using Spack

The Spack Package Manager may be used to download and build CMake, MPI, and NetCDF libraries needed for GCHP. You will need to have a C/C++/Fortran compiler such as GNU Compiler Collection available locally before you start. You can use this local compiler to later install a different compiler version through Spack. The following steps successfully create a GCHP environment using GCC 9.3.0 and OpenMPI 4.0.4 through Spack. To install different versions of GCC and OpenMPI, simply change the version numbers used in the commands. Scroll down for additional info on using IntelMPI, Intel compilers, or MVAPICH2. IntelMPI and MVAPICH2 are free alternative MPI implementations, while Intel compilers are a paid product for compiling libraries and GCHP.

To begin using Spack, clone the latest version by typing `git clone https://github.com/spack/spack.git`. Execute the following commands to initialize Spack's environment (replacing `/path/to/spack` with the path of your *spack* directory). Add these commands to an environment initialization script for easy reuse.

```
$ export SPACK_ROOT=/path/to/spack
$ . $SPACK_ROOT/share/spack/setup-env.sh
```

If you do not already have a copy of your preferred text editor, you can use Spack to install and load one before proceeding (e.g. `spack install emacs; spack load emacs`).

10.1.1 Installing compilers

Ensure Spack recognizes any existing compiler on your system by typing `spack compilers`. You can use this compiler to build a new one.

Setting up and using GNU compilers

GNU compilers are free, open source, and easily installable through Spack. Execute the following at the command prompt to install version 9.3.0 of the GNU Compiler Collection:

```
$ spack install gcc@9.3.0
```

The `package@VERSION` notation is common to all packages in Spack and can be customized to choose different versions.

The installation of `gcc` may take a long time. Once it is complete, you'll need to add it to Spack's list of compilers using the following command:

```
$ spack compiler find $(spack location -i gcc@9.3.0)
```

Setting up and using Intel compilers

If you would like to use Intel compilers, whether pre-installed on your system or which you would like to install through Spack using an existing license, follow the instructions for pre-installed Intel compilers or for Intel compilers you want to install through Spack on the official Spack documentation site. In the instructions below, simply replace `%gcc@9.3.0` with `%intel` to use your Intel compilers to build libraries.

10.1.2 Installing basic dependencies

Make sure that spack recognizes your new compiler by typing `spack compilers`, which should display a list of compilers including GCC 9.3.0.

You should now install Git and CMake using Spack:

```
$ spack install git@2.17.0%gcc@9.3.0
$ spack install cmake@3.16.1%gcc@9.3.0
```

Installing without Slurm support

If you do not intend to use a job scheduler like Slurm to run GCHP, use the following commands to install MPI and NetCDF-Fortran. Otherwise, scroll down to see necessary modifications you must make to include Slurm support.

OpenMPI

```
$ spack install openmpi@4.0.4%gcc@9.3.0
$ spack install netcdf-fortran%gcc@9.3.0 ^netcdf-c^hdf5^openmpi@4.0.4
```

Intel MPI

```
  $ spack install intel-mpi%gcc@9.3.0
  $ spack install netcdf-fortran%gcc@9.3.0 ^intel-mpi



**MVAPICH2**
```

```
$ spack install mvapich2%gcc@9.3.0
$ spack install netcdf-fortran%gcc@9.3.0 ^netcdf-c^hdf5^mvapich2
```

Configuring libraries with Slurm support

If you know the install location of Slurm, edit your spack packages settings at `$HOME/.spack/packages.yaml` (you may need to create this file) with the following:

```
packages:
 slurm:
  paths:
   slurm: /path/to/slurm
  buildable: False
```

This will ensure that when your MPI library is built with Slurm support requested, Spack will correctly use your preinstalled Slurm rather than trying to install a new version.

OpenMPI

You may also run into issues building OpenMPI if your cluster has preexisting versions of PMIx that are newer than OpenMPI's internal version. OpenMPI will search for and use the newest version of PMIx installed on your system, which will likely cause a crash during build because OpenMPI requires you to build with the same libevent library as was used to build PMIx. This information may not be readily available to you, in which case you can tweak the build arguments for OpenMPI to always use OpenMPI's internal version of PMIx. Open `$SPACK_ROOT/var/spack/repos/builtin/packages/openmpi/package.py` and navigate to the `configure_args()` function. In the body of this function, place the following line:

```
config_args.append('--with-pmix=internal')
```

Building libraries with Slurm support

OpenMPI

You need to tell Spack to build OpenMPI with Slurm support and to build NetCDF-Fortran with the correct OpenMPI version as a dependency:

```
$ spack install openmpi@4.0.4%gcc@9.3.0 +pmi schedulers=slurm
$ spack install netcdf-fortran%gcc@9.3.0  ^netcdf-c^hdf5^openmpi@4.0.4+pmi␣
↪schedulers=slurm
```

Intel MPI

No build-time tweaks need to be made to install Intel MPI with Slurm support.

```
$ spack install intel-mpi%gcc@9.3.0
$ spack install netcdf-fortran%gcc@9.3.0 ^intel-mpi
```

Scroll down to find environment variables you need to set when running GCHP with Intel MPI, including when using Slurm.

MVAPICH2

Like OpenMPI, you must specify that you want to build MVAPICH2 with Slurm support and build NetCDF-Fortran with the correct MVAPICH2 version.

```
$ spack install mvapich2%gcc@9.3.0 process_managers=slurm
$ spack install netcdf-fortran%gcc@9.3.0 ^netcdf-c^hdf5^mvapich2
```

10.1.3 Loading Spack libraries for use with GCHP and ESMF

After installing the necessary libraries, place the following in a script that you will run before building/running GCHP (such as `$HOME/.bashrc` or a separate environment script) to initialize Spack and load requisite packages for building ESMF and GCHP:

OpenMPI

```
export SPACK_ROOT=$HOME/spack #your path to Spack
source $SPACK_ROOT/share/spack/setup-env.sh
if [[ $- = *i* ]] ; then
 echo "Loading Spackages, please wait ..."
fi
#==============================================================================
%%%%% Load Spackages %%%%%
#==============================================================================
```

(continues on next page)

(continued from previous page)

```
# List each Spack package that you want to load
pkgs=(gcc@9.3.0            \
 git@2.17.0           \
 netcdf-fortran@4.5.2 \
 cmake@3.16.1         \
 openmpi@4.0.4        )

# Load each Spack package
for f in ${pkgs[@]}; do
  echo "Loading $f"
  spack load $f
done

export MPI_ROOT=$(spack location -i openmpi)
export ESMF_COMPILER=gfortran #intel for intel compilers
export ESMF_COMM=openmpi
```

IntelMPI

```
export SPACK_ROOT=$HOME/spack #your path to Spack
source $SPACK_ROOT/share/spack/setup-env.sh
if [[ $- = *i* ]] ; then
 echo "Loading Spackages, please wait ..."
fi
#==============================================================================
%%%%% Load Spackages %%%%%
#==============================================================================
# List each Spack package that you want to load
pkgs=(gcc@9.3.0            \
 git@2.17.0           \
 netcdf-fortran@4.5.2 \
 cmake@3.16.1         \
 intel-mpi        )

# Load each Spack package
for f in ${pkgs[@]}; do
  echo "Loading $f"
  spack load $f
done

export MPI_ROOT=$(spack location -i intel-mpi)
export ESMF_COMPILER=gfortran #intel for intel compilers
export ESMF_COMM=intelmpi

# Environment variables only needed for Intel MPI
export I_MPI_CC=gcc #icc for intel compilers
export I_MPI_CXX=g++ #icpc for intel compilers
export I_MPI_FC=gfortran #ifort for intel compilers
export I_MPI_F77=gfortran #ifort for intel compilers
export I_MPI_F90=gfortran #ifort for intel compilers

export I_MPI_PMI_LIBRARY=/path/to/slurm/libpmi2.so #when using srun through Slurm
#unset I_MPI_PMI_LIBRARY #when using mpirun
```

MVAPICH2

```
export SPACK_ROOT=$HOME/spack #your path to Spack
source $SPACK_ROOT/share/spack/setup-env.sh
if [[ $- = *i* ]] ; then
 echo "Loading Spackages, please wait ..."
fi
#===============================================================================
%%%%% Load Spackages %%%%%
#===============================================================================
# List each Spack package that you want to load
pkgs=(gcc@9.3.0            \
 git@2.17.0           \
 netcdf-fortran@4.5.2 \
 cmake@3.16.1         \
 mvapich2        )

# Load each Spack package
for f in ${pkgs[@]}; do
  echo "Loading $f"
  spack load $f
done

export MPI_ROOT=$(spack location -i mvapich2)
export ESMF_COMPILER=gfortran #intel for intel compilers
export ESMF_COMM=mvapich2
```

You can also add other packages you've installed with Spack like `emacs` to the `pkgs` lists above.

10.2 ESMF and your environment file

You must load your environment file prior to building and running GCHP.

```
$ source /home/envs/gchpctm_ifort18.0.5_openmpi4.0.1.env
```

If you don't already have ESMF 8.0.0+, you will need to download and build it. You only need to build ESMF once per compiler and MPI configuration (this includes for ALL users on a cluster!). It is therefore worth downloading and building somewhere stable and permanent, as almost no users of GCHP would be expected to need to modify or rebuild ESMF except when adding a new compiler or MPI. Instructions for downloading and building ESMF are available at the GCHP wiki. ESMF may be installable through Spack in the future.

It is good practice to store your environment setup in a text file for reuse. Below are a couple examples that load libraries and export the necessary environment variables for building and running GCHP. Note that library version information is included in the filename for easy reference. Be sure to use the same libraries that were used to create the ESMF build install directory stored in environment variable `ESMF_ROOT`.

Environment file example 1

```
# file: gchpctm_ifort18.0.5_openmpi4.0.1.env

# Start fresh
module --force purge

# Load modules (some include loading other libraries such as netcdf-C and hdf5)
module load intel/18.0.5
module load openmpi/4.0.1
module load netcdf-fortran/4.5.2
```

(continues on next page)

(continued from previous page)

```
module load cmake/3.16.1

# Set environment variables
export CC=gcc
export CXX=g++
export FC=ifort

# Set location of ESMF
export ESMF_ROOT=/n/lab_shared/libraries/ESMF/ESMF_8_0_1/INSTALL_ifort18_openmpi4
```

Environment file example 2 (Spack libraries built with a pre-installed compiler)

```
# file: gchpctm_gcc7.4_openmpi.rc

# Start fresh
module --force purge

# Load modules
module load gcc-7.4.0
spack load cmake
spack load openmpi%gcc@7.4.0
spack load hdf5%gcc@7.4.0
spack load netcdf%gcc@7.4.0
spack load netcdf-fortran%gcc@7.4.0

# Set environment variables
export CC=gcc
export CXX=g++
export FC=gfortran

# Set location of ESMF
export ESMF_ROOT=/n/home/ESMFv8/DEFAULTINSTALLDIR
```

CHAPTER

ELEVEN

USING 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.

11.1 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.

11.2 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 on Harvard's Cannon cluster up to 360 cores and c90 (~1x1.25) resolution averaged 15% slower than equivalent non-container runs, and may perform worse 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.

11.3 Setting up and running GCHP using Singularity

Available GCHP images are listed on Docker Hub. The following command pulls the image of GCHP 13.0.0-beta1 and converts it to a Singularity image named *gchp.sif* in your current directory.

```
$ singularity pull gchp.sif docker://geoschem/gchp:13.0.0-beta.1-8-gcbcd8e4
```

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 *runScript-Samples/* 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 *runConfig.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.

11.4 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 input.geos 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

TWELVE

PLOTTING GCHP OUTPUT

todo

CHAPTER

THIRTEEN

STRETCHED-GRID SIMULATIONS

Note: Stretched-grid simulations are described in [Bindle et al., 2020]. The paper also discusses things you should consider, and offers guidance for choosing appropriate stretching parameters.

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 two things:

1. Create a restart file for your simulation.
2. Update `runConfig.sh` to specify the grid and restart file.

Before setting up your stretched-grid simulation, you will need to choose stretching parameters.

13.1 Choose stretching parameters

The *target face* is 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.

13.2 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 initial_GEOSChem_rst.c90_standard.nc      \
               --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 `initial_GEOSChem_rst.c90_standard.nc`
: Specifies the input restart file is `initial_GEOSChem_rst.c90_standard.nc` (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.

13.3 Update your configuration files

Modify the section of `runConfig.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.

```
#------------------------------------------------
#   Internal Cubed Sphere Resolution
#------------------------------------------------

# Primary resolution is an integer value. Set stretched grid to ON or OFF.
#   24 ~ 4x5, 48 ~ 2x2.25, 90 ~ 1x1.25, 180 ~ 1/2 deg, 360 ~ 1/4 deg
```

(continues on next page)

(continued from previous page)

```
CS_RES=24
STRETCH_GRID=ON

# Stretched grid parameters
# Rules and notes:
#    (1) Minimum STRETCH_FACTOR is 1.0001
#    (2) Target lat and lon must be floats (contain decimal)
#    (3) Target lon must be in range [0,360)
STRETCH_FACTOR=3.0
TARGET_LAT=40.0
TARGET_LON=260.0
```

Next, modify the section of `runConfig.sh` that specifies the simulation restart file. Set `INITIAL_RESTART` to the restart file we created in the *previous step*.

```
#------------------------------------------------
#    Initial Restart File
#------------------------------------------------
# By default the linked restart files in the run directories will be
# used. Please note that HEMCO restart variables are stored in the same
# restart file as species concentrations. Initial restart files available
# on gcgrid do not contain HEMCO variables which will have the same effect
# as turning the HEMCO restart file option off in GC classic. However, all
# output restart files will contain HEMCO restart variables for your next run.
# INITIAL_RESTART=initial_GEOSChem_rst.c${CS_RES}_TransportTracers.nc

# You can specify a custom initial restart file here to overwrite:
INITIAL_RESTART=sg_restart_c48_3_260_40.nc
```

Lastly, execute **`./runConfig.sh`** to update to update your run directory's configuration files.

```
$ ./runConfig.sh
```

CHAPTER

FOURTEEN

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.

14.1 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.

14.2 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',
::
```

14.3 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

FIFTEEN

STRETCHED-GRID SIMULATION: EASTERN US

This tutorial walks you through setting up and running a stretched-grid simulation for ozone in the eastern US. 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 were choosen so that the target face covered the eastern US. Some back-of-the-envelope resolution calculations are

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

and

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

and

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

and

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

where N is the cubed-sphere size and S is the stretch-factor. The actual value 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, like 2.0, would have a refinement that more broad, and the range resolutions would be smaller.

15.1 Tutorial prerequisites

Before continuing with the tutorial:

- You need to be able to run GCHP simulations
- You need to install gcpy >= 1.0.0, and cartopy >= 0.19
- You need emissions data and MERRA2 data for July 2019

Create a new run directory. This run directory should be use full chemistry with standard simulation options, and use MERRA2 meteorology. Make the following modifications to `runConfig.sh`:

- Change the simulation's start time to `"20190701 000000"`
- Change the simulation's end time to `"20190708 000000"`
- Change the simulation's duration to `"00000007 000000"`
- Change `timeAvg_freq` to `"240000"` (daily diagnostics)
- Change `timeAvg_dur` to `"240000"` (daily diagnostics)
- 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.

Note: I chose to use 30 cores on 1 node, and the simulation took 7 hours to run. For comparison, I also ran the simulation on 180 cores across 6 nodes, and that took about 2 hours.

Update `gchp.local.run` so `nCores` matches your setting in `runConfig.sh`. Now you are ready to continue with the tutorial. The rest of the tutorial assume that your current working directory is your run directory.

15.2 Create your restart file

First, create a restart file for the simulation. GCHP ingests the restart file directly (no online regridding), so the first thing you need to do is regrid a restart file to your stretched-grid. You can regrid `initial_GEOSChem_rst.c48_fullchem.nc` with GCPy like so:

```
$ python -m gcpy.file_regrid                                \
     -i initial_GEOSChem_rst.c48_fullchem.nc                \
     --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: This command takes about a minute to run. If you regridding a large restart file (e.g., C180) it may take significantly longer.

15.3 Update `runConfig.sh`

Make the following updates to `runConfig.sh`:

- Change `INITIAL_RESTART` to use `initial_GEOSChem_rst.EasternUS_SG_fullchem.nc`
- Change `CS_RES` 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`

Execute `runConfig.sh` to apply the updates to the various configuration files:

```
$ ./runConfig.sh
```

15.4 Run GCHP

Run GCHP:

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

15.5 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
```

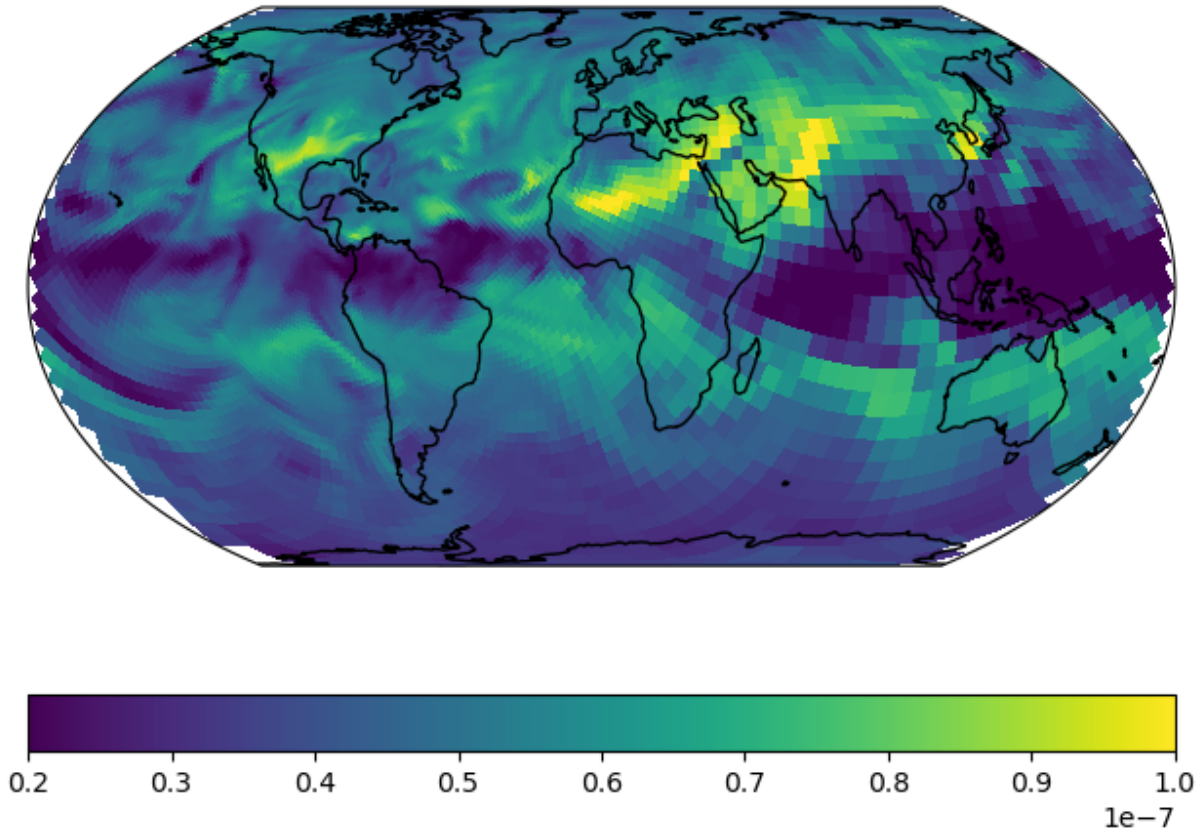
Plot ozone at model level 22:

```
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())
ax.set_global()
ax.coastlines()

# Plot data on each face
for face_idx in range(6):
    x = ds.corner_lons.isel(nf=face_idx)
```

(continues on next page)

(continued from previous page)

```
    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

SIXTEEN

SUPPORT GUIDELINES

GEOS-Chem support is maintained by the GEOS-Chem Support Team (GCST). The GCST members are based 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.

16.1 How to report a bug

We use GitHub to track issues. To report a bug, open a new issue. Please include all the information that might be relevant, including instructions for reproducing the bug.

16.2 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.

16.3 How to submit changes

Please see "Contributing Guidelines".

16.4 How to request an enhancement

Please see "Contributing Guidelines".

CHAPTER

SEVENTEEN

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.

17.1 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.

We use ReadTheDocs to host the GCHP user documentation: https://gchp.readthedocs.io.

17.2 How to submit changes

We use GitHub Flow, so all changes happen through pull requests. This workflow is described here: GitHub Flow. If your change affects multiple submodules, submit a pull request for each submodule with changes, and link to these submodule pull requests in your main pull request.

As the author you are responsible for:

- Testing your changes
- Updating the user documentation (if applicable)
- Supporting issues and questions related to your changes in the near-term

17.3 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.

17.4 How to request an enhancement

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.

17.5 How to report a bug

Please see "Support Guidelines".

17.6 Where can I ask for help?

Please see "Support Guidelines".

CHAPTER

EIGHTEEN

EDITING THIS USER GUIDE

This user guide is generated with Sphinx. Sphinx is an open-source Python project designed to make writing software documentation easier. The documentation is written in a reStructuredText (it's similar to markdown), which Sphinx extends for software documentation. The source for the documentation is the `docs/source` directory in top-level of the source code.

18.1 Quick start

To build this user guide on your local machine, you need to install Sphinx. 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.

```
$ pip install sphinx sphinx-rtd-theme sphinxcontrib-bibtex recommonmark
```

To build this user guide locally, navigate to the `docs/` directory and make the `html` target.

```
gcuser:~$ cd gcpy/docs
gcuser:~/gcpy/docs$ 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`.

18.2 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's

A good starting point would be Eric Holscher's presentations followed by the reStructuredText primer.

18.3 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 `$` should be used to denote the console's prompt. If the current working directory is relevant to the instructions, a prompt like `gcuser:~/path1/path2$` should be used.

Inline literals (e.g. the `$` above) should use the `:literal:` role.

CHAPTER

NINETEEN

GIT SUBMODULES

19.1 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 GCHPctm, i.e.,

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

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 GCHPctm fork, you should see the submodules point to your submodule forks.

CHAPTER

TWENTY

TERMINOLOGY

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.

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

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.

restart file A NetCDF file with initial conditions for a simulation.

run directory A directory that stores a GEOS-Chem simulation. This directory contains configuration files, the simulations ouptut, and sometimes input files like *restart files*.

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.

CHAPTER

TWENTYONE

VERSIONING

todo

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. *Journal of Geophysical Research: Atmospheres*, 106(D19):23073–23095, October 2001. doi:10.1029/2001JD000807.

[Keller et al., 2014] Keller, C. A., Long, M. S., Yantosca, R. M., Da Silva, A. M., Pawson, S., and Jacob, D. J. HEMCO v1.0: a versatile, ESMF-compliant component for calculating emissions in atmospheric models. *Geoscientific Model Development*, 7(4):1409–1417, July 2014. doi:10.5194/gmd-7-1409-2014.

[Long et al., 2015] Long, M. S., Yantosca, R., Nielsen, J. E., 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. *Geoscientific Model Development*, 8(3):595–602, March 2015. doi:10.5194/gmd-8-595-2015.

[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.

[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.

[Bindle et al., 2020] 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., Pawson, S., and Jacob, D. J. Grid-Stretching Capability for the GEOS-Chem 13.0.0 Atmospheric Chemistry Model. *Geoscientific Model Development*, December 2020. doi:10.5194/gmd-2020-398.

INDEX

Symbols

B

C

E

G

H

I

R

S

T