HEMCO

Release 3.6.0

GEOS-Chem Support Team

Feb 01, 2023

HEMCO STANDALONE USER GUIDE

The **Harmonized Emissions Component (HEMCO)** is a software component for computing atmospheric emissions from different sources, regions, and species on a user-defined grid. It can combine, overlay, and update a set of data inventories *base emissions* and *scale factors*, as specified by the user through *the HEMCO configuration file*. Emissions that depend on environmental variables and non-linear parameterizations are calculated in separate *HEMCO extensions*. HEMCO can be run in *standalone mode* or *coupled to an atmospheric model*. A more detailed description of HEMCO is given in Keller *et al.* [[Keller et al., 2014]] and Lin *et al.* [[Lin et al., 2021]].

CHAPTER

ONE

INTRODUCTION TO THIS GUIDE

In this **HEMCO Standalone User Guide**, you will learn how to run HEMCO in **standalone mode** (i.e. not connected to an external model).

For more information about how to configure HEMCO simulations and how to interface HEMCO to external models, please see the *HEMCO Reference Guide*.

1.1 Steps to follow:

1.1.1 Obtain the required hardware

In this chapter, we provide information about the computer equipment that you will need in order to run **`HEMCO`** in standalone mode (aka the **`HEMCO standalone`**).

Computer system requirements

Before you can run HEMCO standalone, you will need to have one the following items.

1. A Unix/Linux based computer system, OR:
2. An account on the Amazon Web Services cloud computing platform.

If your institution has computational resources (e.g. a shared computer cluster with many cores, sufficient disk storage and memory), then you can run HEMCO standalone there. Contact your IT staff for assistance.

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

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

Memory and disk requirements

If you plan to run HEMCO standalone on a local computer system, please make sure that your system has sufficient memory and disk space.

We would recommend at least 4 GB of RAM to run HEMCO standalone. However, if you will be reading data sets at very fine horizional resolution, you will want to increase the memory to perhaps 20-30 GB/RAM.

Also make sure that you have enough disk space to store the amount of input data for your HEMCO standalone simulations.

1.1.2 Install required software libraries

This chapter lists the required software libraries that you must have installed on your system in order to use **`HEMCO standalone`**.

- If you are using a shared computer cluster, then many of these libraries have probably already been pre-installed by your IT staff. Consult with them for more information.
- If you plan to run HEMCO standalone on the Amazon Web services cloud, then all of these libraries will be included with the Amazon Machine Image (AMI) that you will use to start your cloud instance.
- If your computer cluster has none of these libraries installed, then you will have to install them yourself (cf. *Build libraries with Spack*).

Supported compilers for HEMCO

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

The Intel Compiler Suite

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

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

The GCST has tested **`HEMCO`** with these versions (but others may work as well):

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

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

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

The GNU Compiler Collection

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

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

The GCST has tested HEMCO standalone with these versions (but others may work as well):

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

Best way to install: *With Spack*.

Required software packages for HEMCO

Git

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

The HEMCO source code can be downloaded using the Git source code management system from the https://github.com/HEMCO repository.

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

CMake

CMake is software that creates **Makefiles**, or scripts that direct how the HEMCO source code will be compiled into an executable. You will need CMake version 3.13 or later to build HEMCO.

Best way to install: *With Spack*.

GNU Make

GNU Make (sometimes just known as **make**) is software that can build executables from source code. It executes the instructions in the Makefiles created by *CMake*.

Best way to install: *With Spack*.

The netCDF library (plus dependencies)

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

Best way to install: *With Spack*.

Optional but recommended software packages

GCPy

GCPy is our recommended python companion software to HEMCO.

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

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

gdb and cgdb

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

Best way to install: *With Spack*.

ncview

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

nco

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

Best way to install: *With Spack*.

cdo

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

Best way to install: *With Spack*.

1.1.3 Configure your login environment

In this chapter, you will learn how to load the software packages that you have created into your computational environment. This will need to be done each time you log in to your computer system.

Tip: You may skip this section if you plan on using HEMCO standalone on an Amazon EC2 cloud instance. When you initialize the EC2 instance with one of the pre-configured Amazon Machine Images (AMIs) all of the required software libraries will be automatically loaded.

An environment file does the following:

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

Environment files allow you to easily switch between different sets of libraries. For example, you can keep one environment file to load the Intel Compilers for HEMCO standalone and another to load the GNU Compilers.

For general information about how libraries are loaded, see our *Library Guide* in the Supplemental Guides section.

We recommend that you place module load commands into a separate **environment file** rather than directly into your `~/.bashrc` or `~/.bash_aliases` startup scripts.

Sample environment file for GNU 10.2.0 compilers

Below is a sample environment file from the Harvard Cannon computer cluster. This file will load software libraries built with the GNU 10.2.0 compilers.

Save the code below (with any appropriate modifications for your own computer system) to a file named `~/gnu102.env`.

```
# Echo message if we are in a interactive (terminal) session
if [[ $- = *i* ]] ; then
  echo "Loading modules for GEOS-Chem, please wait ..."
fi

#==============================================================================
# Modules (specific to Cannon @ Harvard)
#==============================================================================

# Remove previously-loaded modules
module purge

# Load modules for GNU Compilers v10.2.0
module load git/2.17.0-fasrc01
module load gcc/10.2.0-fasrc01
module load openmpi/4.1.0-fasrc01
```

(continues on next page)

(continued from previous page)

```
module load netcdf-fortran/4.5.3-fasrc03
module load flex/2.6.4-fasrc01
module load cmake/3.17.3-fasrc01

#==============================================================================
# Environment variables
#==============================================================================

# Parallelization settings
export OMP_NUM_THREADS=8
export OMP_STACKSIZE=500m

# Make all files world-readable by default
umask 022

# Specify compilers
export CC=gcc
export CXX=g++
export FC=gfortran

# Netcdf variables for CMake
# NETCDF_HOME and NETCDF_FORTRAN_HOME are automatically
# defined by the "module load" commands on Cannon.
export NETCDF_C_ROOT=${NETCDF_HOME}
export NETCDF_FORTRAN_ROOT=${NETCDF_FORTRAN_HOME}

# Set memory limits to max allowable
ulimit -c unlimited              # coredumpsize
ulimit -l unlimited              # memorylocked
ulimit -u 50000                  # maxproc
ulimit -v unlimited              # vmemoryuse
ulimit -s unlimited              # stacksize

# List modules loaded
module list
```

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

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

```
$ source ~/gnu102.env
```

Sample environment file for Intel 19 compilers

To load software libraries based on the Intel 19 compilers, we can start from our *GNU 10.2.0 environment file* and add the proper **`module load`** commands for Intel 19.

Add the code below (with the appropriate modifications for your system) into a file named `~/intel19.env`.

```
# Echo message if we are in a interactive (terminal) session
if [[ $- = *i* ]] ; then
  echo "Loading modules for GEOS-Chem, please wait ..."
```

(continues on next page)

(continued from previous page)

```
fi

#==============================================================================
# Modules (specific to Cannon @ Harvard)
#==============================================================================

# Remove previously-loaded modules
module purge

# Load modules for Intel compilers v19.0.4
module load git/2.17.0-fasrc01
module load intel/19.0.5-fasrc01
module load openmpi/4.0.1-fasrc01
module load netcdf-fortran/4.5.2-fasrc03
module load flex/2.6.4-fasrc01
module load cmake/3.17.3-fasrc01

#==============================================================================
# Environment variables
#==============================================================================

# Parallelization settings
export OMP_NUM_THREADS=8
export OMP_STACKSIZE=500m

# Make all files world-readable by default
umask 022

# Specify compilers
export CC=icc
export CXX=icpc
export FC=ifort

# Netcdf variables for CMake
# NETCDF_HOME and NETCDF_FORTRAN_HOME are automatically
# defined by the "module load" commands on Cannon.
export NETCDF_C_ROOT=${NETCDF_HOME}
export NETCDF_FORTRAN_ROOT=${NETCDF_FORTRAN_HOME}

# Set memory limits to max allowable
ulimit -c unlimited              # coredumpsize
ulimit -l unlimited              # memorylocked
ulimit -u 50000                  # maxproc
ulimit -v unlimited              # vmemoryuse
ulimit -s unlimited              # stacksize

# List modules loaded
module list
```

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

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

```
$ source intel19.env
```

Tip: Keep a separate environment file for each combination of modules that you will load.

Set environment variables for compilers

Add the following environment variables to your environment file to specify the compilers that you wish to use:

Table 1: Environment variables that specify the choice of compiler

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

These environment variables should be defined in your *environment file*.

Note: Only the Fortran compiler is needed to compile the HEMCO standalone. But if you need to *manually install libraries*, you will also need the C and C++ compilers.

Set environment variables for parallelization

The HEMCO standalone` uses OpenMP parallelization, which is an implementation of shared-memory (aka serial) parallelization.

Important: OpenMP-parallelized programs cannot execute on more than 1 computational node. Most modern computational nodes typically contain between 16 and 64 cores. Therefore, HEMCO standalone simulations will not be able to take advantage of more cores than these.

Add the following environment variables to your environment file to control the OpenMP parallelization settings:

`OMP_NUM_THREADS`

The `OMP_NUM_THREADS` environment variable sets the number of computational cores (aka threads) to use.

For example, the command below will tell HEMCO standalone to use 8 cores within parallel sections of code:

```
$ export OMP_NUM_THREADS=8
```

`OMP_STACKSIZE`

In order to use HEMCO standalone with OpenMP parallelization, you must request the maximum amount of stack memory in your login environment. (The stack memory is where local automatic variables and temporary `$OMP PRIVATE` variables will be created.) Add the following lines to your system startup file and to your GEOS-Chem run scripts:

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

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

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

Fix errors caused by incorrect settings

Be on the lookout for these errors:

1. If *OMP_NUM_THREADS* is set to 1, then your HEMCO standalone simulation will execute using only one computational core. This will make your simulation take much longer than is necessary.
2. If *OMP_STACKSIZE* environment variable is not included in your environment file (or if it is set to a very low value), you might encounter a **segmentation fault**. In this case, the HEMCO standalone "thinks" that it does not have enough memory to perform the simulation, even though sufficient memory may be present.

1.1.4 Download the source code

The **HEMCO** source code may be downloaded (aka "cloned") with Git. By default the **git clone** command will give you the **main** branch by default: default.

```
$ git clone https://github.com/geoschem/hemco.git HEMCO
$ cd HEMCO
```

If you would like a different version of HEMCO you can check out a different branch. For example, to check out the **dev** branch, type:

```
$ git checkout dev
```

You can also check out the **HEMCO** source code at the position of any tag. For example, to use the **HEMCO** version 3.0.0 code (which is by now an old version), type:

```
$ git checkout tags/3.0.0
```

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

1.1.5 Create a run directory

Note: Another useful resource for **HEMCO standalone** run directory creation instructions is our YouTube tutorial.

HEMCO standalone run directories are created from within the source code. A new run directory should be created for each different version of HEMCO 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`.

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

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

Enter ExtData path

The first time you create a HEMCO standalone run directory on your system you will be prompted for a path to the `ExtData` folder, which is the root data directory for HEMCO (as well as for GEOS-Chem).

The path that you specify 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:
-------------------------------------------------------------
```

Choose meteorology source

Enter the integer number that is next to the input meteorology source you would like to use.

```
=============================================================
HEMCO STANDALONE RUN DIRECTORY CREATION
=============================================================

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

Choose horizontal resolution

Enter the integer number that is next to the horizontal resolution you would like to use.

```
-------------------------------------------------------------
Choose horizontal resolution:
-------------------------------------------------------------
  1. 4.0 x 5.0
  2. 2.0 x 2.5
  3. 0.5 x 0.625
  4. 0.25 x 0.3125
  5. Custom
```

Enter HEMCO_Config.rc path

Provide the path to a `HEMCO_Config.rc` file with your emissions settings.

```
-------------------------------------------------------------
Enter the file path to a HEMCO_Config.rc with your
emissions settings.

 - This should be a HEMCO_Config.rc file from a
   pre-generated GEOS-Chem run directory and not a
   template config file from the GEOS-Chem repository.
```

(continues on next page)

(continued from previous page)

```
 - If you do not have a pre-generated HEMCO_Config.rc file,
   type ./HEMCO_Config.rc.sample at the prompt below.
   This will copy a sample configuration file into your
   run directory.  You can then edit this configuration
   file with your preferred emission settings.
-------------------------------------------------------------
```

If you have a pre-configured `HEMCO_Config.rc` file available (e.g. from a GEOS_Chem run directory), then then type the absolute path:

```
/path/to/my/HEMCO_Config.rc
```

If you do not have a `HEMCO_Config.rc` template file handy, then type:

```
./HEMCO_Config.rc.sample
```

This will copy sample `HEMCO_Config.rc` and `HEMCO_Diagn.rc` files to the run directory. You can edit these configuration files to include your preferred emission settings.

Refer to the *HEMCO Reference Guide* for more information about how to edit *the HEMCO configuration file*.

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

Enter run directory name

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

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

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

If you press return, a default name such as `hemco_4x5_merra2`, `hemco_2x25_geosfp`, etc. will be used.

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 a previous version.

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

If a run directory has successfully been created, the name of the run directory will be printed. If you used the default run directory name then you will see output similar to:

```
Created /path/to/hemco_4x5_merra2
```

etc.

Run directory contents

Navigate to the run directory that was just created and get a directory listing:

```
$ cd hemco_4x5_merra2
$ ls
build/    HEMCO_Config.rc  HEMCO_sa_Config.rc    HEMCO_sa_Spec.rc  OutputDir/  rundir.
↪version
CodeDir@  HEMCO_Diagn.rc   HEMCO_sa_Grid.4x5.rc  HEMCO_sa_Time.rc  README      ␣
↪runHEMCO.sh*
```

`build` is the folder is where you will *compile HEMCO standalone*.

`CodeDir` is a symbolic link back to the HEMCO source code.

`OutputDir` is the folder where diagnostic outputs will be generated.

Files ending in `.rc` are user-edtiable configuration files that control HEMCO standalone simulation options. We will discuss these in more detail more in the *Configure a simulation* chapter.

The `rundir.version` file contains information about the Git commit in the HEMCO source code corresponding to this run directory. You will see output similar to this:

```
This run directory was created with /path/to/hemco/HEMCO/run/createRunDir.sh.

HEMCO repository version information:

  Remote URL: git@github.com:geoschem/hemco.git
  Branch: dev
  Commit: Add fixes for generating HEMCO standalone run directory
  Date: Wed Jul 13 10:56:36 2022 -0400
  User: Melissa Sulprizio
  Hash: b29dac4

Changes to the following run directory files are tracked by git:

 [master (root-commit) b8e694d] Initial run directory
 7 files changed, 477 insertions(+)
 create mode 100644 HEMCO_Config.rc
 create mode 100644 HEMCO_Diagn.rc
```

(continues on next page)

(continued from previous page)

```
create mode 100644 HEMCO_sa_Config.rc
create mode 100644 HEMCO_sa_Grid.4x5.rc
create mode 100644 HEMCO_sa_Spec.rc
```

1.1.6 Build the executable

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

Once you have created a *run directory*, you may proceed to compile the HEMCO standalone source code into an executable file. You will compile HEMCO standalone from your *run directory*.

There are two steps to build HEMCO. The first step is to **configure your build settings** with *CMake*. Build settings cover options like enabling or disabling components or whether HEMCO should be compiled in `Debug` mode.

The second step is to **compile the source code into an executable**. For this, you will use *make*, which builds the executable according to your build settings.

Navigate to your build directory

A subdirectory named `build` is included in each *HEMCO standalone run directory* that you create. You can use this directory (known as a **build directory**) to create the HEMCO executable file.

You are not limited to using the build directory that is created inside the *run directory*. In fact, you can create as many build directories you wish in whatever location you wish. For example, if you want to compare HEMCO standalone performance on both *GNU* and *Intel* compilers, you could create two different build directories, one named `build_gnu` and the other `build_intel`. Build directories do not necessarily need to be kept in the *run directory*, but it is convenient to do so.

Each build directory is self-contained, so you can delete it at any point to erase the HEMCO standalone build and its configuration. Most users will typically only need to build HEMCO standalone once, so we recommend using the `build` subdirectory of the *run directory* as the location to create the HEMCO standalone exectuable.

Important: There is one rule for build directories: **a build directory should be a new directory**.

In the example below, we will use the `build` directory within the *run directory* to build the HEMCO standalone executable.

Navigate to the run directory:

```
$ cd /path/to/hemco/run/dir
```

Then navigate to the `build` folder within:

```
$ cd build
```

Initialize the build directory

Run *CMake* to initialze the build directory.

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

`CodeDir` is a symbolic link to the HEMCO source code directory.

The option `-DRUNDIR=..` specifies that the directory where we will run HEMCO standalone is one level above us. This makes sense as our `build` folder is a subdirectory of the run directory. (More about *build options* below:

You will see output similar to this:

```
-- The Fortran compiler identification is GNU 11.2.0
-- Detecting Fortran compiler ABI info
-- Detecting Fortran compiler ABI info - done
-- Check for working Fortran compiler: /path/to/gfortran - skipped
-- Checking whether /path/to/gfortran supports Fortran 90
-- Checking whether /path/to/gfortran supports Fortran 90 - yes
============================================================
HEMCO X.Y.Z
Current status: X.Y.Z
============================================================
-- Found OpenMP_Fortran: -fopenmp (found version "4.5")
-- Found OpenMP: TRUE (found version "4.5")
-- Found NetCDF: /path/to/netcdf/lib/libnetcdff.so
-- Bootstrapping  /path/to/hemco/run/dir
-- Settings:
  * OMP:          ON  OFF
  * USE_REAL8:    ON  OFF
-- Configuring done
-- Generating done
-- Build files have been written to: /path/to/hemco/run/dir/build
```

In the above example output, the version number `X.Y.Z` will refer to the actual HEMCO version number (e.g. `3.4.0`, `3.5.0`, etc.). Also the paths `/path/to/...` in your output instead be the actual paths to the compiler and libraries.

Configuring 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.

HEMCO has no required build settings. You can find the complete list of *HEMCO's build settings here*. The most frequently used build setting is `RUNDIR` which lets you specify one or more run directories where CMake will install HEMCO. Here, "install" refers to copying the compiled executable, and some supplemental files with build settings, to your run directories.

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: /path/to/hemco/run/dir/build
```

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

Compile HEMCO standalone

Compile HEMCO standalone with this command

```
$ make -j
```

The **-j** option will tell *GNU Make* to compile several source code files in parallel. This reduces overall compilation time.

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

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

```
$ make install
```

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

Now you have compiled HEMCO! You can now navigate back from the `build` folder to the run directory (which we remember is one level higher):

```
$ cd ..
```

Recompile when you change the source code

You need to recompile **HEMCO** 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 HEMCO standalone, simply do

```
$ make -j
$ make install
```

which will recompile HEMCO standalone and copy the new executable file to the run directory.

HEMCO standalone build options

RUNDIR
Paths to run directories where **make install** installs HEMCO standalone. 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
Specifies the build type. Allowable values are:

Release
The default option. Compiles HEMCO standalone for speed.

Debug
Compiles HEMCO standalone with several debugging flags turned on. This may help you find common errors such as array-out-of-bounds, division-by-zero, or not-a-number.

Important: The additional error checks that are applied with `Debug` will cause HEMCO standalone to run much more slowly! Do not use `Debug` for long production simulations.

HEMCO_Fortran_FLAGS_<COMPILER_ID>

Additional compiler options for HEMCO standalone for build type `<BUILD_TYPE>`.

<COMPILER_ID>
Valid values are `GNU` and `Intel`.

HEMCO_Fortran_FLAGS_<CMAKE_BUILD_TYPE>_<COMPILER_ID>
Compiler options for HEMCO standalone for the given *CMAKE_BUILD_TYPE*.

<COMPILER_ID>
Valid values are `GNU` and `Intel`.

1.1.7 Configure a simulation

Note: Another useful resource for instructions on configuring HEMCO run directories is our YouTube tutorial.

Navigate to your new directory, and examine the contents:

```
$ cd /path/to/hemco/run/dir
$ ls
build/           HEMCO_Diagn.rc        HEMCO_sa_Spec.rc  README
CodeDir@         HEMCO_sa_Config.rc    HEMCO_sa_Time.rc  rundir.version
HEMCO_Config.rc  HEMCO_sa_Grid.4x5.rc  OutputDir/        runHEMCO.sh*
```

The following files can be modified to set up your HEMCO standalone simulation.

HEMCO_sa_Config.rc
Main configuration file for the HEMCO standalone simulation. This file points to the other configuration files used to set up your simulation (e.g. *HEMCO_sa_Grid.4x5.rc*, *HEMCO_sa_Time.rc*).

This file typically references a *HEMCO_Config.rc* file using

```
>>>include HEMCO_Config.rc
```

which contains the emissions settings. Settings in *HEMCO_sa_Config.rc* will always override any settings in the included *HEMCO_Config.rc* file.

HEMCO_Config.rc
Contains emissions settings. *HEMCO_Config.rc* can be taken from a another model (such as GEOS-Chem), or can be built from a sample file.

For more information on editing *HEMCO_Config.rc*, please see the following chapters: *The HEMCO configuration file*, *Basic examples*, and *More configuration examples*.

Important: Make sure that the path to your data directory in the *HEMCO_Config.rc* file is correct. Otherwise, HEMCO standalone will not be able read data from disk.

HEMCO_Diagn.rc
Specifies which fields to save out to the HEMCO diagnostics file saved in `OutputDir` by default. The frequency to save out diagnostics is controlled by the *DiagnFreq* setting in *HEMCO_sa_Config.rc*

For more information, please see the chapter entitled *Configuration file for the Default collection*.

HEMCO_sa_Grid.4x5.rc
Defines the grid specification. Sample files are provided for 4.0 x 5.0, 2.0 x 2.5, 0.5 x 0.625, and 0.25 x 0.3125 global grids in `HEMCO/run/` and are automatically copied to the run directory based on options chosen when running `createRunDir.sh`. you choose to run with a custom grid or over a regional domain, you will need to modify this file manually.

HEMCO_sa_Spec.rc
Defines the species to include in the HEMCO standalone simulation. By default, the species in a GEOS-Chem full-chemistry simulation are defined. To include other species, you can modify this file by providing the species name, molecular weight, and other properties.

HEMCO_sa_Time.rc
Defines the start and end times of the HEMCO standalone simulation as well as the emissions timestep (s).

runHEMCO.sh
Sample run script for submitting a HEMCO standalone simulation via SLURM.

1.1.8 Download input data

Before starting a HEMCO standalone simulation, make sure that all of the relevant emissions and meteorology that you will need for your simulation are present on disk.

If you are located at an institution where there are several other HEMCO and/or GEOS-Chem users, then data for HEMCO standalone might already be located in a shared folder. Ask your sysadmin or IT staff.

If you are using HEMCO standalone on the Amazon Web Services EC2 cloud computing platform, then you will have access to an S3 bucket (`s3://gcgrid/`) with emissions inventories and meteorological data.

If you still need to download data for your HEMCO standalone simulation, we recommend using the **bashdatacatalog** tool. For more information, please see our Supplemental Guide entitled *Manage a data archive with bashdatacatalog*.

1.1.9 Run a simulation

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

Run interactively

First, navigate to your run directory (if you aren't already there):

```
$ cd /path/to/hemco/run/dir
```

You can run HEMCO standalone interactively at the command line by typing:

```
$ ./hemco_standalone -c HEMCO_sa_Config.rc
```

where `-c` specifies the path to the *HEMCO_sa_Config.rc* configuraiton file.

Run as batch job

Batch job run scripts will vary based on what job scheduler you have available. The example run script included in HEMCO standalone run directories (`runHEMCO.sh`) is for use with SLURM. You may modify this file for your system and preferences as needed.

At the top of all batch job scripts are configurable run settings. Most critically are requested # cores, # nodes, time, and memory. Figuring out the optimal values for your run can take some trial and error.

To submit a batch job using SLURM:

```
$ sbatch runHEMCO.sh
```

Standard output will be sent to a log file `HEMCO_SA.log` once the job is started. 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.

Verify a successful run

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

- *NetCDF* files are present in the `OutputDir/` subdirectory;
- The HEMCO log file `HEMCO.log` ends with `HEMCO X.Y.Z FINISHED.`;
- Standard output file `HEMCO_SA.log` ends with `HEMCO_STANDALONE FINISHED!`;
- 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

- Check to make sure you have downloaded all input files needed for your HEMCO standalone simulation.

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

CHAPTER

TWO

INTRODUCTION TO THIS GUIDE

In this **HEMCO Reference Guide**, you will learn about HEMCO configuration files, HEMCO extensions, HEMCO interfaces, and other technical information.

For more information about how run HEMCO standalone simulations, please see our *HEMCO Standalone User Guide*.

2.1 Contents

2.1.1 Basic examples

Note: The following sections contain simple HEMCO configuration file examples for demonstration purposes. If you are using HEMCO with an external model, then your HEMCO configuration file may be more complex than the examples shown below.

All emission calculation settings are specified in *the HEMCO configuration file*, which is named `HEMCO_Config.rc`.

Modification of the HEMCO source code (and recompilation) is only required if new extensions are added, or to use HEMCO in a new model environent (see sections *HEMCO under the hood* and *Interfaces*).

In the sections that follow, we provide some basic examples that demonstrate how to modify the configuration file to customize your HEMCO simulation.

Example 1: Add global anthropogenic emissions

Suppose monthly global anthropogenic CO emissions from the **MACCity** inventory [[Lamarque et al., 2010]] are stored in file `MACCity.nc` as variable `CO`. The following HEMCO configuration file then simulates CO emissions with gridded hourly scale factors applied to it (the latter taken from variable `factor` of file `hourly.nc`).

The horizontal grid and simulation datetimes employed by HEMCO depends on the HEMCO-to-model interface. If HEMCO is coupled to an external model (such as GEOS-Chem) these values are taken from the chemistry model. If run standalone, the grid specification and desired datetimes need be specified as described in Interfaces.

```
###############################################################################
### BEGIN SECTION SETTINGS
###############################################################################
ROOT:                       /dir/to/data
Logfile:                    HEMCO.log
DiagnFile:                  HEMCO_Diagn.rc
DiagnPrefix:                HEMCO_diagnostics
```

(continues on next page)

(continued from previous page)

```
Wildcard:                    *
Separator:                   /
Unit tolerance:              1
Negative values:             0
Only unitless scale factors: false
Verbose:                     0
Warnings:                    1

### END SECTION SETTINGS ###

############################################################################
### BEGIN SECTION EXTENSION SWITCHES
############################################################################
# ExtNr ExtName           on/off  Species
0       Base              : on    *
    --> MACCITY           :       true

### END SECTION EXTENSION SWITCHES ###

############################################################################
### BEGIN SECTION BASE EMISSIONS
############################################################################
# ExtNr Name sourceFile sourceVar sourceTime C/R/E SrcDim SrcUnit Species ScalIDs Cat␣
↪Hier

(((MACCITY
0 MACCITY_CO $ROOT/MACCity.nc  CO 1980-2014/1-12/1/0 C xy kg/m2/s CO 500 1 1
)))MACCITY

### END SECTION BASE EMISSIONS ###

############################################################################
### BEGIN SECTION SCALE FACTORS
############################################################################
# ScalID Name srcFile srcVar srcTime  CRE Dim Unit Oper

500 HOURLY_SCALFACT $ROOT/hourly.nc factor 2000/1/1/0-23 C xy 1 1

### END SECTION SCALE FACTORS ###

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

### END SECTION MASKS ###
```

The various attributes are explained in more detail in the *Base emissions* and *Scale factors* sections.

Note: We have used an index of 500 for `HOURLY_SCALFACT` in order to reduce confusion with the `Cat` and `Hier` values.

As described in *Data collections*, all of the files contained between the brackets `(((MACCITY` and `)))MACCITY` will be read if you set the switch

```
--> MACCITY           :       true
```

These files will be ignored if you set

```
--> MACCITY           :       false
```

This is a quick way to shut off individual emissions inventories without having to manually comment out many lines of code. You can add a set of brackets, with a corresponding true/false switch, for each emissions inventory that you add to the configuration file.

Example 2: Overlay regional emissions

To add regional monthly anthropogenic CO emissions from the EMEP European inventory [[Vestreng et al., 2009]] (in file `EMEP.nc`) to the simulation, modify the configuration file as follows:

```
##############################################################################
#### BEGIN SECTION EXTENSION SWITCHES
##############################################################################
# ExtNr ExtName           on/off  Species
0       Base              : on    *
    --> MACCITY           :       true
    --> EMEP              :       true

### END SECTION EXTENSION SWITCHES ###

##############################################################################
### BEGIN SECTION BASE EMISSIONS
##############################################################################
#ExtNr Name srcFile srcVar srcTime CRE Dim Unit Species ScalIDs Cat Hier

(((MACCITY
0 MACCITY_CO $ROOT/MACCity.nc CO 1980-2014/1-12/1/0 C xy kg/m2/s CO  500      1 1
)))MACCITY

(((EMEP
0 EMEP_CO    $ROOT/EMEP.nc    CO 2000-2014/1-12/1/0 C xy kg/m2/s CO  500/1001 1 2
)))EMEP

### END SECTION BASE EMISSIONS###

##############################################################################
### BEGIN SECTION SCALE FACTORS
##############################################################################
#ScalID Name srcFile srcVar srcTime CRE Dim Unit Oper

500 HOURLY_SCALFACT $ROOT/hourly.nc factor 2000/1/1/0-23 C xy 1 1

### END SECTION SCALE FACTORS ###

##############################################################################
### BEGIN SECTION MASKS
##############################################################################
#ScalID Name srcFile srcVar srcTime CRE Dim Unit Oper Box

1001 MASK_EUROPE $ROOT/mask_europe.nc MASK 2000/1/1/0 C xy 1 1 -30/30/45/70
```

(continues on next page)

(continued from previous page)

```
### END SECTION MASKS ###
```

For now, we have omitted the **Settings section** because nothing has changed since *the previous example*.

Note the increased hierarchy (2) of the regional EMEP inventory compared to the global MACCity emissions (1) in column *Hier*. This will cause the EMEP emissions to replace the MACCity emissions in the region where EMEP is defined, which is specified by the MASK_EUROPE variable.

Example 3: Adding the AEIC aircraft emissions

To add aircraft emissions from the AEIC inventory [[Stettler et al., 2011]], available in file `AEIC.nc`, modify the *configuration file* accordingly:

```
##############################################################################
#### BEGIN SECTION EXTENSION SWITCHES
##############################################################################
# ExtNr ExtName           on/off  Species
0       Base              : on    *
    --> MACCITY           :       true
    --> EMEP              :       true
    --> AEIC              :       true
### END SECTION EXTENSION SWITCHES ###

##############################################################################
#### BEGIN SECTION BASE EMISSIONS
##############################################################################
#ExtNr Name srcFile srcVar srcTime CRE Dim Unit Species ScalIDs Cat Hier

(((MACCITY
0 MACCITY_CO $ROOT/MACCity.nc CO 1980-2014/1-12/1/0 C xy  kg/m2/s CO 500        1 1
)))MACCITY

(((EMEP
0 EMEP_CO    $ROOT/EMEP.nc    CO 2000-2014/1-12/1/0 C xy  kg/m2/s CO 500 1/1001 1 2
)))EMEP

(((AEIC
0 AEIC_CO    $ROOT/AEIC.nc    CO 2005/1-12/1/0      C xyz kg/m2/s CO -          2 1
)))AEIC

### END SECTION BASE EMISSIONS ###
```

Note the change in the emission category (column *Cat*) from 1 to 2. In this example, category 1 represents anthropogenic emissions and category 2 represents aircraft emissions.

Example 4: Add biomass burning emissions

GFED4 biomass burning emissions (Giglio et al, 2013), which are implemented as a HEMCO Extension, can be added to the simulation by:

1. Adding the corresponding extension to section **Extension Switches**
2. Adding all the input data needed by GFED4 to section **Base Emissions**.

The extension number defined in the **Extension Switches** section must match the corresponding *ExtNr* entry in the Base Emissions section (in this example, 111).

```
##############################################################################
#### BEGIN SECTION EXTENSION SWITCHES
##############################################################################
# ExtNr ExtName           on/off  Species
0       Base              : on    *
    --> MACCITY           :       true
    --> EMEP              :       true
    --> AEIC              :       true
#------------------------------------------------------------------------------
111     GFED              : on    CO
    --> GFED3             :       false
    --> GFED4             :       true
    --> GFED_daily        :       false
    --> GFED_3hourly      :       false
    --> Scaling_CO        :       1.05

### END SECTION EXTENSION SWITCHES ###

##############################################################################
#### BEGIN SECTION BASE EMISSIONS
##############################################################################
#ExtNr Name srcFile srcVar srcTime CRE Dim Unit Species ScalIDs Cat Hier

(((MACCITY
0 MACCITY_CO $ROOT/MACCity.nc  CO 1980-2014/1-12/1/0 C xy  kg/m2/s CO 500      1 1
)))MACCITY

(((EMEP
0 EMEP_CO    $ROOT/EMEP.nc     CO 2000-2014/1-12/1/0 C xy  kg/m2/s CO 500/1001 1 2
)))EMEP

(((AEIC
0 AEIC_CO    $ROOT/AEIC.nc     CO 2005/1-12/1/0      C xyz kg/m2/s CO -        2 1
)))AEIC

##############################################################################
###  BEGIN SECTION EXTENSION DATA (subsection of BASE EMISSIONS SECTION
###
### These fields are needed by the extensions listed above. The assigned ExtNr
### must match the ExtNr entry in section 'Extension switches'. These fields
### are only read if the extension is enabled.  The fields are imported by the
### extensions by field name.  The name given here must match the name used
### in the extension's source code.
##############################################################################

# --- GFED biomass burning emissions (Extension 111) ---
111 GFED_HUMTROP    $ROOT/GFED3/v2014-10/GFED3_humtropmap.nc              humtropp
↪       2000/1/1/0              C xy 1          * - 1 1
```

(continues on next page)

(continued from previous page)

```
(((GFED3
111 GFED_WDL        $ROOT/GFED3/v2014-10/GFED3_gen.1x1.$YYYY.nc          GFED3_BB__
↪WDL_DM  1997-2011/1-12/01/0    C xy kgDM/m2/s * - 1 1
111 GFED_AGW        $ROOT/GFED3/v2014-10/GFED3_gen.1x1.$YYYY.nc          GFED3_BB__
↪AGW_DM  1997-2011/1-12/01/0    C xy kgDM/m2/s * - 1 1
111 GFED_DEF        $ROOT/GFED3/v2014-10/GFED3_gen.1x1.$YYYY.nc          GFED3_BB__
↪DEF_DM  1997-2011/1-12/01/0    C xy kgDM/m2/s * - 1 1
111 GFED_FOR        $ROOT/GFED3/v2014-10/GFED3_gen.1x1.$YYYY.nc          GFED3_BB__
↪FOR_DM  1997-2011/1-12/01/0    C xy kgDM/m2/s * - 1 1
111 GFED_PET        $ROOT/GFED3/v2014-10/GFED3_gen.1x1.$YYYY.nc          GFED3_BB__
↪PET_DM  1997-2011/1-12/01/0    C xy kgDM/m2/s * - 1 1
111 GFED_SAV        $ROOT/GFED3/v2014-10/GFED3_gen.1x1.$YYYY.nc          GFED3_BB__
↪SAV_DM  1997-2011/1-12/01/0    C xy kgDM/m2/s * - 1 1
)))GFED3

(((GFED4
111 GFED_WDL        $ROOT/GFED4/v2015-03/GFED4_gen.025x025.$YYYY.nc      WDL_DM
↪       2000-2013/1-12/01/0    C xy kg/m2/s   * - 1 1
111 GFED_AGW        $ROOT/GFED4/v2015-03/GFED4_gen.025x025.$YYYY.nc      AGW_DM
↪       2000-2013/1-12/01/0    C xy kg/m2/s   * - 1 1
111 GFED_DEF        $ROOT/GFED4/v2015-03/GFED4_gen.025x025.$YYYY.nc      DEF_DM
↪       2000-2013/1-12/01/0    C xy kg/m2/s   * - 1 1
111 GFED_FOR        $ROOT/GFED4/v2015-03/GFED4_gen.025x025.$YYYY.nc      FOR_DM
↪       2000-2013/1-12/01/0    C xy kg/m2/s   * - 1 1
111 GFED_PET        $ROOT/GFED4/v2015-03/GFED4_gen.025x025.$YYYY.nc      PET_DM
↪       2000-2013/1-12/01/0    C xy kg/m2/s   * - 1 1
111 GFED_SAV        $ROOT/GFED4/v2015-03/GFED4_gen.025x025.$YYYY.nc      SAV_DM
↪       2000-2013/1-12/01/0    C xy kg/m2/s   * - 1 1
)))GFED4

(((GFED_daily
111 GFED_FRAC_DAY   $ROOT/GFED3/v2014-10/GFED3_dailyfrac_gen.1x1.$YYYY.nc GFED3_BB__
↪DAYFRAC 2002-2011/1-12/1-31/0  C xy 1         * - 1 1
)))GFED_daily

(((GFED_3hourly
111 GFED_FRAC_3HOUR $ROOT/GFED3/v2014-10/GFED3_3hrfrac_gen.1x1.$YYYY.nc   GFED3_BB__
↪HRFRAC  2002-2011/1-12/01/0-23 C xy 1         * - 1 1
)))GFED_3hourly

### END SECTION BASE EMISSIONS ###
```

As in the previous examples, the tags beginning with `(((` and `)))` denote options that can be toggled on or off in the Extension Switches section. For example, if you wanted to use GFED3 biomass emissions instead of GFED4, you would set the switch for GFED3 to true and the switch for GFED4 to false.

Scale factors and other extension options (e.g. `Scaling_CO`) can be specified in the Extension Switches section.

Example 5: Tell HEMCO to use additional species

The HEMCO configuration file can hold emission specifications of as many species as desired. For example, to add anthropogenic NO emissions from the MACCity inventory, modify the HEMCO configuration file as shown:

```
###############################################################################
#### BEGIN SECTION BASE EMISSIONS
###############################################################################
#ExtNr Name srcFile srcVar srcTime CRE Dim Unit Species ScalIDs Cat Hier

(((MACCITY
0 MACCITY_CO $ROOT/MACCity.nc CO 1980-2014/1-12/1/0 C xy kg/m2/s CO 500 1 1
0 MACCITY_NO $ROOT/MACCity.nc NO 1980-2014/1-12/1/0 C xy kg/m2/s NO 500 1 1
)))MACCITY
```

To include NO in GFED, we can just add NO to the list of species that GFED will process in the Extension Switches section.

```
###############################################################################
#### BEGIN SECTION EXTENSION SWITCHES
###############################################################################
# ExtNr ExtName           on/off  Species
0       Base              : on    *
    --> MACCITY           :       true
    --> EMEP              :       true
    --> AEIC              :       true
#------------------------------------------------------------------------------
111     GFED              : on    CO/NO
    --> GFED3             :       false
    --> GFED4             :       true
    --> GFED_daily        :       false
    --> GFED_3hourly      :       false
    --> Scaling_CO        :       1.05
```

Finally, let's add sulfate emissions to the simulation. Emissions of SO4 are approximated from the MACCity SO2 data, assuming that SO4 constitutes 3.1% of the SO2 emissions. The final configuration file now looks like this:

```
###############################################################################
#### BEGIN SECTION SETTINGS
###############################################################################
ROOT:                         /dir/to/data
Logfile:                      HEMCO.log
DiagnFile:                    HEMCO_Diagn.rc
DiagnPrefix:                  HEMCO_diagnostics
Wildcard:                     *
Separator:                    /
Unit tolerance:               1
Negative values:              0
Only unitless scale factors: false
Verbose:                      0
Warnings:                     1

### END SECTION SETTINGS ###

###############################################################################
### BEGIN SECTION EXTENSION SWITCHES
###############################################################################
```

(continues on next page)

(continued from previous page)

```
# ExtNr ExtName           on/off  Species
0       Base              : on    *
    --> MACCITY           :       true
    --> EMEP              :       true
    --> AEIC              :       true
#------------------------------------------------------------------------------
111     GFED              : on    CO/NO/SO2
    --> GFED3             :       false
    --> GFED4             :       true
    --> GFED_daily        :       false
    --> GFED_3hourly      :       false
    --> Scaling_CO        :       1.05

### END SECTION EXTENSION SWITCHES ###

##############################################################################
#### BEGIN SECTION BASE EMISSIONS
##############################################################################
#ExtNr Name srcFile srcVar srcTime CRE Dim Unit Species ScalIDs Cat Hier
(((MACCITY
0 MACCITY_CO  $ROOT/MACCity.nc CO  1980-2014/1-12/1/0 C xy  kg/m2/s CO  500     1 1
0 MACCITY_NO  $ROOT/MACCity.nc NO  1980-2014/1-12/1/0 C xy  kg/m2/s NO  500     1 1
0 MACCITY_SO2 $ROOT/MACCity.nc SO2 1980-2014/1-12/1/0 C xy  kg/m2/s SO2 -       1 1
0 MACCITY_SO4 -                -   -                  - -   -       SO4 600     1 1
)))MACCITY

(((EMEP
0 EMEP_CO     $ROOT/EMEP.nc     CO 2000-2014/1-12/1/0 C xy  kg/m2/s CO 500/1001 1 2
)))EMEP

(((AEIC
0 AEIC_CO     $ROOT/AEIC.nc     CO 2005/1-12/1/0      C xyz kg/m2/s CO -        2 1
)))AEIC

##############################################################################
###  BEGIN SECTION EXTENSION DATA (subsection of BASE EMISSIONS SECTION
###
### These fields are needed by the extensions listed above. The assigned ExtNr
### must match the ExtNr entry in section 'Extension switches'. These fields
### are only read if the extension is enabled.  The fields are imported by the
### extensions by field name.  The name given here must match the name used
### in the extension's source code.
##############################################################################

# --- GFED biomass burning emissions (Extension 111) ---
111 GFED_HUMTROP    $ROOT/GFED3/v2014-10/GFED3_humtropmap.nc              humtrop
↪      2000/1/1/0             C xy 1         * - 1 1

(((GFED3
111 GFED_WDL        $ROOT/GFED3/v2014-10/GFED3_gen.1x1.$YYYY.nc           GFED3_BB__
↪WDL_DM  1997-2011/1-12/01/0    C xy kgDM/m2/s * - 1 1
111 GFED_AGW        $ROOT/GFED3/v2014-10/GFED3_gen.1x1.$YYYY.nc           GFED3_BB__
↪AGW_DM  1997-2011/1-12/01/0    C xy kgDM/m2/s * - 1 1
111 GFED_DEF        $ROOT/GFED3/v2014-10/GFED3_gen.1x1.$YYYY.nc           GFED3_BB__
↪DEF_DM  1997-2011/1-12/01/0    C xy kgDM/m2/s * - 1 1
111 GFED_FOR        $ROOT/GFED3/v2014-10/GFED3_gen.1x1.$YYYY.nc           GFED3_BB__
↪FOR_DM  1997-2011/1-12/01/0    C xy kgDM/m2/s * - 1 1
```

(continues on next page)

(continued from previous page)

```
111 GFED_PET         $ROOT/GFED3/v2014-10/GFED3_gen.1x1.$YYYY.nc           GFED3_BB__
↪PET_DM  1997-2011/1-12/01/0    C xy kgDM/m2/s * - 1 1
111 GFED_SAV         $ROOT/GFED3/v2014-10/GFED3_gen.1x1.$YYYY.nc           GFED3_BB__
↪SAV_DM  1997-2011/1-12/01/0    C xy kgDM/m2/s * - 1 1
)))GFED3

(((GFED4
111 GFED_WDL         $ROOT/GFED4/v2015-03/GFED4_gen.025x025.$YYYY.nc       WDL_DM
↪       2000-2013/1-12/01/0    C xy kg/m2/s   * - 1 1
111 GFED_AGW         $ROOT/GFED4/v2015-03/GFED4_gen.025x025.$YYYY.nc       AGW_DM
↪       2000-2013/1-12/01/0    C xy kg/m2/s   * - 1 1
111 GFED_DEF         $ROOT/GFED4/v2015-03/GFED4_gen.025x025.$YYYY.nc       DEF_DM
↪       2000-2013/1-12/01/0    C xy kg/m2/s   * - 1 1
111 GFED_FOR         $ROOT/GFED4/v2015-03/GFED4_gen.025x025.$YYYY.nc       FOR_DM
↪       2000-2013/1-12/01/0    C xy kg/m2/s   * - 1 1
111 GFED_PET         $ROOT/GFED4/v2015-03/GFED4_gen.025x025.$YYYY.nc       PET_DM
↪       2000-2013/1-12/01/0    C xy kg/m2/s   * - 1 1
111 GFED_SAV         $ROOT/GFED4/v2015-03/GFED4_gen.025x025.$YYYY.nc       SAV_DM
↪       2000-2013/1-12/01/0    C xy kg/m2/s   * - 1 1
)))GFED4

(((GFED_daily
111 GFED_FRAC_DAY    $ROOT/GFED3/v2014-10/GFED3_dailyfrac_gen.1x1.$YYYY.nc GFED3_BB__
↪DAYFRAC 2002-2011/1-12/1-31/0  C xy 1         * - 1 1
)))GFED_daily

(((GFED_3hourly
111 GFED_FRAC_3HOUR $ROOT/GFED3/v2014-10/GFED3_3hrfrac_gen.1x1.$YYYY.nc   GFED3_BB__
↪HRFRAC  2002-2011/1-12/01/0-23 C xy 1         * - 1 1
)))GFED_3hourly

### END SECTION BASE EMISSIONS ###

################################################################################
#### BEGIN SECTION SCALE FACTORS
################################################################################
# ScalID Name srcFile srcVar srcTime CRE Dim Unit Oper

500 HOURLY_SCALFACT $ROOT/hourly.nc factor  2000/1/1/0-23 C xy 1 1
600 SO2toSO4        0.031           -       -             - -  1 1

### END SECTION SCALE FACTORS ###

################################################################################
#### BEGIN SECTION MASKS
################################################################################
#ScalID Name srcFile srcVar srcTime CRE Dim Unit Oper Box

1001 MASK_EUROPE $ROOT/mask_europe.nc MASK 2000/1/1/0 C xy 1 1 -30/30/45/70

### END SECTION MASKS ###
```

Example 6: Add inventories that do not separate out biofuels and/or trash emissions

Several emissions inventories (e.g. CEDS and EDGAR) lump biofuels and/or and trash emissions together with anthropogenic emissions. For inventories such as these, HEMCO allows you to specify up to 3 multiple categories for each species listing in the HEMCO configuration file. All of the emissions will go into the first listed category, and the other listed categories will be set to zero.

In this example, all NO emissions from the EDGAR inventory power sector will be placed into the the anthropogenic emissions category (`Cat=1`), while the biofuel emissions category (Cat=2) will be set to zero.

```
0 EDGAR_NO_POW EDGAR_v43.NOx.POW.0.1x0.1.nc emi_nox 1970-2010/1/1/0 C xy kg/m2/s NO␣
↪1201/25/115  1/2  2
```

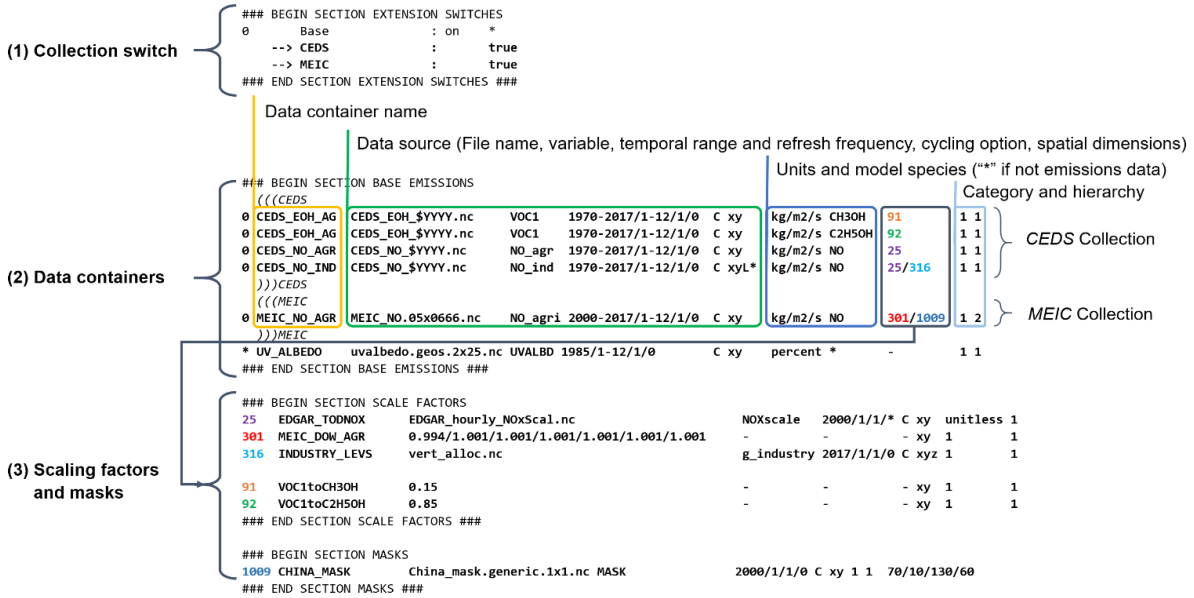
In this example, all NO emissions from CEDS inventory agriculture sector will be placed into the the anthropogenic emissions category (`Cat=1`), while the biofuel emissions category (`Cat=2`) and trash emissions category (`Cat=12`) will be set to zero.

```
0 CEDS_NO_AGR NO-em-anthro_CMIP_CEDS_$YYYY.nc  NO_agr 1750-2014/1-12/1/0 C xy kg/m2/s␣
↪NO  25 1/2/12 5
```

2.1.2 The HEMCO configuration file

The HEMCO Configuration file is composed of several sections: *Settings*, *Base Emissions*, *Scale Factors,*, and *Masks*.

An overview of the structure and key formats of the HEMCO configuration file can be found in Figure 2 of Lin *et al.* [[Lin et al., 2021]]:



Settings

Parameters and variables used by HEMCO are defined in between these comment lines:

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

settings go here

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

The order within the settings section is irrelevant. Many of these settings are optional, and default values will be used if not set.

General simulation settings

These settings control HEMCO simulation options.

ROOT
Root folder containing emissions inventories and other data to be read by HEMCO.

METDIR
Root folder of meteorology data files that are needed for HEMCO extensions. Usually this is a subdirectory of *ROOT*.

MODEL
If present, the `$MODEL` token will be set to the value specified.

If omitted, this value is determined based on compiler switches.

RES
If present, the `$RES` token will be set to the value specified.

If omitted, this value is determined based on compiler switches.

LogFile
Path and name of the output log file (which is typically named `HEMCO.log`). If set to the *Wildcard* character, all HEMCO output is written to **stdout** (i.e. the screen).

Unit `tolerance`
Integer value denoting the tolerance against differences between the units set in the *HEMCO configuration file* and data units found in the source file. Allowable values are"

0
No tolerance. A units mismatch will halt a HEMCO simulation. mismatch).

1
Medium tolerance. A units mismatch will print a warning message but not halt a HEMCO simulation. **(Default setting)**

2
High tolerance. A units mismatch will be ignored.

Negative `values`
Integer value that defines how negative values are handled.

0
No negative values are allowed. **(Default setting)**

1
: All negative values are set to zero and a warning is given.

2
: Negative values are kept as they are.

Verbose
: Integer value that controls the amount of additional information printed to the HEMCO log file. Allowable values are `0` (no additional output) to `3` (lots of additional output). Setting `3` is useful for debugging.

  Default setting: `0`.

Warnings
: Integer value that controls the amount of warnings printed to the HEMCO log file. Allowable values are `0` (no warnings) to `3` (all warnings).

  Default setting: `1` (only severe warnings).

Wildcard
: Wildcard character. On Unix/Linux, this should be set to `*`.

Separator
: Separator symbol. On Unix/4Linux systems, this should be set to `/`.

Mask `fractions`
: If `true`, the fractional mask values are taken into account. This means that mask values can take any value between 0.0 and 1.0.

  If `false`, masks are binary, and grid boxes are 100% inside or outside of a mask region.

  Default setting: `false`

PBL `dry deposition`
: If `true`, it is assumed that dry deposition occurs over the entire boundary layer. In this case, extensions that include loss terms (e.g. air-sea exchange) will calculate a loss term for every grid box that is partly within the planetary boundary layer.

  If `false`, a loss term is calculated for the surface layer only.

  Default setting: `false`

Emissions settings

The following options can be used to hold emissions constant over a year, month, day, or hour, and to scale emissions to a given value:

Emission `year`
: If present, this emission year will be used regardless of the model simulation year.

  If omitted, the emission year will be set to the model simulation year.

Emission `month`
: If present, this emission month will be used regardless of the model simulation month.

  If omitted, the emission month will be set to the model simulation month.

Emission `day`
: If present, this emission day will be used regardless of the model simulation day.

  If omitted, the emission day will be set to the model simulation day.

Emission hour
If present, this emission month will be used regardless of the model simulation hour.

If omitted, the emisison month will be set to the model simulation hour.

EmissScale_<species-name>
Optional argument to define a uniform scale factor that will be applied across all inventories, categories, hierarchies, and extensions. Can be set for every species individually, e.g.

```
EmisScale_NO: 1.5
EmisScale_CO: 2.0
```

Scales all NO emissions by 50% and doubles CO emissions.

Diagnostics settings

The following options control archival of diagnostic quantities. For more information about HEMCO diagnostics, please see the *HEMCO diagnostics* section.

DiagnFile
Specifies the configuration file for the HEMCO default diagnostics collection. This is usually named `HEMCO_Diagn.rc`. This file contains a list of fields to be added to the default collection.

Each line of the diagnostics definition file represents a diagnostics container. It expects the following 7 entries (all on the same line):

1. Container name (character)
2. HEMCO species (character)
3. Extension number (integer)
4. Emission category (integer)
5. Emission hierarchy (integer)
6. Space dimension (2 or 3)
7. Output unit (character)
8. Long name of diagnostic, for the netCDF `long_name` variable attribute (character)

Note: If you are not sure what the container name, extension number, category, and hierarchy are for a given diagnostic, you can set `Verbose` to 3 in the HEMCO configuration file, and run a very short simulation (a couple of model hours). Then you can look at the output in the `HEMCO.log` file to determine what these values should be.

Please see the *Default diagnostics collection* section for more information about the configuration file (e.g. `HEMCO_Diagn.rc`).

DiagnFreq
This setting (located in the HEMCO configuration file) specifies the output frequency of the *Default* collection. Allowable values are:

Always
Archives diagnostics on each time step.

Hourly
Sets the diagnostic time period to 1 hour.

Daily
Sets the diagnostic time period to 1 day.

Monthly
Sets the diagnostic time period to 1 hour.

Annually
Sets the diagnostic time period to 1 year.

End
Sets the diagnostic time period so that output will only happen at the end of the simulation.

YYYYMMDD `hhmnss`
Sets the diagnostic time period to an interval specified by a 15-digit string with year-month-day, hour-minute-second. For example:

- `00010000 000000` will generate diagnostic output once per year.
- `00000001 000000` will generate diagnostic output once per day.
- `00000000 020000` will generate diagnostic output every 2 hours.
- etc.

DiagnPrefix
Specifies the name of the diagnostic files to be created. For example:

```
DiagnPrefix: ./OutputDir/HEMCO_diagnostics
```

will create HEMCO diagnostics files in the `OutputDir/` subdirectory of the run directory, and all files will begin with the text `HEMCO_diagnostics`.

DiagnRefTime
This option must be explicity added to the HEMCO configuration file.

By default, the value of the `time:units` attribute in the `HEMCO_diagnostics.*.nc` files will be `hours since YYYY-MM-DD hh:mn:ss`, where `YYYY-MM-DD hh:mn:ss` is the diagnostics datetime. This default value can be overridden and set to a fixed datetime by setting *DiagnRefTime* in the HEMCO configuration file. For example:

```
DiagnRefTime: hours since 1985-01-01 00:00:00
```

will set the `time:units` attribute to `hours since 1985-01-01 00:00:00`.

DiagNoLevDim
This option must be explicity added to the HEMCO configuration file. If omitted, the default behavior will be `false`.

If `true`, the created `HEMCO_diagnostics*.nc` files will contain dimensions `(time,lat,lon)`. But if at least one of the diagnostic quantities has a `lev` dimension, then the created files will have `(time,lev,lat,lon)` dimensions.

If `false`, the `HEMCO_diagnostics.*.nc` files will always contain dimensions `(time,lev,lat,lon)`.

DiagnTimeStamp
This option must be explicity added to the HEMCO configuration file. If omitted, the default behavior will be *End*.

Allowable values are:

End
Uses the date and time at the end of the diagnostics time window to timestamp diagnostic files. With this

option, a 1-hour simulation from `20220101 000000` to `20220101 010000` will create a diagnostic file named `HEMCO_Diagnostics.202201010100.nc`.

Start
: Uses the date and time at the start of the diagnostics time window to timestamp diagnostic files. With this option, a 1-hour simulation from `20220101 000000` to `20220101 010000` will create a diagnostic file named `HEMCO_Diagnostics.202201010000.nc`.

Mid
: Uses the date and time at the midpoint of the diagnostics time window to timestamp diagnostic files. With this option, a 1-hour simulation from `20220101 000000` to `20220101 010000` will create a diagnostic file named `HEMCO_diagnostics.202201010030.nc`.

HEMCO standalone simulation settings

In standalone mode, the three simulation description files also need be specified:

GridFile
: Path and name of the grid description file, which is usually named `HEMCO_sa_Grid.rc`.

SpecFile
: Path and name of the species description file, which is usually named `HEMCO_sa_Spec.rc`.

GridFile
: Path and name of the time description file, which is usually named `HEMCO_sa_Time.rc`.

User-defined tokens

Users can specify any additional token in the **Settings** section section. The token name/value pair must be separated by the colon (:) sign. For example, adding the following line to the settings section would register token `$ENS` (and assign value 3 to it):

```
ENS: 3
```

User-defined tokens can be used the same way as the built-in tokens (`$ROOT`, `$RES`, `YYYY`, etc.). See `sourceFile` in the Base emissions for more details about tokens.

Important: User-defined token names must not contain numbers or special characters such as `.`, `_`, `-`, or `x`.

Extension switches

HEMCO performs automatic emission calculations using all fields that belong to the *base emisisons extension*. Additional emissions that depend on environmental parameter such as wind speed or air temperature–and/or that use non-linear parameterizations–are calculated through *HEMCO extensions*. A list of currently implemented extensions in HEMCO is given in *Keller et al. (2014)*. To add new extensions to HEMCO, modifications of the source code are required, as described further in *HEMCO under the hood*.

The first section of the configuration file lists all available extensions and whether they shall be used or not. For each extension, the following attributes need to be specified:

ExtNr
: Extension number associated with this field. All *base emissions* should have extension number zero. The extension number` of the data listed in section *HEMCO extensions* data must match with the corresponding extension number.

The extension number can be set to the wildcard character. In that case, the field is read by HEMCO (if the assigned species name matches any of the HEMCO species, see *Species* below) but not used for emission calculation. This is particularly useful if HEMCO is only used for data I/O but not for emission calculation.

ExtName
Extension name.

Toggle
If `on`, the extension will be used.

If `off`, the extension will not be used.

Species
List of species to be used by this extension. Multiple species are separated by the *Separator* symbol (e.g. `/`). All listed species must be supported by the given extension.

- For example, the soil NO emissions extension only supports one species (NO). An error will be raised if additional species are listed.

Additional extension-specific settings can also be specified in the 'Extensions Settings' section (see also an example in *Basic examples* and the definition of *Data collections*. These settings must immediately follow the extension definition.

HEMCO expects an extension with extension number zero, denoted the *base emisisons extension* extension. All emission fields linked to the base extension will be used for automatic emission calculation. Fields assigned to any other extension number will not be inlcuded in the base emissions calculation, but they are still read/regridded by HEMCO (and can be made available readily anywhere in the model code). These data are only read if the corresponding extension is enabled.

All species to be used by HEMCO must be listed in column *Species* of the base extension switch. In particular, all species used by any of the other extensions must also be listed as base species, otherwise they will not be recognized. It is possible (and recommended) to use the *Wildcard* character, in which case HEMCO automatically determines what species to use by matching the atmospheric model species names with the species names assigned to the base emission fields and/or any emission extension.

The environmental fields (wind speed, temperature, etc.) required by the extensions are either passed from the atmospheric model or read through the HEMCO configuration file, as described in *HEMCO extensions*.

Base emissions

The BASE EMISSIONS section lists all base emission fields and how they are linked to *scale factors*. Base emissions settings must be included between these comment lines:

```
###############################################################################
### BEGIN SECTION BASE EMISSIONS
###############################################################################
settings go here

### END SECTION BASE EMISSIONS ###
```

The *ExtNr* field is defined in *Extension switches*.

Other attributes that need to be defined for each base emissions entry are:

Name
Descriptive field identification name. Two consecutive underscore characters (__) can be used to attach a 'tag' to a name. This is only of relevance if multiple base emission fields share the same species, category, hierarchy, and scale factors. In this case, emission calculation can be optimized by assigning field names that onlydiffer by its tag to those fields (e.g. `DATA__SECTOR1`, `DATA__SECTOR2`, etc.).

For fields assigned to extensions other than the base extension (`ExtNr = 0`), the field names are prescribed and must not be modified because the data is identified by these extensions by name.

sourceFile
Path and name of the input file.

Name tokens can be provided that become evaluated during runtime. For example, to use the root directory specified in the *Section settings section*, the `$ROOT` token can be used. Similarly the token `$CFDIR` refers to the location of the configuration file. This allows users to reference data relative to the location of the configuration file. For instance, if the data is located in subfolder `data` of the same directory as the configuration file, the file name can be set to `$CFDIR/data/filename.nc`.

Similarly, the **date tokens** `$YYYY`, `$MM`, `$DD`, `$HH`, and `$MN` can be used to refer to the the current valid year, month, day, hour, and minute, respectively. These values are determined from the current simulation datetime and the *sourceTime* specification for this entry.

The tokens `$MODEL` and `$RES` refer to the meteorological model (*MODEL*) and resolution (*RES*). These tokens can be set explicitly in the settings section. In GEOS-Chem they are set to compiler-flag specific values if not set in the settings section. Any token defined in the settings section can be used to construct a part of the file name (see *User-defined tokens*).

As an alternative to an input file, **geospatial uniform values** can directly be specified in the configuration file (see e.g. scale factor `SO2toSO4` in *Basic examples*). If multiple values are provided (separated by the separator character), they are interpreted as different time slices. In this case, the *sourceTime* attribute can be used to specify the times associated with the individual slices. If no time attribute is set, HEMCO attempts to determine the time slices from the number of data values: 7 values are interpreted as weekday (Sun, Mon, ..., Sat); 12 values as month (Jan, ..., Dec); 24 values as hour-of-day (12am, 1am, ..., 11pm).

Uniform values can be combined with **mathematical expressions**, e.g. to model a sine-wave emission source. Mathematical expressions must be labeled `MATH:`, followed by the expression, e.g. `MATH:2.0+sin(HH/12*PI)`.

Country-specific data can be provided through an ASCII file (`.txt`). More details on this option are given in the Input File Format section.

If this entry is **left empty** (-), the filename from the preceding entry is taken, and the next 5 attributes will be ignored (see entry `MACCITY_SO4` in *Basic examples*.

sourceVar
Source file variable of interest. Leave empty (-) if values are directly set through the *sourceFile* attribute or if *sourceFile* is empty.

sourceTime
This attribute defines the time slices to be used and the data refresh frequency. The format is `year/month/day/hour`. Accepted are discrete dates for time-independent data (e.g. `2000/1/1/0`) and time ranges for temporally changing fields (e.g. `1980-2007/1-12/1-31/0-23`). Data will automatically become updated as soon as the simulation date enters a new time interval.

The provided time attribute determines the data refresh frequency. It does not need to correspond to the datetimes of the input file.

- For example, if the input file contains daily data of year 2005 and the time attribute is set to `2005/1/1/0`, the file will be read just once (at the beginning of the simulation) and the data of Jan 1, 2005 is used throughout the simulation.
- If the time attribute is set to `2005/1-12/1/0`, the data is updated on every month, using the first day data of the given month. For instance, if the simulation starts on July 15, the data of July 1,2005 are used until August 1, at which point the data will be refreshed to values from August 1, 2005.
- A time attribute of `2005/1-12/1-31/0` will make sure that the input data are refreshed daily to the current day's data.

- Finally, if the time attribute is set to `2005/1-12/1-31/0-23`, the data file is read every simulation hour, but the same daily data is used throughout the day (since there are no hourly data in the file). Providing too high update frequencies is not recommended unless the data interpolation option is enabled (see below).

If the provided time attributes do not match a datetime of the input file, the **most likely** time slice is selected. The most likely time slice is determined based on the specified source time attribute, the datetimes available in the input file, and the current simulation date. In most cases, this is just the closest available time slice that lies in the past.

- For example, if a file contains annual data from 2005 to 2010 and the source time attribute is set to `2005-2010/1-12/1/0`, the data of 2005 is used for all simulation months in 2005.
- More complex datetime selections occur for files with discontinuous time slices, e.g. a file with monthly data for year 2005, 2010, 2020, and 2050. In this case, if the time attribute is set to `2005-2020/1-12/1/0`, the monthly values of 2005 are (re-)used for all years between 2005 and 2010, the monthly values of 2010 are used for simulation years 2010 - 2020, etc.

It is possible to use tokens `$YYYY`, `$MM`, `$DD`, and `$HH`, which will automatically be replaced by the current simulation date. Weekly data (e.g. data changing by the day of the week) can be indicated by setting the day attribute to `WD` (the wildcard character will work, too, but is not recommended). Weekly data needs to consist of at least seven time slices - in increments of one day - representing data for every weekday starting on Sunday. It is possible to store multiple weekly data, e.g. for every month of a year: `2000/1-12/WD/0`. These data must contain time slices for the first seven days of every month, with the first day per month representing Sunday data, then followed by Monday, etc. (irrespective of the real weekdays of the given month). If the wildcard character is used for the days, the data will be interpreted if (and only if) there are exactly seven time slices. See the Input File Format section for more details. Default behavior is to interpret weekly data as 'local time', i.e. token `WD` assumes that the provided values are in local time. It is possible to use weekly data referenced to UTC time using token `UTCWD`.

Similar to the weekday option, there is an option to indicate hourly data that represents local time: `LH`. If using this flag, all hourly data of a given time interval (day, month, year) are read into memory and the local hour is picked at every location. A downside of this is that all hourly time slices in memory are updated based on UTC time. For instance, if a file holds local hourly data for every day of the year, the source time attribute can be set to `2011/1-12/1-31/LH`. On every new day (according to UTC time), this will read all 24 hourly time slices of that UTC day and use those hourly data for the next 24 hours. For the US, for instance, this results in the wrong daily data being used for the last 6-9 hours of the day (when UTC time is one day ahead of local US time).

There is a difference between source time attributes `2005-2008/$MM/1/0` and `2005-2008/1-12/1/0`. In the first case, the file will be updated annually, while the update frequency is monthly in the second case. The token `$MM` simply indicates that the current simulation month shall be used whenever the file is updated, but it doesn't imply a refresh interval. Thus, if the source time attribute is set to `$YYYY/$MM/$DD/$HH`, the file will be read only once and the data of the simulation start date is taken (and used throughout the simulation). For uniform values directly set in the configuration file, all time attributes but one must be fixed, e.g. valid entries are `1990-2007/1/1/0` or `2000/1-12/1/1`, but not `1990-2007/1-12/1/1`.

Note: All data read from netCDF file are assumed to be in UTC time, except for weekday data that are always assumed to be in local time. Data read from the configuration file and/or from ASCII are always assumed to be in local time.

It is legal to keep different time slices in different files, e.g. monthly data of multiple years can be stored in files `file_200501.nc`, `file_200502.nc`, ..., `file_200712.nc`. By setting the source file attribute to `file_$YYYY$MM.nc` and the source time attribute to `2005-2007/1-12/1/0`, data of `file_200501.nc` is used for simulation dates of January 2005 (or any January of a previous year), etc. The individual files can also contain only a subset of the provided data range, e.g. all monthly files of a year can be stored in one file: `file_2005.nc`, `file_2006.nc`, `file_2007.nc`. In this case, the source file name should be set to

`file_$YYYY`, but the source time attribute should still be `2005-2007/1-12/1/0` to indicate that the field shall be updated monthly.

This attribute can be set to the wildcard character (`*`), which will force the file to be updated on every HEMCO time step.

File reference time can be shifted by a fixed amount by adding an optional fifth element to the time stamp attribute. For instance, consider the case where 3-hourly averages are provided in individual files with centered time stamps, e.g.: `file.yyyymmdd_0130z.nc`, `file.yyyymmdd_0430z.nc`, ..., `file.yyymmdd_2230z.nc`. To read these files **at the beginning of their time intervals**, the time stamp can be shifted by 90 minutes: `2000-2016/1-12/1-31/0-23/+90minutes`. At time 00z, HEMCO will then read file 0130z and keep using this file until 03z, when it switches to file 0430z. Similarly, it is possible to shift the file reference time by any number of years, months, days, or hours. Time shifts can be forward or backward in time (use − sign to shift backwards).

CRE

Controls the time slice selection if the simulation date is outside the range provided in attribute source time (see above). The following options are available:

C

Cycling: Data are interpreted asclimatology and recycled once the end of the last time slice is reached. For instance, if the input data contains monthly data of year 2000, and the source time attribute is set to `2000/1-12/1/0 C`, the same monthly data will be re-used every year.

If the input data spans multiple years (e.g. monthly data from 2000-2003), the closest available year will be used outside of the available range (e.g. the monthly data of 2003 is used for all simulation years after 2003).

CS

Cycling, Skip: Data are interpreted as climatology and recycled once the end of the last time slice is reached. Data that aren't found are skipped. This is useful when certain fields aren't found in a restart file and, in that case, those fields will be initialized to default values.

CY

Cycling, Use Simulation Year:, Same as *C*, except don't allow `Emission year` setting to override year value.

CYS

Cycling, Use Simulation Year, Skip: Same as *CS*, except don't allow `Emission year` setting to override year value.

R

Range: Data are only considered as long as the simulation time is within the time range specified in attribute *sourceTime*. The provided range does not necessarily need to match the time stamps of the input file. If it is outside of the range of the netCDF time stamps, the closest available date will be used.

For instance, if a file contains data for years 2003 to 2010 and the provided range is set to `2006-2010/1/1/0 R`, the file will only be considered between simulation years 2006-2010. For simulation years 2006 through 2009, the corresponding field on the file is used. For all years beyond 2009, data of year 2010 is used. If the simulation date is outside the provided time range, the data is ignored but HEMCO does not return an error - the field is simply treated as empty (a corresponding warning is issued in the HEMCO log file).

- Example: if the source time attribute is set to `2000-2002/1-12/1/0 R`, the data will be used for simulation years 2000 to 2002 and ignored for all other years.

RA

Range, Averaging Otherwise: Combination of flags *R* and *A*. As long as the simulation year is within the specified year range, HEMCO will use just the data from that particular year. As soon as the simulation year is outside the specified year range, HEMCO will use the data averaged over the specified years.

- Consider the case where the emission file contains monthly data for years 2005-2010. Setting the time attribute to `2005-2010/1-12/1/0 R` will ensure that this data is only used within simulation years 2005 to 2010 and ignored outside of it.
- When setting the time attribute to `2005-2010/1-12/1/0 A`, HEMCO will always use the 2005-2010 averaged monthly values, even for simulation years 2005 to 2010.
- A time attribute of `2005-2010/1-12/1/0 RA` will make sure that HEMCO uses the monthly data of the current year if the simulation year is between 2005 and 2010, and the 2005-2010 average for simulation years before and after 2005 and 2010, respectively.

RF

Range, Forced: Same as `R`, but HEMCO stops with an error if the simulation date is outside the provided range.

RY

Range, Use Simulation Year: Same as *R*, except don't allow `Emission year` to override year value.

E

Exact: Fields are only used if the time stamp on the field exactly matches the current simulation datetime. In all other cases, data is ignored but HEMCO does not return an error.

- For example, if *sourceTime* is set to `2000-2013/1-12/1-31/0 E`, every time the simulation enters a new day HEMCO will attempt to find a data field for the current simulation date. If no such field can be found on the file, the data is ignored (and a warning is prompted). This setting is particularly useful for data that is highly sensitive to date and time, e.g. restart variables.

EF

Exact, Forced: Same as *E*, but HEMCO stops with an error if no data field can be found for the current simulation date and time.

EC

Exact, Read/Query Contiuously..

ECF

Exact, Read/Query Continuously, Forced.

EFYO

Exact, Forced, Simulation Year, Once: Same as *EF*, with the following additions:

- `Y`: HEMCO will stop thie simulation if the simulation year does not match the year in the file timestamp.
- `O`: HEMCO will only read the file once.

This setting is typically only used for model restart files (such as GEOS-Chem Classic restart files). This ensures that the simulation will stop unless the restart file timestamp matches the simulation start date and time.

> **Attention:** Consider changing the time cycle flag from *EFYO* to *CYS* if you would like your simulation to read a data file (such as a simulation restart file) whose file timestamp differs from the simulaton start date and time.

EY

Exact, Use Smulation Year: Same as *E*, except don't allow `Emission year` setting to override year value.

A

Averaging: Tells HEMCO to average the data over the specified range of years.

- For instance, setting *sourceTime* to `1990-2010/1-12/1/0 A` will cause HEMCO to calculate monthly means between 1990 to 2010 and use these regardless of the current simulation date.

The data from the different years can be spread out over multiple files. For example, it is legal to use the averaging flag in combination with files that use year tokens such as `file_$YYYY.nc`.

I

Interpolation: Data fields are interpolated in time. As an example, let's assume a file contains annual data for years 2005, 2010, 2020, and 2050. If *sourceTime* is set to `2005-2050/1/1/0 I`, data becomes interpolated between the two closest years every time we enter a new simulation year. If the simulation starts on January 2004, he value of 2005 is used for years 2004 and 2005. At the beginning of 2006, the used data is calculated as a weighted mean for the 2005 and 2010 data, with 0.8 weight given to 2005 and 0.2 weight given to 2010 values. Once the simulation year changes to 2007, the weights hange to 0.6 for 2005 and 0.4 for 2010, etc. The interpolation frequency is determined by *sourceTime* the source time attribute.

For example, setting the source time attribute to `2005-2050/1-12/1/0 I` would result in a recalculation of the weights on every new simulation month. Interpolation works in a very similar manner for discontinuous monthly,daily, and hourly data. For instance if a file contains monthly data of 2005, 2010, 2020, and 2050 and the source time attribute is set to `2005-2050/1-12/1/0 I`, the field is recalculated every month using the two bracketing fields of the given month: July 2007 values are calculated from July 2005 and July 2010 data (with weights of 0.6 and 0.4, respectively), etc.

Data interpolation also works between multiple files. For instance, if monthly data are stored in files :literal`file_200501.nc`, `file_200502.nc`, etc., a combination of source file name `file_$YYYY$MM.nc` and *sourceTime* attribute `2005-2007/1-12/1-31/0 :literal:I` will result in daily data interpolation between the two bracketing files, e.g. if the simulation day is July 15, 2005, the fields current values are calculated from files `file_200507.nc` and `file_200508.nc`, respectively.

Data interpolation across multiple files also works if there are file 'gaps', for example if there is a file only every three hours: `file_20120101_0000.nc`, `file_20120101_0300.nc`, etc. Hourly data interpolation between those files can be achieved by setting source file to :file:file_\protect\T1\textdollarYYYY\protect\T1\textdollarMM\protect\T1\textdollarDD_\protect\T1\textdollarHH00.nc`, and *sourceTime* to `2000-2015/1-12/1-31/0-23 I` (or whatever the covered year range is).

SrcDim

Spatial dimension of input data (`xy` for horizontal data; `xyz` for 3-dimensional data).

The *SrcDim* attribute accepts an integer number as vertical coordinate to indicate the number of vertical levels to be read, as well as the direction of the vertical axis. For example, to use the lowest 5 levels of the input data only, set *SrcDim* to `xy5`. This will place the lowest 5 levels of the input data into HEMCO levels 1 to 5. To use the topmost 5 levels of the input data, set *SrcDim* to `xy-5`. The minus sign will force the vertical axis to be flipped, i.e. the 5 topmost levels will be placed into HEMCO levels 1 to 5 (in reversed order, so that the topmost level of the input data will be placed in HEMCO lev el 1, etc.).

The *SrcDim* attribute can also be used to indicate the level into which 2D data shall be released by setting the vertical coordinate to *:literal:`LX``*, with `X` being the release level. For instance, to emit a 2D field into level 5, set *SrcDim* to `xyL5`.

HEMCO can has two options to specify the emission injection height:

1. The vertical height can be given as model level (default) or in meters, e.g. to emit a source at 2000m: `xyL=2000m`.
2. For 2D fields it is legal to define a range of levels, in which case the emissions are uniformly distributed across these levels (maintaining the original total emissions). Examples for this are:
   - `xyL=1:5`: Emit into levels 1-5;
   - `xyL=2:5000m` Emit between model level 2 and 5000m;

- `xyL=1:PBL`: Emit from the surface up to the PBL top.

HEMCO can also get the injection height information from an external source (i.e. netCDF file). For now, these heights are expected to be in meters. The injection height data must be listed as a scale factor and can then be referenced in the *SrcDim* setting.

HEMCO can read read netCDF files with an arbitrary additional dimension. For these files, the name of the additional dimension and the desired dimension index must be specified as part of the *SrcDim* attribute.

- For example, to read a file that contains 3D ensemble data (with the individual ensemble runs as additional dimension `ensemble`), set *SrcDim* to `xyz+"ensemble=3` to indicate that you wish to read the third ensemble member. You may also use a *user-defined token* for the dimension index to be used, e.g. `xyz+"ensemble=$ENS"`.

Note: Arbitrary additional dimensions are currently not supported in a high-performance environment that uses the ESMF/MAPL input/output libraries.

SrcUnit
Units of the data.

Species
HEMCO emission species name. Emissions will be added to this species. All HEMCO emission species are defined at the beginning of the simulation (see the Interfaces section) If the species name does not match any of the HEMCO species, the field is ignored altogether.

The species name can be set to the wildcard character, in which case the field is always read by HEMCO but no species is assigned to it. This can be useful for extensions that import some (species-independent) fields by name.

The three entries below only take effect for fields that are assigned to the base extension (`ExtNr = 0`), e.g. that are used for automatic emission calculation. They are used by HEMCO to determine how the final emission fields are assembled from all provided data fields.

ScalIDs
Identification numbers of all scale factors and masks that shall be applied to this base emission field. Multiple entries must be separated by the separator character. The *ScalIDs* must correspond to the numbers provided in the *Scale factors* and *Masks* sections.

Cat
Emission category. Used to distinguish different, independent emission sources. Emissions of different categories are always added to each other.

Up to three emission categories can be assigned to each entry (separated by the separator character). Emissions are always entirely written into the first listed category, while emissions of zero are used for any other assigned category.

In practice, the only time when more than one emissions category needs to be specified is when an *inventory does not separate between anthropogenic, biofuels, and/or trash emissions*

For example, the CEDS inventory uses categories `1/2/12` because CEDS lumps both biofuel emissions and trash emissions with anthropogenic Because. The `1/2/12` category designation means "Put everything into the first listed category (1=anthropogenic), and set the other listed categories (2=biofuels, 12=trash) to zero.

Hier
Emission hierarchy. Used to prioritize emission fields within the same emission category. Emissions of higher hierarchy overwrite lower hierarchy data. Fields are only considered within their defined domain, i.e. regional inventories are only considered within their mask boundaries.

Scale factors

The SCALE FACTORS section of the configuration file lists all scale factors applied to the base emission field. Scale factors that are not used by any of the base emission fields are ignored. Scale factors can represent:

1. Temporal emission variations including diurnal, seasonal, or interannual variability;
2. Regional masks that restrict the applicability of the base inventory to a given region; or
3. Species-specific scale factors, e.g., to split lumped organic compound emissions into individual species.

This sample snippet of the HEMCO configuration file shows how scale factors can either be read from a netCDF file or listed as a set of values.

```
############################################################################
### BEGIN SECTION SCALE FACTORS
############################################################################
# ScalID Name srcFile srcVar srcTime CRE Dim Unit Oper

# %%% Hourly factors, read from disk %%%
1 HOURLY_SCALFACT hourly.nc                                          factor 2000/1/1/0-23␣
↪C xy 1 1

# %%% Scaling SO2 to SO4 (molar ratio) %%%
2 SO2toSO4        0.031                                              -      -           ␣
↪- -  1 1

# %%% Daily scale factors, list 7 entries %%%
20 GEIA_DOW_NOX   0.784/1.0706/1.0706/1.0706/1.0706/1.0706/0.863 -      -           ␣
↪- xy 1 1

### END SECTION SCALE FACTORS ###
```

Options *sourceFile*, *sourceVar*, *sourceTime*, *CRE*, *SrcDim*, and *SrcUnit*, are described in *Base emissions*.

Other scale factor options not previously described are:

Scale factor options not previously described are:

ScalID
: Scale factor identification number. Used to link the scale factors to the base emissions through the corresponding ScalIDs attribute in the :ref`hco-cfg-base`.

Oper
: Scale factor operator. Determines the operation performed on the scale factor. Possible values are:

  - 1 for multiplication (Emission = Base * Scale);
  - -1 for division (Emission = Base / Scale);
  - 2 for squared (Emission = Base * Scale**2).

MaskID
: Optional. ScalID of a mask field. This optional value can be used if a scale factor shall only be used over a given region. The provided MaskID must have a corresponding entry in the *Masks section* of the configuration file.

Note: Scale factors are assumed to be `unitless` (aka `1`) and no automatic unit conversion is performed.

Masks

This section lists all masks used by HEMCO. Masks are binary scale factors (1 inside the mask region, 0 outside). If masks are regridded, the remapped mask values (1 and 0) are determined through regular rounding, i.e. a remapped mask value of 0.49 will be set to 0 while 0.5 will be set to 1.

The MASKS section in the HEMCO configuration file will look similar to this (it will vary depending on the type of GEOS-Chem simulation you are using):

```
###############################################################################
### BEGIN SECTION MASKS
###############################################################################
# ScalID Name sourceFile sourceVar sourceTime CRE SrcDim SrcUnit Oper Lon1/Lat1/Lon2/
↪Lat2

#==============================================================================
# Country/region masks
#==============================================================================
1000 EMEP_MASK   EMEP_mask.geos.1x1.20151222.nc           MASK     2000/1/1/0 C xy␣
↪unitless 1 -30/30/45/70
1002 CANADA_MASK Canada_mask.geos.1x1.nc                  MASK     2000/1/1/0 C xy␣
↪unitless 1 -141/40/-52/85
1003 SEASIA_MASK SE_Asia_mask.generic.1x1.nc              MASK     2000/1/1/0 C xy␣
↪unitless 1  60/-12/153/55
1004 NA_MASK     NA_mask.geos.1x1.nc                      MASK     2000/1/1/0 C xy␣
↪unitless 1 -165/10/-40/90
1005 USA_MASK    usa.mask.nei2005.geos.1x1.nc             MASK     2000/1/1/0 C xy␣
↪unitless 1 -165/10/-40/90
1006 ASIA_MASK   MIX_Asia_mask.generic.025x025.nc         MASK     2000/1/1/0 C xy␣
↪unitless 1 46/-12/180/82
1007 NEI11_MASK  USA_LANDMASK_NEI2011_0.1x0.1.20160921.nc LANDMASK 2000/1/1/0 C xy 1␣
↪       1 -140/20/-50/60
1008 USA_BOX     -129/25/-63/49                           -        2000/1/1/0 C xy 1␣
↪       1 -129/25/-63/49

### END SECTION MASKS ###
```

The required attributes for mask fields are described below:

Options *ScalID* and *Oper* are described in *Scale factors*.

Options *Name*, *sourceFile*, *sourceVar*, *sourceTime*, *CRE*, *SrcDim*, and *SrcUnit*, are described in *Base emissions*.

The `Box` option is deprecated.

Instead of specifying the *sourceFile* and *sourceVar* fields, you can directly provide the lower left and upper right box coordinates: `Lon1/Lat1/Lon2/Lat2` . Longitudes must be in degrees east, latitudes in degrees north. Only grid boxes whose mid points are within the specified mask boundaries. You may also specify a single grid point (`Lon1/Lat1/Lon1/Lat1/`).

Caveat for simulations using cropped horizontal grids

Consider the following combination of global and regional emissions inventories:

In the *Base Emissions* section:

```
0 GLOBAL_INV_SPC1    ...  SPC1 -     1 5
0 INVENTORY_1_SPC1   ...  SPC1 1001  1 56
0 INVENTORY_2_SPC1   ...  SPC1 1002  1 55
```

In the *Masks* section:

```
1001 REGION_1_MASK ... 1 1 70/10/140/60
1002 REGION_2_MASK ... 1 1 46/-12/180/82
```

For clarity, we have omitted the various elements in these entries of `HEMCO_Config.rc` that are irrelevant to this issue.

With this setup, we should expect the following behavior:

1. Species `SPC1` should be emitted globally from inventory `GLOBAL_INV` (hierarchy = 5).
2. Regional emissions of `SPC1` from `INVENTORY_1` (hierarchy = 56) should overwrite global emissions in the region specified by `REGION_1_MASK`.
3. Likewise, regional emissions of `SPC1` from `INVENTORY_2` (hierarchy = 55) should overwrite global emissions in the region specified by `REGION_2_MASK`.
4. In the locations where `REGION_2_MASK` intersects `REGION_1_MASK`, emissions from `INVENTORY_1` will be applied. This is because `INVENTORY_1` has a higher hierarchy (56) than `INVENTORY_2` (55).

When running simulations that use cropped grids, one or both of the boundaries specified for the masks (`70/10/140/60` and `46/-12/180/82`) in `HEMCO_Config.rc` can potentially extend beyond the bounds of the simulation domain. If this should happen, HEMCO would treat the regional inventories as if they were global, the emissions for the highest hierarchy (i.e., `INVENTORY_1`) would be applied globally. Inventories with lower hierarchies would be ignored.

Tip: Check the HEMCO log output for messages to make sure that none of your desired emissions have been skipped.

The solution is to make the boundaries of each defined mask region at least a little bit smaller than the boundaries of the nested domain. This involves inspecting the mask itself to make sure that no relevant gridboxes will be excluded.

For example, assuming the simulation domain extends from 70E to 140E in longitude, using this mask definition:

```
1001 REGION_1_MASK ... 1 1 70/10/136/60
```

would prevent `INVENTORY_1` from being mistakely treated as a global inventory. We hope to add improved error checking for this condition into a future HEMCO version.

Data collections

The fields listed in *the HEMCO configuration file* data collections. Collections can be enabled/disabled in section extension switches. Only fields that are part of an enabled collection will be used by HEMCO.

The beginning and end of a collection is indicated by an opening and closing bracket, respectively: :literal:(((CollectionName` and `)))CollectionName`. These brackets must be on individual lines immediately preceeding / following the first/last entry of a collection. The same collection bracket can be used as many times as needed.

The collections are enabled/disabled in the Extension Switches section (see *Extension Switches*). Each collection name must be provided as an extension setting and can then be readily enabled/disabled:

```
###############################################################################
#### BEGIN SECTION EXTENSION SWITCHES
###############################################################################
# ExtNr ExtName           on/off  Species
0       Base              : on    *
    --> MACCITY           :       true
    --> EMEP              :       true
    --> AEIC              :       true

### END SECTION EXTENSION SWITCHES

###############################################################################
### BEGIN SECTION BASE EMISSIONS
###############################################################################
ExtNr Name srcFile srcVar srcTime CRE Dim Unit Species ScalIDs Cat Hier

(((MACCITY
0 MACCITY_CO MACCity.nc  CO 1980-2014/1-12/1/0 C xy  kg/m2/s CO 500      1 1
)))MACCITY

(((EMEP
0 EMEP_CO    EMEP.nc     CO 2000-2014/1-12/1/0 C xy  kg/m2/s CO 500/1001 1 2
)))EMEP

(((AEIC
0 AEIC_CO    AEIC.nc     CO 2005/1-12/1/0      C xyz kg/m2/s CO -        2 1
)))AEIC

### END SECTION BASE EMISSIONS ###

###############################################################################
#### BEGIN SECTION SCALE FACTORS
###############################################################################
# ScalID Name srcFile srcVar srcTime CRE Dim Unit Oper

500 HOURLY_SCALFACT $ROOT/hourly.nc factor  2000/1/1/0-23 C xy 1 1
600 SO2toSO4        0.031           -       -             - -  1 1

### END SECTION SCALE FACTORS ###

###############################################################################
#### BEGIN SECTION MASKS
###############################################################################
#ScalID Name srcFile srcVar srcTime CRE Dim Unit Oper Box
```

(continues on next page)

(continued from previous page)

```
1001 MASK_EUROPE $ROOT/mask_europe.nc MASK 2000/1/1/0 C xy 1 1 -30/30/45/70

### END SECTION MASKS ###
```

Extension names

The collection brackets also work with *extension names*, e.g. data can be included/excluded based on extensions. This is particularly useful to include an emission inventory for standard emission calculation if (and only if) an extension is not being used (see example below).

Undefined collections

If, for a given collection, no corresponding entry is found in the extensions section, it will be ignored. Collections are also ignored if the collection is defined in an extension that is disabled. It is recommended to list all collections under the base extension.

Exclude collections

To use the opposite of a collection switch, .not. can be added in front of an existing collection name. For instance, to read file NOT_EMEP.nc only if EMEP is not being used:

```
(((.not.EMEP
0 NOT_EMEP_CO    $ROOT/NOT_EMEP.nc     CO 2000/1-12/1/0 C xy kg/m2/s CO 500/1001 1 2
))).not.EMEP
```

Combine collections

Multiple collections can be combined so that they are evaluated together. This is achieved by linking collection names with .or.. For example, to use BOND biomass burning emissions only if both GFED and FINN are not being used:

```
(((.not.GFED.or.FINN
0 BOND_BM_BCPI   $ROOT/BCOC_BOND/v2014-07/Bond_biomass.nc BC   2000/1-12/1/0 C xy kg/
↪m2/s BCPI 70 2 1
0 BOND_BM_BCPO   -                                         -    -             - - -  ␣
↪     BCPO 71 2 1
0 BOND_BM_OCPI   $ROOT/BCOC_BOND/v2014-07/Bond_biomass.nc OC   2000/1-12/1/0 C xy kg/
↪m2/s OCPI 72 2 1
0 BOND_BM_OCPO   -                                         -    -             - - -  ␣
↪     OCPO 73 2 1
0 BOND_BM_POA1   -                                         -    -             - - -  ␣
↪     POA1 74 2 1
))).not.GFED.or.FINN
```

2.1.3 HEMCO extensions

Overview

Emission inventories sometimes include dynamic source types and nonlinear scale factors that have functional dependencies on local environmental variables such as wind speed or temperature, which are best calculated online during execution of the model. HEMCO includes a suite of additional modules (extensions) that perform online emission calculations for a variety of sources (see list below). Extensions are enabled in section *Extension Switches* of *the HEMCO configuration file*.

List of extensions

The full list of available extensions is given below. Extensions can be selected individually in the *Extension Switches* section of the *The HEMCO configuration file*, as can the species to be considered.

DustAlk

- **Species**: DSTAL1, DSTAL2, DSTAL3, DSTAL4
- **Reference**: Fairlie et al (check)

DustDead

Emissions of mineral dust from the DEAD dust mobilization model.

- **Species**: DST1, DST2, DST3, DST4
- **Reference**: Zender *et al.* [[Zender et al., 2003]]

DustGinoux

Emissions of mineral dust from the P. Ginoux dust mobilization model.

- **Species**: DST1, DST2, DST3, DST4
- **Reference**: Ginoux *et al.* [[Ginoux et al., 2001]]

FINN

Biomass burning emissions from the FINN model.

- **Species**: NO, CO, ALK4, ACET, MEK, ALD2, PRPE, C2H2, C2H4, C3H8, CH2O, C2H6, SO2, NH3, BCPI, BCPO, OCPI, OCPO, GLYC, HAC, SOAP
- **Reference**: Wiedinmyer *et al.* [[Wiedinmyer et al., 2014]]

GC_Rn-Pb-Be

Emissions of radionuclide species as used in the GEOS-Chem model.

- **Species**: Rn222, Be7, Be7Strat, Be10, Be10Strat

ZHANG_Rn222

If *ZHANG_Rn222* is `on`, then Rn222 emissions will be computed according to Zhang *et al.* [[Zhang et al., 2021]].

If *ZHANG_Rn222* is `off`, then Rn222 emissions will be computed according to Jacob *et al.* [[Jacob et al., 1997]].

GFED

Biomass burning emissions from the GFED model.

- **Version**: GFED3 and GFED4 are available.
- **Species**: NO, CO, ALK4, ACET, MEK, ALD2, PRPE, C2H2, C2H4, C3H8, CH2O C2H6, SO2, NH3, BCPO, BCPI, OCPO, OCPI, POG1, POG2, MTPA, BENZ, TOLU, XYLE NAP, EOH, MOH, SOAP,

- **Reference**: van der Werf *et al.* [[van der Werf et al., 2010]]

Inorg_Iodine

- **Species**: HOI, I2
- **Reference**: TBD

LightNOx
Emissions of NOx from lightning.

- **Species**: NO
- **Species**: [[Murray et al., 2012]]

MEGAN
Biogenic VOC emissions.

- **Version**: 2.1
- **Species:** ISOP, ACET, PRPE, C2H4, ALD2, CO, OCPI, MONX, MTPA, MTPO, LIMO, SESQ
- **Reference:** Guenther *et al.* [[Guenther et al., 2012]]

PARANOx
Plume model for ship emissions.

- **Species**: NO, NO2, O3, HNO3
- **Reference**: Vinken *et al.* [[Vinken et al., 2011]]

SeaFlux
Air-sea exchange.

- Species: DMS, ACET, ALD2, MENO3, ETNO3, MOH
- References: Johnson [[Johnson 2010]], Nightingale *et al.* [[Nightingale et al., 2000]]

SeaSalt
Sea salt aerosol emission.

- **Species**: SALA, SALC, SALACL, SALCCL, SALAAL, SALCAL, BrSALA, BrSALC, MOPO, MOPI
- **References**: Jaeglé *et al.* [[Jaegle et al., 2011]], Gong [[Gong 2003]]

SoilNOx
Emissons of NOx from soils and fertilizers.

- **Species**: NO
- **Reference**: Hudman *et al.* [[Hudman et al., 2012]]

Volcano
Emissions of volcanic SO2 from AEROCOM.

- **Species**: SO2
- **Reference**:

TOMAS_Jeagle
Size-resolved sea salt emissions for TOMAS aerosol microphysics simulations.

- **Species**: SS1, SS2, SS3, SS4, SS5, SS6, SS7, SS8, SS9, SS10, SS11, SS12, SS13, SS14, SS15, SS16, SS17, SS18, SS19, SS20, SS21, SS22, SS23, SS24, SS25, SS26, SS27, SS28, SS29, SS30, SS31, SS32, SS33, SS34, SS35, SS36, SS37, SS38, SS39, SS40
- **Reference**: Jaeglé *et al.* [[Jaegle et al., 2011]]

TOMAS_DustDead
Size-resolved dust emissions for TOMAS aerosol microphysics simulations.

- **Species**: DUST1, DUST2, DUST3, DUST4, DUST5, DUST6, DUST7, DUST8, DUST9, DUST10, DUST11, DUST12, DUST13, DUST14, DUST15, DUST16, DUST17, DUST18, DUST19, DUST20, DUST21, DUST22, DUST23, DUST24, DUST25, DUST26, DUST27, DUST28, DUST29, DUST30, DUST31, DUST32, DUST33, DUST34, DUST35, DUST36, DUST37, DUST38, DUST39, DUST40
- **Reference**: Zender *et al.* [[Zender et al., 2003]]

Gridded data

HEMCO can host all environmentally independent data sets (e.g. source functions) used by the extensions. The environmental variables are either provided by the atmospheric model or directly read from file through the HEMCO configuration file. Entries in *the HEMCO configuration file* file are given priority over fields passed down from the atmospheric model, i.e. if the HEMCO configuration file contains an entry for a given environmental variable, this field will be used instead of the field provided by the atmospheric model. The field name provided in the HEMCO configuration file must exactly match the name of the HEMCO environmental parameter.

To use the NCEP reanalysis monthly surface wind fields (http:, , www.esrl.noaa.gov, psd, data, gridded, data.ncep.reanalysis.derived.surface.html) in all HEMCO extensions, add the following two lines to the *Base Emissions* section of *the HEMCO configuration file*:

```
* U10M /path/to/uwnd.mon.mean.nc uwnd 1948-2014/1-12/1/0 C xy m/s * - 1 1
* V10M /path/to/vwnd.mon.mean.nc vwnd 1948-2014/1-12/1/0 C xy m/s * - 1 1
```

This will use these wind fields for all emission calculations, even if the atmospheric model uses a different set of wind fields.

It is legal to assign scale factors (and masks) to the environmental variables read through *the HEMCO configuration file*. This is particularly attractive for sensitivity studies. For example, a scale factor of 1.1 can be assigned to the NCEP surface wind fields to study the sensitivity of emissions on a 10% increase in wind speed:

In the *Base Emissions* section:

```
* U10M /path/to/uwnd.mon.mean.nc uwnd 1948-2014/1-12/1/0 C xy m/s * 123 1 1
* V10M /path/to/vwnd.mon.mean.nc vwnd 1948-2014/1-12/1/0 C xy m/s * 123 1 1
```

In the *Scale Factors* section:

```
123 SURFWIND_SCALE 1.1 - - - xy 1 1
```

As for any other entry in the HEMCO configuration file, spatially uniform values can be set directly in the HEMCO configuration file. For example, a spatially uniform, but monthly varying surface albedo can be specified by adding the following entry to the *Base Emissions* section of *the HEMCO configuration file*:

```
* ALBD 0.7/0.65/0.6/0.5/0.5/0.4/0.45/0.5/0.55/0.6/0.6/0.7 - 2000/1-12/1/0 C xy 1 * -␣
↪1 1
```

Environmental fields used by HEMCO

The following fields can be passed from the atmospheric model to HEMCO for use by the various extensions:

AIR
Air mass.

- **Dim**: xyz
- **Units**: kg
- **Used by**: *GC_Rn-Pb-Be*, *PARANOx*

AIRVOL
Air volume (i.e. volume of grid box).

- **Dim**: xyz
- **Units**: kg
- **Used by**: *PARANOx*

ALBD
Surface albedo.

- **Dim**: xy
- **Units**: unitless
- **Used by**: *SoilNOx*, *SeaFlux*

CLDFRC
Cloud fraction

- **Dim**: xy
- **Units**: unitless
- **Used by**: *MEGAN*

CNV_MFC
Convective mass flux.

- **Dim**: xyz
- **Units**: kg/m2/s
- **Used by**: *LightNOx*

FRAC_OF_PBL
Fraction of grid box within the planetary boundary layer (PBL).

- **Dim**: xyz
- **Units**: unitless
- **Used by**: *PARANOx*, *SeaFlux*

FRCLND
Land fraction

- **Dim**: xy
- **Units**: unitless
- **Used by**: *GC_Rn-Pb-Be*, *SeaFlux*

GWETROOT
Root soil moisture.

- **Dim**: xy
- **Units**: unitless
- **Used by**: *MEGAN*

GWETTOP
Top soil moisture.

- **Dim**: xy
- **Units**: unitless
- **Used by**: *MEGAN*

HNO3
HNO3 mass.

- **Dim**: xyz
- **Units**: kg
- **Used by**: *PARANOx*

JO1D
Photolysis J-value for O1D.

- **Dim**: xy
- **Units**: 1/s
- **Used by**: *PARANOx*

JNO2
Photolysis J-value for NO2.

- **Dim**: xy
- **Units**: 1/s
- **Used by**: *PARANOx*

LAI
Leaf area index.

- **Dim**: xy
- **Units**: cm2 leaf/cm2 grid box
- **Used by**: *MEGAN*

NO
NO mass.

- **Dim**: xyz
- **Units**: kg
- **Used by**: *PARANOx*

NO2
NO2 mass.

- **Dim**: xyz
- **Units**: kg

- **Used by**: *PARANOx*

O3
O3 mass.

- **Dim**: xyz
- **Units**: kg
- **Used by**: *PARANOx*

PARDF
Diffuse photosynthetic active radiation

- **Dim**: xy
- **Units**: W/m2
- **Used by**: *MEGAN*

PARDR
Direct photosynthetic active radiation

- **Dim**: xy
- **Units**: W/m2
- **Used by**: *MEGAN*

RADSWG
Short-wave incident surface radiation

- **Dim**: xy
- **Units**: W/m2
- **Used by**: *SoilNOx*

SNOWHGT
Snow height (mm of H2O equivalent).

- **Dim**: xy
- **Units**: kg H2O/m2
- **Used by**: *DustDead*, *TOMAS_DustDead*

SPHU
Specific humidity

- **Dim**: xyz
- **Units**: kg H2O/kg air
- **Used by**: *DustDead*, *PARANOx*, *TOMAS_DustDead*

SZAFACT
Cosine of the solar zenith angle.

- **Dim**: xy
- **Units**: unitless
- **Used by**: *MEGAN*

TK
Temperature.

- **Dim**: xyz

- **Units**: K
- **Used by**: *DustDead*, *LightNOx*, *TOMAS_DustDead*

TROPP
Tropopause pressure.

- **Dim**: xy
- **Units**: Pa
- **Used by**: *GC_Rn-Pb-Be*, *LightNOx*

TSKIN
Surface skin temperature

- **Dim**: xy
- **Units**: K
- **Used by**: *SeaFlux*, *SeaSalt*

U10M
E/W wind speed @ 10 meters above surface.

- **Dim**: xy
- **Units**: m/s
- **Used by**: *DustAlk*, *DustDead*, *DustGinoux*, *PARANOx*, *SeaFlux*, *SeaSalt*, *SoilNOx*, *TOMAS_DustDead*, *TOMAS_Jeagle*

USTAR
Friction velocity.

- **Dim**: xy
- **Units**: m/s
- **Used by**: *DustDead*, *TOMAS_DustDead*

V10M
N/S wind speed @ 10 meters above surface.

- **Dim**: xy
- **Units**: m/s
- **Used by**: *DustAlk*, *DustDead*, *DustGinoux*, *PARANOx*, *SeaFlux*, *SeaSalt*, *SoilNOx*, *TOMAS_DustDead*, *TOMAS_Jeagle*

WLI
Water-land-ice flags (`0` = water, `1` = land, `2` = ice).

- **Dim**: xy
- **Units**: unitless
- **Used by**: Almost every extension

Z0
Roughness height.

- **Dim**: xy
- **Units**: m
- **Used by**: *DustDead*, *TOMAS_DustDead*

Restart variables

Some extensions rely on restart variables, i.e. variables that are highly dependent on historical information such as previous-day leaf area index or soil NOx pulsing factor. During a simulation run, the extensions continuously archive all necessary information and update estart variables accordingly. The updated variables become automatically written into the HEMCO restart file (`HEMCO_restart.YYYYMMDDhhmmss.nc`) at the end of a simulation. The fields from this file can then be read through the HEMCO configuration file to resume the simulation at this date ("warm" restart). For example, the soil NOx restart variables can be made available to the soil NOx extension by adding the following lines to the *Base Emissions section* of *the HEMCO configuration file*.

```
104 PFACTOR         ./HEMCO_restart.$YYYY$MM$DD$HH00.nc  PFACTOR       $YYYY/$MM/$DD/
↪$HH E xy  unitless NO - 1 1
104 DRYPERIOD       ./HEMCO_restart.$YYYY$MM$DD$HH00.nc  DRYPERIOD     $YYYY/$MM/$DD/
↪$HH E xy  unitless NO - 1 1
104 GWET_PREV       ./HEMCO_restart.$YYYY$MM$DD$HH00.nc  GWET_PREV     $YYYY/$MM/$DD/
↪$HH E xy  unitless NO - 1 1
104 DEP_RESERVOIR   ./HEMCO_restart.$YYYY$MM$DD$HH00.nc  DEP_RESERVOIR $YYYY/$MM/$DD/
↪$HH E xy  unitless NO - 1 1
```

Many restart variables are very time and date-dependent. It is therefore recommended to set the time slice selection flag to E to ensure that only data is read that exactly matches the simulation start date (also see *Base emissions*. HEMCO will perform a "cold start" if no restart field can be found for a given simulation start date, e.g. default values will be used for those restart variables.

Built-in tools for scaling/masking

HEMCO has built-in tools to facilitate the application of both uniform and spatiotemporal *scale factors* to emissions calculated by the extensions. At this point, not all extensions take advantage of these tools yet. A list of extensions that support the built-in scaling tools are given below.

For extensions that support the built-in scaling tools, you can specify the uniform and/or spatiotemporal scale factors to be applied to the extension species of interest in section *Extension switches the HEMCO configuration file*.

For example, to uniformly scale GFED CO by a factor of 1.05 and GFED NO emissions by a factor of 1.2, add the following two lines to the HEMCO configuration file (highlighted in GREEN):

```
111    GFED               : on    CO/NO/ACET/ALK4
   --> GFED3              :       false
   --> GFED4              :       true
   --> GFED_daily         :       false
   --> GFED_3hourly       :       false
   --> Scaling_CO         :       1.05
   --> Scaling_NO         :       1.20
```

Similarly, a spatiotemporal field to be applied to the species of interest can be defined via setting `ScaleField`, e.g.

```
111     GFED               : on    CO/NO/ACET/ALK4
    --> GFED3              :       false
    --> GFED4              :       true
    --> GFED_daily         :       false
    --> GFED_3hourly       :       false
    --> Scaling_CO         :       1.05
    --> Scaling_NO         :       1.20
    --> ScaleField_NO      :       GFED_SCALEFIELD_NO
```

The corresponding scale field needs be defined in section *Base emissions* . A simple example would be a monthly varying scale factor for GFED NO emissions:

```
111 GFED_SCALEFIELD_NO   0.9/1.1/1.3/1.4/1.6/1.7/1.7/1.8/1.5/1.3/0.9/0.8 - 2000/1-12/
↪1/0 C xy unitless * - 1 1
```

It is legal to apply scale factors and/or masks to the extension scale fields (in the same way as the 'regular' base emission fields). A more sophisticated example on how to scale soil NOx emissions is given in HEMCO examples.

Extensions supporting built-in scaling/masking

The following extensions currently support the built-in scaling/masking tools: *SoilNOx*, *GFED*, *FINN*.

Adding new HEMCO extensions

All HEMCO extensions are called through the extension interface routines in `HEMCO/Extensions/hcox_driver_mod.F90`: `HCOX_INIT`, `HCOX_RUN`, `HCOX_FINAL`. For every new extension, a corresponding subroutine call needs to be added to those three routines. You will quickly see that these calls only take a few arguments, most importantly the HEMCO state object `HcoState` and the extensions state object `ExtState`.

`ExtState` is defined in `HEMCO/src/Extensions/hcox_state_mod.F90`. It contains logical switches for each extension as well as pointers to any external data (such as met fields). For a new extension, you'll have to add a new logical switch to the Ext_State object. If you need external data that is not yet included in ExtState, you will also have to add those (including the pointer associations in subroutine `SET_EXTOPT_FIELDS` in `GeosCore/hco_interface_gc_mod.F90`.

The initialization call (`HCOX_XXX_INIT`) should be used to initialize all extension variables and to read all settings from the HEMCO configuration file. There are a number of helper routines in `HEMCO/src/Extensions/hco_extlist_mod.F90` to do this:

- `GetExtNr( ExtName )` returns the extension number for the given extension name. Will return –1 if extension is turned off!
- `GetExtOpt( ExtNr, Attribute, Value, RC )` can be used to read any additional extension options (logical switches, path and names of csv-tables, etc.). Note that value can be of various types (`logical`, `character`, `double`,...).
- `GetExtHcoID( HcoState, ExtNr, HcoIDs, SpcNames, nSpc, RC )` matches the extension species names (as defined in the configuration file) to the species defined in HEMCO state (i.e. to all available HEMCO species). A value of –1 is returned if the given species is not defined in HEMCO.

All `ExtState` variables used by this extension should be updated. This includes the logical switch and all external data needed by the extension. For example, if the extension needs temperature data, this pointer should be activated by setting `ExtState%TK%DoUse = .TRUE.`

The run call (`HCOX_XXX_RUN`) calculates the 2D fluxes and passes them to HcoState via subroutine `HCO_EmisAdd( HcoState, Flux, HcoID, RC)`. External data is assessed through `ExtState` (e.g. `ExtState%TX%Arr%Val(I,J,L)`), and any data automatically read from netCDF files (through the HEMCO interface) can be obtained through `EmisList_GetDataArr( am_I_Root, FieldName, Pointer, RC )` The body of the run routine is typically just the code of the original module.

It's probably easiest to start from an existing extension (or the `Custom` extension template) and to add any modifications as is needed.

2.1.4 Units in HEMCO

Overview

Attention: We recommend that you provide explicit scale factors for unit conversions in *the HEMCO configuration file*. This will avoid some *known issues* with unit conversions that were recently discovered.

HEMCO classifies all data fields as fluxes, concentrations, or unitless data. Data are internally stored in HEMCO standard units of `[kg emitted species/m2/s]` for fluxes, and `[kg emitted species/m3]` for concentrations. No unit conversion is performed on unitless data.

The classification of a data field depends on the units attribute in the netCDF file, the *SrcUnit* attribute in *the HEMCO configuration file*, and the unit tolerance setting in the HEMCO configuration file (see below). In general, the original units of the input data is determined based on the units attribute on the netCDF file, and data is converted to HEMCO units accordingly. The mass conversion factor is determined based on the species assigned to the given field throuh attribute *Species* in the HEMCO configuration file. It depends on the species molecular weight (MW), the MW of the emitted species, and the molecular ratio (molecules of emitted species per molecules of species). If the input data is found to be in non-standard units (e.g. `L` instead of `m3`, `g` instead of `kg`, etc.), HEMCO will attempt to convert to standard units. This feature is not fully tested yet, and it is recommended to provide input data in standard units wherever possible.

SrcUnit attribute

The *SrcUnit* attribute in *the HEMCO configuration file* gives the user some control on unit conversion.

If *SrcUnit* is set to `1`, data are treated as unitless irrespective of the units attribute on the file. This option works on all fields only if unit tolerance is relaxed to `2` (for unit tolerance of `1`, the input data must be in one of the units recognized by HEMCO as `unitless`).

If *SrcUnit* is set to `count`, the input data is assumed to represent index-based scalar fields (e.g. land types). No unit conversion is performed on these data and regridding will preserve the absolute values.

Special attention needs to be paid to species that are emitted in quantities other than mass of species, e.g. `kg C`. For these species, the species MW differs from the emitted species MW, and the molecular ratio determines how many molecules are being emitted per molecule species. By default, HEMCO will attempt to convert all input data to kg emitted species. If a species is emitted as `kgC/m2/s` and the input data is in kg/m2/s, the mass will be adjusted based on the emitted MW, species MW, and the ratio emitted MW / species MW. Only input data that is already in `kgC/m2/s` will not be converted. This behavior can be avoided by explicitly set the *SrcUnit* to the same unit as on the input file. In this case, HEMCO will not convert between species MW and emitted MW. This is useful for cases where the input data does not contain data of the actual species, e.g. if VOC emissions are calculated by scaling CO emissions (see examples below).

Unit tolerance setting

The unit tolerance setting (see the *Settings* section of *the HEMCO configuration file* indicates the tolerance of HEMCO if discrepancies are found between the units found in the input file and attribute *SrcUnit* of the configuration file.

- If the unit tolerance is set to `0`, HEMCO stops with an error if the *SrcUnit* attribute does not exactly match with the units attribute found in the input data.
- Unit tolerance of `1` enables the default behavior.
- Unit tolerance of `2` will take the *SrcUnit* attribute as the data input unit, regardless netCDF units attribute.

Unitless data

The following units are currently recognized as 'unitless' by HEMCO

- `1`
- `count`
- `unitless`
- `fraction`
- `factor`
- `scale`
- `hours`
- `v/v`
- `v/v/s`
- `s-1`
- `m2/m2`
- `kg/kg`
- `K`
- `W/m2`
- `pptv`
- `ppt`
- `ppbv`
- `ppb`
- `ppmv`
- `ppm`
- `ms-1`
- `m`
- `cm2cm-2`
- `dobsons`
- `dobsons/day`
- `hPa`
- `Pa`

Examples with units

Attention: We recommend that you provide explicit scale factors for unit conversions in *the HEMCO configuration file*. This will avoid some *known issues* with unit conversions that were recently discovered.

File `file1.nc` contains field `DATA` in units of `kg/m2/s`. It shall be applied to species acetone (`ACET`), which is emitted as `kg C`. The species molecular weight of ACET is `58`, the emitted molecular weight is `12` (i.e. that of carbon), and the molecular ratio is `3` (3 molecules of carbon per molecule of acetone).

The following entry in the HEMCO configuration file will interpret the input data as `kg acetone/m2/s`, and convert it to `kg C/m2/s` using a scale factor of `0.62` (= 12/58*3):

```
#--> data is converted from kg acetone/m2/s to kgC/m2/s
0 ACET  /path/to/file1.nc  DATA 2000/1/1/0 C xy kgC/m2/s ACET - 1 1
```

The following entry will avoid the unit conversion from kg to kgC:

```
#--> data is kept in kg species/m2/s
0 ACET  /path/to/file1.nc  DATA 2000/1/1/0 C xy kg/m2/s ACET - 1 1
```

Note that the opposite does not work: If `file2.nc` contains data in units of `kgC/m2/s`, it is not possible to convert to kg species/m2/s and the following two entries have the same effect:

```
 #--> data is converted from kgC/m2/s to kg emitted species/m2/s,
 #    which is also kgC/m2/s``
0 ACET  /path/to/file2.nc  DATA 2000/1/1/0 C xy kg/m2/s  ACET - 1 1

#--> data is kept in kgC/m2/s
0 ACET  /path/to/file2.nc  DATA 2000/1/1/0 C xy kgC/m2/s ACET - 1 1
```

However, if one wants to use file2 for a species not emitted as kg carbon, say CO, the source unit attribute matters!

```
 #--> data is converted from kgC/m2/s to kg CO/m2/s
0 ACETasCO  /path/to/file2.nc  DATA 2000/1/1/0 C xy kg/m2/s  CO - 1 1

#--> data is kept in kgC/m2/s
0 ACETasCO  /path/to/file2.nc  DATA 2000/1/1/0 C xy kgC/m2/s CO - 1 1
```

Tips for testing

The unit factor applied by HEMCO is written into the HEMCO log file if *Verbose* is set to 2 or higher.

2.1.5 HEMCO diagnostics

Overview

HEMCO diagnostics are organized in **collections**, with each collection consisting of a dynamic number of diagnostic fields (aka **diagnostic containers**). Each collection has a fixed output frequency (*DiagnFreq*) assigned to it. All fields within a collection are written out at the same interval: *Hourly*, *Daily*, etc.

The contents of a collection (i.e. the diagnostics containers) are defined at the beginning of a simulation and become continuously updated and written out during the simulation. A number of attributes attached to each diagnostic define the properties of a given field and how to perform field operations such as time averaging, unit conversion, etc. These

attributes include the **field name** (this will also be the netCDF variable name), the designated field **output units**, the **averaging method**, and an explicit **unit conversion factor**. The latter three determine how data is internally stored and returned for output. The data returned for output is not necessarily in the same units as it is internally stored.

By default, HEMCO assumes the passed fields are in kg/m2/s, stores them in kg/m2, and returns the average flux over the designated output interval in the units assigned to this field (default is kg/m2/s). This behavior can be avoided by explicitly setting the averaging method.

TODO: Find out where these get defined

Currently supported averaging methods are:

`instantaneous`
: Instantaneous values (recommended method).

`mean`
: Arithmetic mean over the diagnostic interval.

`sum`
: Total sum over the diagnostic interval.

`cumulsum`
: Cumulative sum since simulation start.

Explicitly setting the averaging method will disable automatic unit conversion and the fields passed to this diagnostic will be stored as is. The optional unit conversion factor can be set to perform a unit conversion before returning the field for output.

Note: It is highly recommended to explicitly set the averaging method for all fields that are not in kg/m2/s.

Built-in diagnostic collections

HEMCO has three built-in diagnostic collections (*Default*, *Restart*, and *Manual*) that are automatically created on every HEMCO run. These collections are used by HEMCO for internal data exchange and to write out restart variables. These collections are 'open', i.e. the user can add additional diagnostic fields to them if needed. The user can also define new collections (see below).

The Default collection

The **Default** collection contains emission diagnostics intended to be written to disk, e.g. for analysis purposes. All fields of the default collection are written out at the frequency provided in setting `DiagnFreq` in the settings section of the HEMCO configuration file. The name of the corresponding diagnostics files can be specified via the `DiagnPrefix` setting. The simulation date at the time of output will be appended to the diagnostics prefix, e.g. the diagnostics for Aug 1, 2008 will be written as `HEMCO_Diagnostics.200808010000.nc`. The datetime can denote the beginning, middle, or end (default) of the time interval, as specified by setting `DiagnTimeStamp` (see below).

Several *options for the default diagnostic collection* can be specified at the top of the *HEMCO configuration file* file. Commonly-used options are `DiagnFile`, `DiagnFreq`, and `DiagnPrefix`.

Configuration file for the Default collection

Adding the following entries to the diagnostic configuration file (i.e. the same file specified by *DiagnFreq*, commonly called HEMCO_Diagn.rc) will make HEMCO write out total NO and CO emissions, as well as GFED biomass burning CO emissions (e.g. only emissions from extension 111):

```
# Name          Spec ExtNr  Cat Hier Dim Unit     LongName
EmisNO_Total    NO   -1     -1  -1   2   kg/m2/s  NO_emission_flux_from_all_
↪sectors
EmisCO_Total    CO   -1     -1  -1   2   kg/m2/s  CO_emission_flux_from_all_
↪sectors
EmisCO_GFED     CO   111    -1  -1   2   kg/m2/s  CO_emission_flux_from_
↪biomass_burning
```

If you want to just diagnose regional emissions, then you need to set the diagnostics extension number, category and hierarchy accordingly. For example, if you want EPA16 emissions for CO over the USA, then add this line:

```
#Name           Spec ExtNr  Cat Hier Dim Unit     Longname
EmisCO_EPA16    CO   0      1   50   2   kg/m2/s  CO_emission_flux_from_EPA16_
↪inventory
```

It is important that you define valid values for all attributes up to the hierarchy. As soon as you set an attribute to default (-1), HEMCO will take the sum up to this attribute. For example, the following diagnostics would simply return total base emissions:

```
#Name           Spec ExtNr  Cat Hier Dim Unit     Longname
EmisCO_EPA16    CO   0      -1  50   2   kg/m2/s  CO_emission_flux_from_
↪EPA16_inventory
```

Restart

The output frequency of the **Restart** collection is End, meaning that its content is only written out at the end of a simulation. The HEMCO Restart collection primarily consists of a suite of fields needed by some of the HEMCO extensions for a "warm" HEMCO restart (e.g. the 10-day running mean temperature, etc.). These fields are automatically added to the HEMCO restart collection and filled within the respective extensions. Once archived, fields can be made available to an extension via the HEMCO configuration file.

Manual

Fields in the **Manual** collection do not become written out to disk. Rather, they provide a tool to exchange data files within and outside of HEMCO, e.g. to pass sector-specific emission fluxes from HEMCO to the atmospheric model.

Some HEMCO extensions automatically create and fill a number of manual diagnostics. For example, the PARANOX extension (used in GEOS-Chem) stores the O3 and HNO3 loss fluxes in the manual diagnostics PARANOX_O3_DEPOSITION_FLUX and PARANOX_HNO3_DEPOSITION_FLUX, respectively.

Importing diagnostic content into an external model

The content of the *Default collection* can be specified through the HEMCO diagnostics definitions file (specified by the `DiagnFile` option).

The content of the *Manual* and *Restart* collections currently need to be defined within the model code (e.g. it is hard-coded). This should be done in high-level routines (at the HEMCO-to-model interface level).

Module `hco_diagn_mod.F90` (found in `HEMCO/src/Core/`) provides a suite of routines to define, fill, obtain, etc. diagnostic fields. Similarly, `hco_restart_mod.F90` (also found in `HEMCO/src/Core/`) provides routines for managing HEMCO restart variables.

2.1.6 More configuration examples

Scale factor examples

Scale (or zero) emissions with a shapefile country mask

HEMCO has the ability to define country-specific scale factors. To utilize this feature, you must first specify a mask file in the **NON-EMISSIONS DATA** section of *the HEMCO configuration file*, such as:

```
#==============================================================================
# --- Country mask file ---
#==============================================================================
* COUNTRY_MASK /path/to/file/countrymask_0.1x0.1.nc CountryID 2000/1/1/0 C xy count *␣
↪- 1 1
```

The mask file specified above was created from a shapefile obtained from the GADM database. The country mask netCDF file (countrymask_0.1x0.1.nc) identifies countries by their ISO 3166-1 numeric code. Countries and their ISO3166-1-numeric codes are listed in the country_codes.csv file.

The country-specific scale factors can be specified in a separate ASCII file ending with with the suffix `.txt`. The container name of the mask file (e.g. `COUNTRY_MASK`) must be given in the first line of the file. The following lines define the country-specific scale factors. ID 0 is reserved for the default values that are applied to all countries with no specific values listed. An example `scalefactor.txt` file is provided below:

```
# Country mask field name
COUNTRY_MASK

# Country data
# Name    | ID   | Scale factor
DEFAULT     0      1.0
CHINA       156    0.95
INDIA       356    1.10
KOREA       410    0.0
```

The scale factor(s) listed are interpreted by HEMCO the same way as other scale factors. Multiple values separated by `/` are interpreted as temporally changing values:

- 7 values = Sun, Mon, . . . , Sat;
- 12 values = Jan, Feb, . . . , Dec;
- 24 values = 12am, 1am, . . . , 11pm (local time!).

The country-specific scale factors would then be defined in the *Scale Factors* section of *the HEMCO configuration file* as:

```
501 SCALE_COUNTRY /path/to/file/scalefactor.txt  - - - xy count 1
```

The scale factors can the be applied to the emission field(s) that you wish to scale. For example:

```
0 MIX_NO_IND MIX_Asia_NO.generic.025x025.nc NO_INDUSTRY 2008-2010/1-12/1/0 C xy kg/m2/
↪s NO  1/27/25/1006/ 501 1/2 45
```

These steps can also be used to scale emissions for different regions (e.g. provinces, states) by providing HEMCO with a mask file containing the regions to be scaled.

Scale (or zero) emissions with a rectangular mask

Important: If you are using HEMCO versions prior to 3.5.0, you may encounter a bug when trying to follow this example. See Github issue: https://github.com/geoschem/HEMCO/issues/153 for a workaround.

Another way to scale all emissions over a country (or set them to zero) is to apply a rectangular mask.

For example, to set all emissions over Australia and surrounding islands to zero, add this line to the *Masks* section of *the HEMCO configuration file*:

```
1010 AUS_MASK 105.0/-46.0/160.0/-10.0 - 2000/1/1/0 C xy 1 1 105/-46/160/-10
```

Here you directly provide the lower left and upper right corner of the mask region mask instead of a netCDF file: `lon1/lat1/lon2/lat2` You can then combine this mask with a scale factor of zero to eliminate any emissions over that area.

In *Base emissions*

```
0 HTAP_NO_IND /path/to/HTAP_NO_INDUSTRY.generic.01x01.nc emi_no 2008-2010/1-12/1/0 C␣
↪xy kg/m2/s NO 1/27/25/501 1/2 4
```

In *Scale Factors*:

```
501 SCALE_AUS 0.0 - - - xy unitless 1 1010
```

In *Masks*:

```
# Defines a rectangular region that should cover AUS + surrounding islands
1010 AUS_MASK 105.0/-46.0/160.0/-10.0 - 2000/1/1/0 C xy 1 1 105.0/-46.0/160.0/-10.0
```

Scale emissions by species

You may define uniform scale factors for single species that apply across all emission inventories, sectors and extensions. These scale factors can be set in the *Settings* section of *the HEMCO configuration file*, using the `EmissScale_<species-name>`, where `<species-name>` denotes the name of a HEMCO species such as `CO`, `CH4`, `NO`, etc.

For instance, to scale all NO emissions by 50% add the line `EmisScale_NO` to the *Settings* section of the *the HEMCO configuration file*:

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

ROOT:                         /path/to/HEMCO/data/directory
Logfile:                      HEMCO.log
... etc ...
EmisScale_NO                  1.5

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

Zero emissions of selected species

To zero emissions of a given species (e.g. NO) from any inventory listed under *Base Emissions*, do the following:

1. Create your own scale factor and assign value 0.0 to it. This must go into the *Scale Factors* section of *the HEMCO configuration file*:

   ```
   400 ZERO 0.0 - - - xy 1 1
   ```

2. Apply this scale factor to all of the emissions entries in the HEMCO configuration file that you would like to zero out. For example:

   ```
   0 MIX_NO_IND  /path/to/MIX_Asia_NO.generic.025x025.nc NO_INDUSTRY  2008-2010/1-12/
   ↪1/0 C xy kg/m2/s  NO  1/27/25/400/1006 1/2 45
   ```

This can be a useful way to set the emissions of some species to zero for sensitivity study purposes.

Note: All scale factors should be listed before masks.

Scale extension emissions globally by species

You may pass a global scale factor to the *HEMCO extensions*. For example, to double soil NO emissions everywhere, add the `Scaling_NO` to the section for the *SoilNOx* extension. This is located in the *Extension Switches* section of *the HEMCO configuration file*, as shown below:

```
104     SoilNOx           : on    NO
    --> Use fertilizer NOx:       true
    --> Scaling_NO        :       2.0
```

Scale summertime soil NOx emisions over the US

It is possible to pass uniform and/or spatiotemporal scale factors to some of the extensions, including *SoilNOx*.

For instance, suppose you want to halve summertime soil NOx emissions over the continental US. You can do this by defining a scale field (here, `SOILNOX_SCALE`) to the *SoilNOx* emission field in the *Extension Switches* section of *the HEMCO configuration file*:

```
104 SOILNOX_ARID    /path/to/soilNOx.climate.generic.05x05.nc  ARID     2000/1/1/0 C␣
↪xy unitless NO -   1 1
104 SOILNOX_NONARID /path/to/soilNOx.climate.generic.05x05.nc  NON_ARID 2000/1/1/0 C␣
↪xy unitless NO -   1 1
104 SOILNOX_SCALE   1.0                                        -        2000/1/1/0 C␣
↪xy unitless *  333 1 1
```

SOILNOX_SCALE is just a dummy scale factor with a global uniform value of 1.0. The actual temporal scaling over the US is done via scale factor 333 assigned to this field. This approach ensures that all *SoilNOx* emissions outside of the US remain intact.

The next step is to define scale factor 333 (named SOILNOX_SCALE) in the *Scale Factors* section of the *configuration file*:

```
# Scale factor to scale US soil NOx emissions by a factor of 0.5 in month June-August
333 SOILNOX_SCALE 1.0/1.0/1.0/1.0/1.0/0.5/0.5/0.5/1.0/1.0/1.0/1.0 - 2000/1-12/1/0 -␣
↪xy 1 1 5000
```

Scale factor SOILNOX_SCALE defines a monthly varying scale factor, with all scale factors being 1.0 except for months June-August, where the scale factor becomes 0.5. The last column of the SOILNOX_SCALE entry assigns mask number 5000 to this scale factor. This ensures that the scale factor will only be applied over the region spanned by mask 5000. This musk mast be defined in the *Masks* section of *the HEMCO configuration file*:

```
1005 USA_MASK      /path/to/usa.mask.nei2005.geos.1x1.nc  MASK 2000/1/1/0 C xy 1 1 -
↪165/10/-40/90
5000 SOILNOX_MASK   -106.3/37.0/-93.8/49.0                  -    -         - xy 1 1 -
↪106.3/37.0/-93.8/49.0
```

In this example, mask 5000 is defined as the region between 106.3 - 93.8 degrees west and 37.0 - 49.0 degrees north. If you want to apply the soil NOx scaling over the entire US, you can also just refer to the existing USA mask, e.g.:

```
# Scale factor to scale US soil NOx emissions by a factor of 0.5 in month June-August.
333 SOILNOX_SCALE 1.0/1.0/1.0/1.0/1.0/0.5/0.5/0.5/1.0/1.0/1.0/1.0 - 2000/1-12/1/0 -␣
↪xy 1 1 1005
```

Mask file examples

Exercise care in defining mask regions

In an effort to reduce I/O HEMCO ignores any emission entries that are deemed "irrelevant" because there is another (global) emission entry for the same species and emission category (*Cat*), but higher hierarchy (*Hier*).

For instance, suppose you have the following two fields defined under *Base Emissions*:

```
0 TEST_1 file.nc var 2000/1/1/0 C xy 1 1 CO - 1 1
0 TEST_2 file.nc var 2000/1/1/0 C xy 1 1 CO - 1 2
```

In this case, during initialization HEMCO determines that TEST_1 is obsolete because it will always be overwritten by TEST_2 because of its higher hierarchy. But if there is a mask assigned to an emission inventory, HEMCO uses the provided mask domain to determine whether this inventory has to be treated as "global" or not.

Going back to the example above, let's add a mask to TEST_2:

```
0 TEST_1 file.nc var 2000/1/1/0 C xy 1 1 CO -    1 1
0 TEST_2 file.nc var 2000/1/1/0 C xy 1 1 CO 1000 1 2
```

and let´s define the following *mask*:

```
1000 TEST_MASK mask.nc var 2000/1/1/0 C xy 1 1 -180/180/-90/90
```

HEMCO uses the mask range (`180/180/-90/90`) to define the extension of this mask. If that range covers the entire HEMCO grid domain, it considers every emission inventory linked with this mask as ¨global¨. In our example, `TEST_2` would still be considered global because the mask extends over the entire globe, and `TEST_1` is thus ignored by HEMCO.

However, changing the mask domain to something smaller will tell HEMCO that `TEST_2` is not global, and that it cannot drop `TEST_1` because of that:

```
1000 TEST_MASK mask.nc var 2000/1/1/0 C xy 1 1 -90/180/-45/45
```

Long story short: if you set the mask range to a domain that is somewhat smaller than your simulation window, things work just fine. But if you set the range to something bigger, HEMCO will start ignoring emission files.

Preserve fractional values when masking emissions

Question from a HEMCO user:

> I see that when the mask files are regridded they are remapped to 0 or 1 via regular rounding. Unfortunately, this method will not work well for my application, because the region I am trying to zero out is a small region inside the 4x5 grid cell and thus the current mask will not change the emissions on a $4^\circ \times 5^\circ$ scale.
>
> I was wondering whether it would be possible/straightforward to modify the mask regridding method such that $4^\circ \times 5^\circ$ emissions scale will scale with the fraction of the gird cell that is masked (e.g., if a quarter of the grid cells in one of the $4^\circ \times 5^\circ$ grid are masked, the emissions will scale down by 25%).

For this application, it may better to define your mask file in the *Scale Factors* section of *the HEMCO configuration file*.

By defining a mask in the *Masks* section, HEMCO identifies the data container type as MASK and treats the data as binary. Long story short:

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

If your mask file is currently defined here ...

### END SECTION MASKS ###
```

If you instead move that line to the SECTION SCALE FACTORS then HEMCO will treat the mask as type SCAL. I believe that would preserve the regridded value (in your example 0.25) and apply that to the emissions in a 4x5 grid box.

```
###############################################################################
### BEGIN SECTION SCALE FACTORS
###############################################################################

... put your mask file here instead ...

### END SECTION SCALE FACTORS ###
```

Create emissions for geographically tagged species

Important: Tagging emissions by geographic regions is currently supported only for *base emissions* but not for emissions computed by *HEMCO extensions*. We hope to add this capability into a future HEMCO version.

If you are using HEMCO interfaced to an external model, and need to create emissions for geographically tagged species, follow thse steps.

1. Define masks for your geographic regions in the *Masks* secton of *the HEMCO configuration file*:

```
#==============================================================================
# Country/region masks
#==============================================================================
1001 MASK_1  -30/30/45/70    - 2000/1/1/0 C xy 1 1 -30/30/45/70
1002 MASK_2  -118/17/-95/33  - 2000/1/1/0 C xy 1 1 -118/17/-95/33
1003 MASK_3  my_mask_file.nc - 2000/1/1/0 C xy 1 1 105/-46/160/-10

# ... etc ...
```

   If your mask regions are rectangular, you can specify the longitude and latitude at the box corners (such as was done for `MASK_1` and `MASK_2`). You may also read a mask definition from a netCDF file (as was done for `MASK_3`).

2. In the *Base Emissions* section of *the HEMCO configuration file*, add extra entries for tagged species underneath the entry for the global species, such as:

```
#==============================================================================
# --- EDGAR v4.2 emissions, various sectors ---
#==============================================================================
(((EDGAR

### Gas and oil ###
0 CH4_GAS__1B2a    v42_CH4.0.1x0.1.nc  ch4_1B2a  2004-2008/1/1/0 C xy kg/m2/s CH4␣
↪  -    1 1
0 CH4_GAS__1b2a_a  -                   -         -               - -  -       CH4_
↪a 1001 1 1
0 CH4_GAS__1b2a_b  -                   -         -               - -  -       CH4_
↪b 1002 1 1
0 CH4_GAS__1b2a_c  -                   -         -               - -  -       CH4_
↪c 1003 1 1
# ... etc ...

### Coal mines ###
0 CH4_COAL__1B1    v42_CH4.0.1x0.1.nc  ch4_1B1   2004-2008/1/1/0 C xy kg/m2/s CH4␣
↪  -    2 1
0 CH4_COAL__1B1_a  -                   -         -               - -  -       CH4_
↪a 1001 2 1
0 CH4_COAL__1B1_b  -                   -         -               - -  -       CH4_
↪b 1002 2 1
0 CH4_COAL__1B1_c  -                   -         -               - -  -       CH4_
↪c 1003 2 1
# ... etc ...``
```

This will put the total emissions into your CH4 tracer (tracer #1). It will then also apply the regional masks to the total emissions and then store them into tagged species (i.e. `CH4_a`, `CH4_b`, and `CH4_c`). These tagged species must also be defined in your external model with the same names.

HEMCO extensions examples

Fix MEGAN extension emissions to a specified year

Question submitted by a HEMCO user:

> Is it possible to fix *MEGAN* emissions to a given year? I know this works for many other *base emissions* inventories, but MEGAN emissions are dependent on environmental variables.

Your best option may be to run the HEMCO standalone and save out MEGAN emissions for the desired year. Then, in a subsequent run, you can read in the *HEMCO diagnostic output* files containing the archived *MEGAN* emissions.

1. Run the HEMCO standalone model. Make sure the following entries to your `HEMCO_Diagn.rc` file:

```
EmisISOP_Biogenic  ISOP   108    -1  -1   2   kg/m2/s  ISOP_emissions_from_
↪biogenic_sources
EmisISOP_Biogenic  ISOP   108    -1  -1   2   kg/m2/s  ISOP_emissions_from_
↪biogenic_sources
EmisALD2_Biogenic  ALD2   108    -1  -1   2   kg/m2/s  ALD2_emissions_from_
↪biogenic_sources
# ... etc for other MEGAN species ...
```

   In the above entries, `108` tells HEMCO to get the emissions from the *MEGAN* extension, which is listed in the *Extension Switches* section of the *configuration file* with *ExtNr* 108.

2. Add the following lines in the *Settings* section of *the HEMCO configuration file*:

```
DiagnFile:                  HEMCO_Diagn.rc
DiagnPrefix:                HEMCO_diagnostics
DiagnFreq:                  Monthly
```

   For more information, see the sections on *DiagnFile*, *DiagnPrefix*, *DiagnFreq*.

3. Turn off the MEGAN extension in the *Extension Switches* section of the configuration file.

```
108     MEGAN                   : off   ISOP/ACET/PRPE/...etc additional species...
```

4. Add entries for reading the fixed MEGAN emission that were archived in Step 1 under *Base Emissions*. For example:

```
0 MEGAN_ISOP /path/to/HEMCO_diagnostic.2016$MM010000.nc EmisISOP_Biogenic 2016/1-
↪12/1/1/0 C xy kg/m2/s ISOP - 4 1
```

   Note: HEMCO category `Cat = 4` is reserved for biogenic emissions.

5. Run HEMCO in either standalone mode, or coupled to an external model, dependingon your application.

Add 2D emissions into specific levels

HEMCO can emit emissions into a layer other than the surface layer. For example:

```
0 EMEP_CO EMEP.nc CO 2000-2014/1-12/1/0 C xyL5 kg/m2/s CO 1/1001 1 2
```

will release the `EMEP_CO` into level 5 instead of level 1. Theoretically, you could create a separate HEMCO entry for every emission level (under *Base Emissions*:

```
0 EMEP_CO_L1 EMEP.nc CO 2000-2014/1-12/1/0 C xyL1 kg/m2/s CO 1 150/1001 1 2
0 EMEP_CO_L2 EMEP.nc CO 2000-2014/1-12/1/0 C xyL2 kg/m2/s CO 1 151/1001 1 2
0 EMEP_CO_L3 EMEP.nc CO 2000-2014/1-12/1/0 C xyL3 kg/m2/s CO 1 152/1001 1 2
```

and assign *Scale Factors* (e.g. 150, 151, 152) to specify the fraction of EMEP emissions to be added into each level:

```
151 EMEP_LEV1_FRAC 0.5 - - - xy 1 1
152 EMEP_LEV2_FRAC 0.1 - - - xy 1 1
153 EMEP_LEV3_FRAC 0.1 - - - xy 1 1``
```

But this approach is somewhat cumbersome. Also, this won't give you the possibility to specifically emit a fraction above the PBL given that the PBL height is variable over time.

Use this notation (under *Base Emissions*) to tell HEMCO that you would like EMEP emissins to be added into levels 1 through 3:

```
0 EMEP_CO_L1 EMEP.nc CO 2000-2014/1-12/1/0 C xyL=1:3 kg/m2/s CO 1 1001 1 2
```

The emissions are then spread across the lowest 3 model levels based upon the model level thicknesses.

Instead of specifying the model levels, you may also specify the altitude in meters or use `PBL` for the planetary boundary layer:

```
# Emit from surface up to 2500 meters
0 EMEP_CO_L1 EMEP.nc CO 2000-2014/1-12/1/0 C xyL=1:2500m kg/m2/s C 1001 1 2

# Emit between 1000 and 5000 meters altitude
0 EMEP_CO_L1 EMEP.nc CO 2000-2014/1-12/1/0 C xyL=1000m:5000m kg/m2/s CO 1 1001 1 2

# Emit between 5000 meters altitude and model level 17
0 EMEP_CO_L1 EMEP.nc CO 2000-2014/1-12/1/0 C xyL=500m:17 kg/m2/s CO 1 1001 1 2

# Emit from the surface to the PBL top
0 EMEP_CO_L1 EMEP.nc CO 2000-2014/1-12/1/0 C xyL=1:PBL kg/m2/s CO 1 1001 1 2
```

HEMCO can also read the emission levvel from an external source (e.g. netCDF file) that is listed as a scale factor. This field can then be referred to using its scale factor ID. As an example, let's assume daily varying emission heights for 2009-2010 are archived in `emis_heights.nc` as variable `emish` in units of `m`. available for years 2009 to 2010). You can then define a *Scale Factor* such as:

```
300 EMIT_HEIGHT emis_heights.nc emish 2009-2010/1-12/1-31/0 C xy m 1
```

and refer to this scale factor as the upper bound of the injection height under *Base Emissions*:

```
0 GFAS_CO GFAS_201606.nc cofire 2009-2010/1-12/1-31/0 C xyL=1:scal300 kg/m2/s CO - 5 3
```

It should be noted that HEMCO always regrids the fields to the model grid before doing any data operations. If the emission height file is very spotty and contains a lot of zeros the averaged injection heights may be too low. In this case it may be required to set all zeros to missing values (which are ignored by HEMCO) to achieve the desired result.

Vertically distributing emissions

In HEMCO 3.0.0 and later versions, the capability to vertically allocate emissions has been added. To achieve this, HEMCO first copies emissions to all levels when dimensions `xyL*` are specified. Scale factors can then be applied to determine distribute the emissions vertically.

For example, let's assume that we have a file `vert_alloc.nc` containing the ratio of emissions to apply to each level for CEDS energy, industry, and ship emissions. We must add the following entries to under the *Scale Factors* section of the *the HEMCO configuration file*:

```
#==============================================================================
# --- CEDS vertical partitioning ---
#==============================================================================
(((CEDS
315 ENERGY_LEVS   vert_alloc.nc g_energy   2017/1/1/0 C xyz 1 1
316 INDUSTRY_LEVS vert_alloc.nc g_industry 2017/1/1/0 C xyz 1 1
317 SHIP_LEVS     vert_alloc.nc cmv_c3     2017/1/1/0 C xyz 1 1
)))CEDS
```

These scale factors are then applied to the `CEDS_*_ENE`, `CEDS_*_IND`, and `CEDS_*_SHIP` fields that are listed under *Base Emissions*. These fields are 2D in the CEDS data files, but we now can specify dimensions `xyL*` instead of `xy` to tell HEMCO to copy the field into each emissions level:

```
0 CEDS_CO_ENE CO-em-total-anthro_CEDS_$YYYY.nc  CO_ene  1970-2017/1-12/1/0 C xyL* kg/
↪m2/s CO 26/37/35/315 1  5
0 CEDS_CO_IND CO-em-total-anthro_CEDS_$YYYY.nc  CO_ind  1970-2017/1-12/1/0 C xyL* kg/
↪m2/s CO 26/316       1  5
0 CEDS_CO_SHP CO-em-total-anthro_CEDS_$YYYY.nc  CO_shp  1970-2017/1-12/1/0 C xyL*`kg/
↪m2/s CO 26/317       10 5
```

Mathematical expressions examples

You may define mathematical expressions in *the HEMCO configuration file*. Similar to uniform values, these must be placed in in the *sourceFile* column. All expressions are evaluated during run-time. They can be used e.g. to model an oscillating emission source. All mathematical expressions must contain at least one time-dependent variable that is evaluated on-the-fly. Mathematical expressions are specified by using the prefix `MATH:`, followed by the mathematical expression. The expression is a combination of variables, mathematical operations, and constants (e.g. `MATH:5.0+2.5*sin(HH)`.

Supported variables and operators

The following variable names and mathematical operations are currently supported:

Variable names

- `YYYY` (current year)
- `MM` (current month)
- `DD` (current day)
- `HH` (current hour)
- `NN` (current minute)
- `SS` (current second)

- SS (current second)
- DOY (day of year)
- DOM (days in current month)
- WD (Weekday: 0=Sun, 1=Mon .. 7=Sat)
- LH (hour in local time)
- PI (the constant PI)

Basic mathematical operators: + - / * ^ ( )

Advanced mathematical functions: *sin*, *cos*, *tan*, *asin*, *acos*, *atan*, *sinh*, *cosh*, *tanh*, *sind*, *cosd*, *tand*, *log*, *log10*, *nint*, *anint*, *aint*, *exp*, *sqrt*, *abs*, *floor*. The names refer to the equivalent Fortran functions.

Important: When using mathematical expressions, we recommend setting the *sourceTime* attribute to *, especially if you are using the short-term variables (HH, NN, SS, LH). This will ensure that your expression will get evaluated on every emission time step.

Example: Define a sinusoidal source

To define a sine-wave emission source of NO with an oscillation frequency of 24 hours, add the following line to section *Base Emissions* in *the HEMCO configuration file*. Place the mathematical expression under the *sourceFile* column (i.e. the 3rd column):

```
0 SINE_NO  MATH:sin(HH/12*PI) - * C xy kg/m2/s NO - 1 500
```

This defines an emission category (*Cat*) of 1 and hierarchy (*Hier*) of 500. No scale factors are applied.

Important: Mathematical expressions can produce negative emissions, which by default cause HEMCO to stop with an error. Negative emissions can be enabled by setting `Negative values:  2` in the *Settings* section of *the HEMCO configuration file*.

In order to avoid negative values, you may specify an offset, as is shown below:

```
0 SINE_NO  MATH:2.0+sin(HH/12*PI) - * C xy kg/m2/s NO - 1 500
```

Other examples

Assign emissions to passive species in an external model

The HEMCO passive species module allows you to run a suite of passive species alongside any simulation, i.e. it works with all simulation types. To use the passive species within GEOS-Chem, follow these steps:

Let's assume you are using HEMCO in an external model, and that you have two passive species named PASV1 and PASV2 that have constant emissions fluxes. Add the following entries to the *Base Emissions* section of *the HEMCO configuration file*:

```
# Assign PASV1 a flux of 0.001 kg/m2/s
0 PASV1_Flux 1.0e-3  - - - xy kg/m2/s PASV1 - 1 1
```

(continues on next page)

(continued from previous page)

```
# Assign PASV2 a flux of 1e-9 kg/m2/s
0 PASV2_Flux 1.0e-9  - - - xy kg/m2/s PASV2 - 1 1

# ... etc for additional species ...
```

To define emissions for passive species that are geographically tagged, simply assign corresponding mask values in the third-to-last column:

```
0 PASV1_Flux 1.0e-3  - - - xy kg/m2/s PASV1 1000 1 1
0 PASV2_Flux 1.0e-9  - - - xy kg/m2/s PASV2 1001 1 1

# ... etc for additional species...
```

Here, 1000 and 1001 refer to *mask definitions* in *the HEMCO configuration file*.

Next, request HEMCO diagnostic output. Define the following entries in the *diagnostics configuration file* (aka `HEMCO_Diagn.rc`):

```
# Name        Spec  ExtNr Cat  Hier  Dim  Unit     Longname
PASV1_TOTAL   PASV1 -1    -1   -1    2    kg/m2/s  PASV1_emission_flux
PASV2_TOTAL   PASV2 -1    -1   -1    2    kg/m2/s  PASV2_emission_flux

# ... etc for additional species ...
```

To activate these diagnostics, you must specify values for `DiagnFile` and `DiagnFreq` in the *Settings* section of *the HEMCO configuration file*:

```
DiagnFile:                HEMCO_Diagn.rc
DiagnFreq:                00000000 003000
```

The `DiagnFile` option tells HEMCO to read the diagnostic definitions in the file that you specify (the default is `HEMCO_Diagn.rc`). Use `DiagnFreq` to specify the diagnostic frequency (i.e. the interval at which diagnostics output will be created).

2.1.7 HEMCO under the hood

This section provides a short description of the main principles of HEMCO. More details are provided in the source code, and references to the corresponding modules is given where appropriate.

Overview

The HEMCO code can be broken up into three parts: *core code*, *extensions* and *interfaces*.

- The *core code* consists of all core modules that are essential for every HEMCO simulation.
- The *extensions* are a collection of emission parameterizations that can be optionally selected (e.g. dust emissions, air-sea exchange, etc.). Most of the extensions require meteorological variables (2D or 3D fields) passed from an atmospheric model or an external input file to HEMCO. (See the *HEMCO extensions* section for more information.)
- The *interfaces* are top-level routines that are only required in a given model environment (e.g. in stand-alone mode or under an ESMF framework). The HEMCO-model interface routines are located outside of the HEMCO code structure, calling down to the HEMCO driver routines for both the HEMCO core and extensions.

HEMCO stores all emission data (*base emissions*, *scale factors*, *masks*) in a generic data structure (a **HEMCO data container**). Input data read from disk is translated into this data structure by the HEMCO input/output module (`src/Core/hcoio_dataread_mod.F90`). This step includes unit conversion and regridding.

HEMCO data objects

All emission data (*Base emissions*, *Scale factors*, *Masks*) are internally stored in a **data container**. For each data element of *the HEMCO configuration file*, a separate data container object is created when reading the configuration file at the beginning of the simulation. The data container object is a Fortran derived type that holds information of one entry of the configuration file. All file data information such as filename, file variable, time slice options, etc. are stored in a `FileData` derived type object (defined in `src/Core/hco_filedata_mod.F90`). This object also holds a pointer to the data itself. All data is stored as 2 or 3 dimensional data arrays. HEMCO can keep multiple time slices in memory simultaneously, e.g. for diurnal scale factors, in which case a vector of data arrays is created. Data arrays are defined in module `/src/core/hco_arr_mod.F90`.

Data containers (and as such, emissions data) are accessed through three different linked lists: `ConfigList`, `ReadList`, and `EmisList`. These lists all point to the same content (i.e. the same containers) but ordered in a manner that is most efficient for the intended purpose:

- For example, `ReadList` contains sub-lists of all containers that need to be updated annually, monthly, daily, hourly, or never. Thus, if a new month is entered, only a few lists (monthly, daily and hourly) have to be scanned and updated instead of going through the whole list of data containers.
- Similarly, `EmisList` sorts the data containers by model species, emission category (*`Cat`*) and hierarchy (*`Hier`*) . This allows an efficient emission calculation since the EmisList has to be scanned only once.

List containers and generic linked list routines are defined in `src/Core/hco_datacont_mod.F90`. Specific routines for `ConfigList`, `ReadList` and `EmisList` are defined in `src/Core/hco_config_mod.F90`, `src/Core/hco_readlist_mod.F90`, and and `src/Core/hco_emislist_mod.F90` respectively.

Core code

HEMCO core consists of all routines and variables required to read, store, and update data used for emissions calculation. The driver routines to execute (initialize, run and finalize) a HEMCO core simulation are (see hco_driver_mod.F90: `HCO_INIT`, `HCO_RUN`, `HCO_FINAL`). These are also the routines that are called at the interface level (see *the HEMCO-to-model interface* section).

Each HEMCO simulation is defined by its state object `HcoState`, which is a derived type that holds all simulation information, including a list of the defined HEMCO species, emission grid information, configuration file name, and additional run options. More details on the HEMCO state object can be found in `src/Core/hco_state_mod.F90`. `HcoState` is defined at the interface level and then passed down to all HEMCO routines

Initialize: HCO_INIT

Before running HEMCO, all variables and objects have to be initialized properly. The initialization of HEMCO occurs in three steps:

1. Read the HEMCO configuration file (subroutine `Config_ReadFile` in `src/Core/hco_config_mod.F90`). This writes the content of the entire configuration file into buffer, and creates a data container for each data item (*base emission scale factor*, *mask*) in `ConfigList`.
2. Initialize `HcoState`.
3. Call `HCO_INIT`, passing `HcoState` to it. This initializes the HEMCO clock object (see `src/Core/hco_clock_mod.F90`) and creates the `ReadList` (`src/Core/hco_readlist_mod.F90`). The `ReadList` links to the data containers in `ConfigList`, but sorted by data update frequency. Data that is

not used at all (e.g. scale factors that are not used by any base emission, or regional emissions that are outside of the emission grid). The `EmisList` linked list is only created in the run call.

Note that steps 1 and 2 occur at the *the HEMCO-to-model interface level*.

Run: HCO_RUN

This is the main function to run HEMCO. It can be repeated as often as necessary. Before calling this routine, the internal clock object has to be updated to the current simulation time (subroutine `HcoClock_Set` in `src/Core/hco_clock_mod.F90`). `HCO_RUN` performs the following steps:

1. Updates the time slice index pointers. This is to make sure that the correct time slices are used for every data container. For example, hourly scale factors can be stored in a data container holding 24 individual 2D fields. Module `src/Core/hco_tidx_mod.F90` organizes how to properly access these fields.
2. Read/update the content of the data containers (`ReadList_Read`). Checks if there are any fields that need to be read/updated (e.g. if this is a new month compared to the previous time step) and updates these fields if so by calling the data interface (see *Interfaces*).
3. Creates/updates the `EmisList` object. Similar to `ReadList`, `EmisList` points to the data containers in `ConfigList`, but sorted according to species, emission hierarchy, emissions category. To optimize emission calculations, EmisList already combines :base emission fields that share the same species, category, hierarchy, scale factors, and field name (without the field name tag, see *Base Emissions*).
4. Calculate core emissions for the current simulation time. This is performed by subroutine `hco_calcemis` in `src/Core/hco_calc_mod.F90`. This routine walks through `EmisList` and calculates the emissions for every base emission field by applying the assigned scale factors to it. The (up to 10) container IDs of all scale factors connected to the given base emission field (as set in *the HEMCO configuration file*) are stored in the data container variable `ScalIDs`. A container ID index list is used to efficiently retrieve a pointer to each of those containers (see `cIDList` in `src/Core/hco_datacont_mod.F90`).

Finalize: HCO_FINAL

This routine cleans up all internal lists, variables, and objects. This does not clean up the HEMCO state object, which is removed at the interface level.

Extensions

HEMCO extensions are used to calculate emissions based on meteorological input variables and/or non-linear parameterizations. Each extension is provided in a separate Fortran module. Each module must contain a public subroutine to initialize, run and finalize the extension. Emissions calculated in the extensions are added to the HEMCO emission array using subroutine `HCO_Emis_Add` in `src/Core/HCO_FluxArr_mod.F90`.

Meteorological input data is passed to the individual extension routines through the extension state object ExtState, which provides a pointer slot for all met fields used by any of the extension (see `src/Extensions/hcox_state_mod.F90`). These pointers must be assigned at the interface level (see *the HEMCO-model interface section*).

In analogy to the core module, the three main routines for the extensions are (in `src/Extensions/hcox_driver_mod.F90`):

- `HCOX_Init`
- `HCOX_Run`
- `HCOX_Final`

These subroutines invoke the corresponding calls of all (enabled) *extensions* and must be called at the interface level (after the core routines).

Extension settings (as specified in the configuration file, see also *Extension switches*) areautomatically read by HEMCO. For any given extension, routines `GetExtNr` and `GetExtOpt` can be used to obtain the extension number (*`ExtNr`*) and desired setting value, respectively (see `src/Core/HCO_ExtList_Mod.F90`). Routine `HCO_GetExtHcoID` should be used to extract the HEMCO species IDs of all species registered for this extension.

Gridded data associated to an extension (i.e. listed in section extension data of the configuration file) is automatically added to the `EmisList`, but ignored by the HEMCO core module during emissions calculation. Pointers to these data arrays can be obtained through routine EmisList_GetDataArr in HCO_EmisList_Mod.F90. Note that this routine identifies the array based on its container name. It is therefore important that the container name set in the configuration file matches the names used by this routine!

Interfaces

HEMCO-to-model interface

Note: For additional information about coupling HEMCO to other models, please see our *Coupling HEMCO to other models* chapter.

The interface provides the link between HEMCO and the model environment. This may be a sophisticated Earth System model or a simple environment that allows the user to run HEMCO in standalone mode. The standalone interface is provided along with the HEMCO distribution (`src/Interfaces/hcoi_standalone_mod.F90`). The HEMCO-to-GEOS-Chem model interface is included in the GEOS-Chem source code (`GeosCore/hcoi_gc_main_mod.F90`). HEMCO has also been successfully employed as a stand-alone gridded component within an ESMF environment. Please contact Christoph Keller for more information on the ESMF implementation.

The interface routines provide HEMCO with all the necessary information to perform the emission calculation. This includes the following tasks:

Initialization:

- Read the *configuration file* (`Config_ReadFile` in `src/Core/hco_config_mod.F90`).
- Initialize `HcoState object` (`HcoState_Init` in `src/Core/hco_state_mod.F90`).
- Define the emission grid. Grid definitions are stored in `HcoState%Grid`. The emission grid is defined by its horizontal mid points and edges (all 2D fields), the hybrid sigma coordinate edges (3D), the grid box areas (2D), and the grid box heights. The latter is only used by some extensions (*`DustDead`*, *`LightNOx`*') and may be left undefined if those are not used.
- Define emission species. Species definitions are stored in vector `HcoState%Spc(:)` (one entry per species). For each species, the following parameter are required:
  1. HEMCO species ID: unique integer index for species identification. For internal use only.
  2. Model species ID: the integer index assigned to this species by the employed model.
  3. Species name
  4. Species molecular weight in g/mol.
  5. Emitted species molecular weight in g/mol. This value can be different to the species molecular weight if species are emitted on a molecular basis, e.g. in mass carbon (in which case the emitted molecular weight becomes 12 g/mol).

6. Molecular ratio: molecules of emitted species per molecules of species. For example, if C3H8 is emitted as kg C, the molecular ratio becomes 3.
7. K0: Liquid over gas Henry constant in M/atm.
8. CR: Temperature dependency of K0 in K.
9. pKa: The species pKa, used for correction of the Henry constant.

The molecular weight - together with the molecular ratio - determine the mass scaling factors used for unit conversion in hco_unit_mod.F90. The Henry coefficients are only used by the air-sea exchange extension (and only for the specified species) and may be left undefined for other species and/or if the extension is not used.

- Define simulation time steps. The emission, chemical and dynamic time steps can be defined separately.
- Initialize HEMCO core (`HCO_Init` in `src/Core/hco_driver_mod.F90`)
- Initialize HEMCO extensions (code:*HCOX_Init* in `src/Core/hcox_driver_mod.F90`)

Run:

- Set current time (`HcoClock_Set` in `src/Core/hco_clock_mod.F90`)
- Reset all emission and deposition values (`HCO_FluxArrReset` in `src/Core/hco_fluxarr_mod.F90`)
- Run HEMCO core to calculate emissions (`HCO_Run` in `src/Core/hco_driver_mod.F90`)
- Link the used meteorology field objects of `ExtState` to desired data arrays (this step may also be done during initialization)
- Run *HEMCO extensions* to add extensions emissions (`HCOX_Run` in `src/Core/hcox_driver_mod.F90`)
- Export HEMCO emissions into desired environment

Finalization:

- Finalize HEMCO extensions and extension state object ExtState (HCOX_Final in hcox_driver_mod.F90).
- Finalize HEMCO core (HCO_Final in hco_driver_mod.F90).
- Clean up HEMCO state object HcoState (HcoState_Final in hco_state_mod.F90).

Data interface (reading and regridding)

The data interface (in `src/Core/hcoi_dataread_mod.F90`) organizes reading, unit conversion, and remapping of data from source files. Its public routine `HCOI_DataRead` is only called by subroutine `ReadList_Fill` in `src/Core/hco_readlist_mod.F90`. Data processing is performed in three steps:

1. Read data from file using the source file information (file name, source variable, desired time stamp) provided in the configuration file.
2. Convert unit to HEMCO units based on the unit attribute read from disk and the srcUnit attribute set in the configuration file. See *Input file format* for more information.
3. Remap original data onto the HEMCO emission grid. The grid dimensions of the input field are determined from the source file. If only horizontal regridding is required, e.g. for 2D data or if the number of vertical levels of the input data is equal to the number of vertical levels of the HEMCO grid, the horizontal interpolation routine used by GEOS-Chem is invoked. If vertical regridding is required or to interpolate index-based values (e.g. discrete integer values), the NcRegrid tool described in *Joeckel (2006)* is used.

Run multiple instances of HEMCO

It is possible to run multiple instances of HEMCO at the same time. These instances can operate on different grids, use different configuration files, etc. This is made possible by wrapping all information of a HEMCO simulation into a `HCO_State` derived type object (defined in `src/Core/hco_state_mod.F90`). Similarly, all emission extension information is included in an `Ext_State` derived type (in `src/Extensions/hcox_state_mod.F90`). These two objects together fully define the HEMCO setup and are being passed to the top level HEMCO routines (INIT/RUN/FINALIZE), e.g.:

```
CALL HCO_Run( am_I_Root, HcoState, Phase, RC )
# ...etc ...
CALL HCOX_Run( am_I_Root, HcoState, ExtState, RC )
```

To run more than one HEMCO instance in parallel, one need to define multiple HcoState instances and then call each of these separately, e.g.:

```
CALL HCO_Run( am_I_Root, HcoStateA, Phase, RC )
CALL HCO_Run( am_I_Root, HcoStateB, Phase, RC )
# ... etc ...
```

The HEMCO state objects also carry the 3D emission arrays, and when using multiple instances one needs to ensure that these arrays are properly connected to the 'emission end user', e.g. PBL mixing routine, etc. In the GEOS-Chem implementation of HEMCO, the module hco_interface_mod.F90 (in GeosCore) provides the interface between HEMCO and GEOS-Chem: it is the owner of the HcoState and ExtState object, and contains a number of wrapper routines to exchange information between HEMCO and GEOS-Chem. In the GEOS model, the standalone HEMCO component uses a linked list that can carry a dynamic number of HEMCO instances, and then loops over the linked list to perform all model operations (init,run,finalize) on all members of the linked list.

Important: Several HEMCO extensions still use global arrays and currently cannot be used in multi-instance simulations. As of 8/29/2018, the following extensions are likely to cause problems in multi-instance simulations: Ginoux dust emissions, FINN biomass burning, GFED biomass burning, Iodine emissions, PARANOx ship emissions, sea flux emissions, sea salt emissions.

2.1.8 Input file format

Currently, HEMCO can read data from the following data sources:

1. **Gridded data from netCDF file**. More detail on the netCDF file are given below. In an ESMF environment, the MAPL/ESMF generic I/O routines are used to read/remap the data. In a non-ESMF environment, the HEMCO generic reading and remapping algorithms are used. Those support vertical regridding, unit conversion, and more (see below).
2. **Scalar data directly specified in the HEMCO configuration file.** Scalar values can be set in the HEMCO configuration file directly. If multiple values - separated by the separator sign (/) - are provided, they are interpreted as temporally changing values: 7 values = Sun, Mon, ..., Sat; 12 values = Jan, Feb, ..., Dec; 24 values = 12am, 1am, ..., 11pm (local time!). Mask box boundaries can also be provided directly in the HEMCO configuration file. The entry must have exactly four values, interpreted as lower left and upper right mask box corners (lon1/lat1/lon2/lat2).
3. **Country-specific data specified in a separate ASCII file.** This file must end with the suffix '.txt' and hold the country specific values listed by country ID. The IDs must correspond to the IDs of a corresponding (netCDF) mask file. The mask file must be listed in the HEMCO configuration file. For example:

```
#==============================================================================
# --- Country mask file ---
#==============================================================================
* COUNTRY_MASK $ROOT/MASKS/v2014-07/countrymask_0.1x0.1.nc CountryID 2000/1/1/0 C xy␣
↪count * - 1 1
```

In the .txt file containing the country-specific scale factors, the container name of this mask file (e.g. `COUNTRY_MASK`) must be given in the first line of the file. In that file, ID 0 is reserved for the default values that are applied to all countries with no specific values listed. The .txt file must be structured as follows:

```
# Country mask field name
COUNTRY_MASK

# CountryName CountryID CountryValues
DEFAULT       0         1.0/2.0/3.0/4.0/5.0/6.0/7.0
```

The `CountryValues` are interpreted the same way as scalar values, except that they are applied to all grid boxes with the given country ID.

COARDS compatibility

Gridded input files are expected to be in the Network Common Data Form (netCDF) format and must adhere to the COARDS metadata conventions

For an in-depth description of the COARDS netCDF conventions, please see the Supplemental Guide entitled *Prepare COARDS-compliant netCDF files*. Also be aware of some additional considerations for the *time* and *vertical level* dimensions.

Units of data variables

It is recommended to store data in one of the HEMCO standard units:

- `kg/m2/s` for fluxes;
- `kg/m3` for concentrations;
- `1` for unitless data;
- `count` for index-based data, i.e. discrete distributions (for instance, land types represented as integer values).

HEMCO will attempt to convert all data to one of those units, unless otherwise via the *`SrcUnit`* attribute (see the *Base Emissions*) section.

Mass conversion (e.g. from molecules to kg) is performed based on the properties (e.g. molecular weight) of the species assigned to the given data set. It is also possible to convert between species-based and molecule-based units (e.g. kg vs. kg(C)). This conversion is based on the emitted molecular weight and the molecular ratio of the given species (see the HEMCO-model Interface) section. More details on unit conversion are given in module `src/Core/hco_unit_mod.F90`.

Index-based data is regridded in such a manner that every grid box on the new grid represents the index with the largest relative contribution from the overlapping boxes of the original grid. All other data are regridded as "concentration: quantities, i.e. conserving the global weighted average.

For more information, we invite you to read our Preparing data files for use with HEMCO wiki page.

Arbitrary additional netCDF dimension

HEMCO can read netCDF files with an additional, arbitrary dimension. The dimension name and dimension index to be read must be given explicitly in the HEMCO configuration file as part of the *SrcDim* file attribute). This feature is currently not available in an ESMF environment.

Regridding

Vertical regridding

HEMCO is able to perform some limited vertical interpolation.

> **Warning: HEMCO assumes that the input data is on the same grid as the model grid if it has the same number (nz) of, or plus one (nz+1) vertical levels than the model.** In the case of the same number of vertical levels, HEMCO assumes that the input data is already on the model grid and no interpolation is performed. In the case of input data having nz+1 levels, the data is interpreted as being on grid edges instead of grid midpoints.

Collapsing into various GEOS grids. Additional vertical regridding options are available for the various GEOS grids (e.g. to regrid native GEOS-5 levels to reduced GEOS-5 levels, or to remap GEOS-5 data onto the vertical GEOS-4 grid). These options are only available if the corresponding compiler flags are set (this is the default case for GEOS-Chem users).

Conservative vertical interpolation using MESSy. If input data is specified with vertical coordinates in `lev` attribute of the netCDF file with units `atmosphere_hybrid_sigma_pressure_coordinate`, HEMCO can perform vertical interpolation using MESSy to the model grid.

Regridding GEOS-Chem 3-D input data in other models. In other models where HEMCO is used for emissions, but do not necessarily use the GEOS vertical grids (e.g., WRF-GC, GEOS-Chem within CESM, CAM-chem with HEMCO), input data from GEOS-Chem files which have 72 levels will automatically be regridded to the model levels, for compatibility.

By default, HEMCO assumes that the vertical coordinate direction is upwards, i.e. the first level index corresponds to the surface layer. The vertical axis can be reversed by setting the srcDim attribute in the HEMCO configuration file accordingly (e.g. xy-72 if the input data has 72 levels on a reversed vertical axis).

Horizontal regridding

In a non-ESMF environment, HEMCO can only regrid between rectilinear grids (e.g. lat-lon).

Nested HEMCO configuration files

HEMCO configuration files can be nested by adding an include statement to the master HEMCO configuration file (`HEMCO_Config.rc`), e.g.:

```
>>>include HEMCO_Config_nested.rc
```

The emission information contained in `HEMCO_Config_nested.rc` will then be used along with the emission configuration specified in `HEMCO_Config.rc`. Information in the master configuration file take precedence over the information in the nested files. If the same setting or extension switch/option is defined in both the master and the nested configuration file, HEMCO will use the one from the master file.

Include statements can be placed anywhere in the HEMCO configuration file. It is legal to nest multiple files (up to 5 levels deep).

2.1.9 Coupling HEMCO to other models

This page details technical information useful for developers who wish to couple **HEMCO** (the "Harmonized" Emissions Component) emissions component to other models.

The description of **HEMCO** coupling to other models is available in [[Lin et al., 2021]], which describes coupling to GEOS-Chem Classic, GCHP, WRF-GC, **CESM2-GC**, and future NOAA models.

Overview

This work is made possible by a restructuring of **HEMCO**, named HEMCO 3.0. HEMCO 3.0 separates model-specific components such as I/O, Regridding and the model speciation interface, into modular components, and isolate the HEMCO emissions Core.

This work is currently being actively worked on by the GEOS-Chem Support Team and Haipeng Lin (Harvard) as part of coupling GEOS-Chem with the CESM model.

Useful resources

- HEMCO Repository: geoschem/HEMCO on GitHub.
- Original description paper: [[Keller et al., 2014]].
- Coupling and HEMCO 3.0 description paper: [[Lin et al., 2021]].
- The HEMCO User's Guide
- HEMCO versions

Terminology

As part of the **HEMCO 3.0** restructuring, "HEMCO" is now divided into three pieces depending on their function:

- **The HEMCO Core.** Emissions calculations logic, containers, data types, etc.
- **Data Input Layer.** I/O (previously `HCOIO_Read/Write_*_Mod`), Regridding (`HCO_MESSY_REGRID`, `HCO_INTERP_MOD`), ... This will be rearranged into `Regrid/` and `IO/` folders in a future version. Right now due to dependencies, some of these files still live in the `Core/` folder.
- **Model Interface Layer.** Code that couples **HEMCO** with other models. There are common utilities available at `Interfaces/HCO_Interface_Common.F90`.

Note: Note that not all code pertinent to model coupling actually lives inside of **HEMCO**; this is by design, as data types that are external to **HEMCO** (i.e. GEOS-Chem types such as `State_Met`, CESM types such as `physics_state`, WRF types such as `domain`) must be maintained with the model and not inside HEMCO. Some code lives in `Interfaces/`, and some will live inside the model.

Technical Notes (Data Input Layer)

TBD

Technical Notes (Model Interface Layer)

HEMCO 3.0 Model Interface Layer Overview

In order to interface **HEMCO** with the target model, there are a few primary tasks that need to be performed as outlined below.

Data/code that needs to be provided to **HEMCO** based on the target model's data structures include:

- The clock and time-step of the target model
- List of species and physical properties (molecular weight required; other properties such as Henry's law constants are optional, only for extensions such as SeaFlux)
- Grid information (`I`, `J`, `L` atmospheric '0-D box' dimensions required; if using HEMCO built-in regrid, then specifics are needed. See below)

Data/code that needs to be **retrieved from HEMCO** into the target model's data structures (i.e. state object for constituent flux/concentrations) include:

- Emissions fluxes (kg/m2/s format) retrieved from HEMCO, aggregated per species ID, for current time step
- Other data retrieved from HEMCO (using `HCO_GetPtr` or `HCO_EvalFld`)

Important: Avoid calling HEMCO functions directly from outside of a specific module designed to interface HEMCO with the model. This is so the interface can be updated more easily if subroutines within HEMCO such as `HCO_GetPtr` change, and the HEMCO state (:code`HcoState`) doesn't need to be passed to everywhere in your model that needs to retrieve data from HEMCO. **It is also useful so regridding to/from HEMCO can be performed in a centralized location, if so needed by the model.** For example, GEOS-Chem wraps `HCO_GetPtr` and `HCO_EvalFld` into its own interface, `HCO_GC_GetPtr`, `HCO_GC_EvalFld`, which will auto-magically add the `HcoState` argument, in addition to handling regridding if necessary.

Things that come out-of-the-box and generally do not require customization to a specific model:

- Reading configuration file (`HEMCO_Config.rc`), although the path needs to be specified
- HEMCO "driver" (run) routines
- Managing HEMCO memory (initializing HEMCO state in `HcoState`, extensions state in `ExtState`, etc.)

Reading the HEMCO configuration file and defining species list

This is a three-step process. First initialize the configuration object (`HcoConfig`):

```
call ConfigInit(HcoConfig, HMRC, nModelSpecies=nSpc)
```

You have to register the species first in addition to some other HcoConfig properties:

```
HcoConfig%amIRoot   = masterproc
HcoConfig%MetField  = 'MERRA2'
HcoConfig%GridRes   = ''
HcoConfig%nModelSpc = nHcoSpc
```

(continues on next page)

(continued from previous page)

```
HcoConfig%nModelAdv = nHcoSpc            ! # of adv spc?

do N = 1, nHcoSpc
   HcoConfig%ModelSpc(N)%ModID   = N ! model id
   HcoConfig%ModelSpc(N)%SpcName = trim(solsym(N))
enddo
```

Then open the configuration file in two phases; after phase 1, initialize the log file on the MPI root process:

```
call Config_ReadFile(HcoConfig%amIRoot, HcoConfig, HcoConfigFile, 1, HMRC, IsDryRun=.
↪false.)

! Open the log file
if(masterproc) then
   call HCO_LOGFILE_OPEN(HcoConfig%Err, RC=HMRC)
endif

call Config_ReadFile(HcoConfig%amIRoot, HcoConfig, HcoConfigFile, 2, HMRC, IsDryRun=.
↪false.)
```

Warning: **Note that the species count has to be populated three times.** Once above at `ConfigInit`, and twice inside the *initialized HEMCO Config object*.

Some species physical properties need to be defined for **HEMCO** extensions, such as molecular weight and henry's law constants:

```
!-----------------------------------------------------------------------
! Register HEMCO species information (HEMCO state object)
!-----------------------------------------------------------------------
do N = 1, nHcoSpc
    HcoState%Spc(N)%ModID          = N                ! model id
    HcoState%Spc(N)%SpcName        = trim(solsym(N)) ! species name
    HcoState%Spc(N)%MW_g           = adv_mass(N)      ! mol. weight [g/mol]

    ! HcoState%Spc(N)%HenryK0 ! [M/atm]
    ! HcoState%Spc(N)%HenryCR ! [K]
    ! HcoState%Spc(N)%HenryPKA ! [1]
enddo
```

Note: If you are not using HEMCO extensions, only `ModID`, `SpcName` and `MW_g` need to be defined.

Defining Grid

Define atmospheric column numbers

```
HcoState%NX = my_IM
HcoState%NY = my_JM
HcoState%NZ = LM
```

Define the vertical grid

There are many ways of defining the vertical discretization. Check `HCO_VertGrid_Define`.

```
! Pass Ap, Bp values, units [Pa], [unitless]
call HCO_VertGrid_Define(HcoState%Config,                &
                         zGrid = HcoState%Grid%zGrid,    &
                         nz    = HcoState%NZ,            &
                         Ap    = Ap,                     &
                         Bp    = Bp,                     &
                         RC    = HMRC)
```

Define horizontal grid parameters

Note: HEMCO **requires HORIZONTAL grid information only if it is using internal regridding routines**, i.e. `MAP_A2A` or MESSy. Otherwise, this can be filled with dummy information.

Warning: If **HEMCO** internal regridding (`MAP_A2A`) regridding routines are used, **only rectilinear grids are supported.**

This is because `XMid`, `YMid`, ... arrays are **1-dimensional** and thus curvilinear coordinates cannot be stored. The underlying `MAP_A2A` algorithm **can** handle curvilinear; it is just due to the data structure. This will be fixed in a future HEMCO version.

```
! Point to grid variables
HcoState%Grid%XMID%Val          => XMid    (my_IS:my_IE  , my_JS:my_JE  )
HcoState%Grid%YMID%Val          => YMid    (my_IS:my_IE  , my_JS:my_JE  )
HcoState%Grid%XEdge%Val         => XEdge   (my_IS:my_IE+1, my_JS:my_JE  )
HcoState%Grid%YEdge%Val         => YEdge   (my_IS:my_IE  , my_JS:my_JE+1)
HcoState%Grid%YSin%Val          => YSin    (my_IS:my_IE  , my_JS:my_JE+1)
HcoState%Grid%AREA_M2%Val       => AREA_M2 (my_IS:my_IE  , my_JS:my_JE  )
```

Here we point **HEMCO**'s variables to structures we have created in the model. Examples in how to create these structures are available in the HEMCO-CESM interface.

Defining Met Fields for HEMCO Extensions

An example to translate and define meteorological quantities such as temperature, humidity, etc. is available in the HEMCO-CESM interface.

Running HEMCO

Prerequisites:

```
! HEMCO
use HCO_Interface_Common,    only: GetHcoVal, GetHcoDiagn
use HCO_Clock_Mod,           only: HcoClock_Set, HcoClock_Get
use HCO_Clock_Mod,           only: HcoClock_EmissionsDone
use HCO_Diagn_Mod,           only: HcoDiagn_AutoUpdate
use HCO_Driver_Mod,          only: HCO_Run
use HCO_EmisList_Mod,        only: Hco_GetPtr
use HCO_FluxArr_Mod,         only: HCO_FluxArrReset
use HCO_GeoTools_Mod,        only: HCO_CalcVertGrid, HCO_SetPBLm
```

Update the HEMCO clock

Also make sure the time steps are set correctly. Use from the common utilities:

```
call HCOClock_Set(HcoState, year, month, day,  &
                  hour, minute, second, IsEmisTime=.true., RC=HMRC)
```

Reset fluxes for new timestep

```
call HCO_FluxArrReset(HcoState, HMRC)
```

Update vertical grid parameters

HEMCO needs an updated vertical grid at each time step. Data passed into `HCO_CalcVertGrid` can vary and the definition can be checked for acceptable parameters.

```
call HCO_CalcVertGrid(HcoState, PSFC, ZSFC, TK, BXHEIGHT, PEDGE, HMRC)

call HCO_SetPBLm(HcoState, PBLM=State_HCO_PBLH, &
                 DefVal=1000.0_hp, & ! default value
                 RC=HMRC)
```

Some dummy setup (advanced)

To document.

```
! Range of species and emission categories.
! Set Extension number ExtNr to 0, indicating that the core
! module shall be executed.
HcoState%Options%SpcMin = 1
HcoState%Options%SpcMax = -1
HcoState%Options%CatMin = 1
HcoState%Options%CatMax = -1
HcoState%Options%ExtNr  = 0

! Use temporary array?
HcoState%Options%FillBuffer = .FALSE.
```

Run HEMCO driver

```
call HCO_Run( HcoState, 1, HMRC, IsEndStep=.false. )
call HCO_Run( HcoState, 2, HMRC, IsEndStep=.false. )
```

Run HEMCO extensions driver

Necessary only if you are using **HEMCO** extensions.

```
call HCOX_Run(HcoState, ExtState, HMRC)
```

Close timestep

```
!-----------------------------------------------------------------------
! Update "autofill" diagnostics.
! Update all 'AutoFill' diagnostics. This makes sure that all
! diagnostics fields with the 'AutoFill' flag are up-to-date. The
! AutoFill flag is specified when creating a diagnostics container
! (Diagn_Create).
!-----------------------------------------------------------------------
call HcoDiagn_AutoUpdate(HcoState, HMRC)

!-----------------------------------------------------------------------
! Tell HEMCO we are done for this timestep...
!-----------------------------------------------------------------------
call HcoClock_EmissionsDone(HcoState%Clock, HMRC)
```

Retrieving emissions data from HEMCO

You can either use the common utilities, where data is retrieved using `GetHcoValEmis`, or tap into the arrays directly.

For generic data containers, pass the container name like so:

```
! For grabbing data from HEMCO Ptrs (uses HEMCO single-precision)
real(sp), pointer                       :: Ptr2D(:,:)
real(sp), pointer                       :: Ptr3D(:,:,:)

logical                                 :: FND

call HCO_GetPtr(HcoState, 'CONTAINER_NAME', Ptr2D, HMRC, FOUND=FND)
```

Retrieving deposition velocities (depv) from HEMCO

Warning: Important: Note that deposition (sink terms) fluxes are handled separately from emissions in HEMCO. This is particularly important if you use HEMCO to calculate deposition terms, e.g. the sink term in `SeaFlux` (sea-air exchange). The standard in HEMCO is that the sink terms are stored as deposition velocities (`depv`, unit `1/s`) so HEMCO generally does not need to be aware of concentrations.

A thorough discussion of this is in the HEMCO GitHub issue tracker. The code to handle deposition velocities from HEMCO is generally as follows:

```
!-----------------------------------------------------------------
! Also add drydep frequencies calculated by HEMCO (e.g. from the
! air-sea exchange module) to DFLX.  These values are stored
! in 1/s.  They are added in the same manner as the drydep freq values
! from drydep_mod.F90.  DFLX will be converted to kg/m2/s later.
! (ckeller, 04/01/2014)
!-----------------------------------------------------------------
CALL GetHcoValDep( NA, I, J, L, found, dep )
IF ( found ) THEN
   dflx(I,J,NA) = dflx(I,J,NA)                                    &
                + ( dep * spc(I,J,NA) / (AIRMW / ThisSpc%MW_g)  )
ENDIF
```

2.1.10 Known bugs and issues

Please see our HEMCO issue tracker on Github for a list of recent HEMCO bugs and fixes.

Current bug reports

These bug reports (listed on the HEMCO issue tracker) are currently unresolved. We hope to fix these in future HEMCO releases.

Masks cannot be applied to extensions

It is currently not possible to *geographically tag emissions* computed by *HEMCO extensions* in the same way that you would do for *base emissions*. We hope to add this feature into a future HEMCO release.

HEMCO may not recognize alternate spellings of units

If a unit string (e.g. `kg/m2/s`) read from a netCDF file matches the unit string listed under the *`SrcUnit`* column of *the HEMCO configuration file*, then no unit conversion will happen.

But if the unit string in the file is e.g. `kg m-2 s-1` and the unit in the configuration file is `kg/m2/s`, then HEMCO detects this as a difference in units, and will try to apply an automatic conversion that is really unnecssary.

Therefore, we recommend not to rely on HEMCO's automatic unit capability, and to specfiy all scale factors for unit conversions explicitly in the configuration file.

2.1.11 HEMCO version history

Please see the CHANGELOG.md file the HEMCO GitHub repository for a list of updates by HEMCO version.

2.1.12 Key References

- GEOS-Chem was first described in Bey *et al.* [[Bey et al., 2001]].
- HEMCO is described in Keller *et al.* [[Keller et al., 2014]] and Lin *et al.* [[Lin et al., 2021]].

Other references for GEOS-Chem are available on the GEOS-Chem website. A list of references for current HEMCO emission inventories is available in Table 1 of Lin et al., 2021. References for emissions inventories cited in HEMCO examples are included below.

References

CHAPTER

THREE

LOAD REQUIRED LIBRARIES

This supplemental guide describes the how to load the required software dependencies for **GEOS-Chem** and **HEMCO** into your computational environment.

3.1 On the Amazon Web Services Cloud

All of the required software libraries for **GEOS-Chem** and **HEMCO** will be included in the Amazon Machine Image (AMI) that you use to initialize your Amazon Elastic Cloud Compute (EC2) instance. For more information, please see our our GEOS-Chem cloud computing tutorial.

3.2 On a shared computer cluster

If you plan to use **GEOS-Chem** or **HEMCO** on a shared computational cluster (e.g. at a university or research institution), then there is a good chance that your IT staff will have already installed several of the required libraries.

Depending on your system's setup, there are a few different ways in which you can make these libraries available for use in your computational environment. These are described in the following sections:

3.2.1 Check if libraries are available as modules

Many high-performance computing (HPC) clusters use a module manager such as Lmod or environment-modules to load software packages and libraries. A module manager allows you to load different compilers and libraries with simple commands.

One downside of using a module manager is that you are locked into using only those compiler and software versions that have already been installed on your system by your IT staff. But in general, module managers succeed in ensuring that only well-tested compiler/software combinations are made available to users.

Example: Load modules for GNU compilers 8.2.0

If your computer system uses **lmod**, you can load software packages into your computational environment with **module load** commands, such as:

```
$ module purge
$ module load git/2.17.0-fasrc01
$ module load gcc/8.2.0-fasrc01
$ module load openmpi/3.1.1-fasrc01
$ module load netcdf/4.1.3-fasrc02
```

(continues on next page)

(continued from previous page)

```
$ module load perl/5.26.1-fasrc01
$ module load cmake/3.17.3-fasrc01
```

In this example (from the Harvard Cannon cluster), the version number and build identifier are part of the module name. This setup may differ on your system.

Tip: Consult your computer system documentation for more information on software package names.

Here is a summary of what the above commands do:

module purge
: Removes all previously loaded modules

module load git/...
: Loads Git (version control system)

module load gcc/...
: Loads the GNU Compiler Collection (suite of C, C++, and Fortran compilers)

module load openmpi/...
: Loads the OpenMPI library (a dependency of netCDF)

module load netcdf/..
: Loads the netCDF library

  Important: Depending on how the netCDF libraries have been installed on your system, you might also need to load the netCDF-Fortran library separately, e.g.:

  ```
  module load netcdf-fortran/...
  ```

module load perl/...
: Loads Perl (scripting language)

module load cmake/...
: Loads Cmake (needed to compile GEOS-Chem)

3.2.2 Check if Spack-built libraries are available

If your system doesn't have a module manager installed, check to see if the required libraries for **GEOS-Chem** and **HEMCO** were built the Spack package manager. Type

```
$ spack find
```

to locate any Spack-built software libraries on your system. If there Spack-built libraries are found, you may present, you may load them into your computational environment with **spack load** commands:

```
$ spack load gcc@10.2.0
$ spack load netcdf-c%gcc@10.2.0
$ spack load netcdf-fortran%gcc@10.2.0
... etc ...
```

When loading a Spack-built library, you can specify its version number. For example, **spack load gcc@10.2.0** tells Spack to load the GNU Compiler Collection version 10.2.0.

You may also specify a library by the compiler it was built with. For example, **`spack load netcdf-fortran%gcc@10.2.0`** tells Spack to load the version of netCDF-Fortran that was built with GNU Compiler Collection version 10.2.0.

These specification methods are often necessary to select a given library in case there are several available builds to choose from.

We recommend that you place **`spack load`** commands into an environment file.

If a Spack environment has been installed on your system, type:

```
spack env activate -p ENVIRONMENT-NAME
```

to load all of the libraries in the environment together.

To deactivate the environment, type:

```
spack deactivate
```

3.2.3 Check if libaries have been manually installed

If your computer system does not use a module manager and does not use Spack, check for a manual library installation. Very often, common software libraries are installed into standard locations (such as the `/usr/lib` or `/usr/local/lib` system folders). Ask your sysadmin for more information.

Once you know the location of the compiler and netCDF libraries, you can set the proper environment variables for GEOS-Chem and HEMCO.

3.2.4 If there are none of these, install them with Spack

If your system has none of the required software packages that **`GEOS-Chem`** and **`HEMCO`** need, then we recommend that you *use Spack to build the libraries yourself*. Spack makes the process easy and will make sure that all software dependences are resolved.

Once you have installed the libraries with Spack, you can load the libraries into your computational environment *as described above*.

CHAPTER

FOUR

BUILD LIBRARIES WITH SPACK

Here are some up-to-date instructions on installing a software stack for **GEOS-Chem Classic** or **HEMCO** with Spack.

Note: If you will be using GCHP, please see gchp.readthedocs.io for instructions on how to download required libraries with Spack.

4.1 Initial Spack setup

4.1.1 Install spack to your home directory

Spack can be installed with Git, as follows:

```
cd ~
$ git clone git@github.com:spack/spack.git
```

4.1.2 Initialize Spack

To initialize Spack type these commands:

```
$ export SPACK_ROOT=${HOME}/spack
$ source ${SPACK_ROOT}/spack/share/spack/setup-env.sh
```

4.1.3 Make sure the default compiler is in compilers.yaml

Tell Spack to search for compilers:

```
$ spack compiler find
```

You can confirm that the default compiler was found by inspecing `compilers.yaml` file with your favorite editor, e.g.:

```
$ emacs ~/.spack/linux/compilers.yaml
```

For example, the default compiler that was on my cloud instance was the GNU Compiler Collection 7.4.0. This collection contains C (**gcc**), C++ (:program`g++`), and Fortran (**gfortran**) compilers. These are specified in the `compiler.yaml` file as:

```
compilers:
- compiler:
    spec: gcc@7.4.0
    paths:
      cc: /usr/bin/gcc-7
      cxx: /usr/bin/g++-7
      f77: /usr/bin/gfortran-7
      fc: /usr/bin/gfortran-7
    flags: {}
    operating_system: ubuntu18.04
    target: x86_64
    modules: []
    environment: {}
    extra_rpaths: []
```

As you can see, the default compiler executables are located in the `/usr/bin` folder. This is where many of the system-supplied executable files are located.

4.2 Build the GCC 10.2.0 compilers

Let's build a newer compiler verion with Spack. In this case we'll build the GNU Compiler Collection 10.2.0 using the default compilers.

```
$ spack install gcc@10.2.0 target=x86_64 %gcc@7.4.0
$ spack load gcc%10.2.0
```

4.2.1 Update compilers.yaml

In order for Spack to use this new compiler to build other packages, the `compilers.yaml` file must be updated using these commands:

```
$ spack load gcc@10.2.0
$ spack compiler find
```

4.3 Install required libraries for GEOS-Chem

Now that we have installed a the GNU Compiler Collection 10.2.0, we can use it to build the required libraries for **GEOS-Chem Classic** and **HEMCO**.

4.3.1 HDF5

Now we can start installing libraries. First, let's install **HDF5**, which is a dependency of **netCDF**.

```
$ spack install hdf5%gcc@10.2.0 target=x86_64 +cxx+fortran+hl+pic+shared+threadsafe
$ spack load hdf5%gcc@10.2.0
```

The `+cxx+fortran+hl+pic+shared+threadsafe` specifies necessary options for building HDF5.

4.3.2 netCDF-Fortran and netCDF-C

Now that we have installed :program:, we may proceed to installing **netCDF-Fortran** (which will install **netCDF-C** as a dependency).

```
$ spack install netcdf-fortran%gcc@10.2.0 target=x86_64 ^
↪hdf5+cxx+fortran+hl+pic+shared+threadsafe
$ spack load netcdf-fortran%gcc@10.2.0
$ spack load netcdf-c%gcc@10.2.0
```

We tell Spack to use the same version of HDF5 that we just built by appending `^hdf5+cxx+fortran+hl+pic+shared+threadsafe` to the spack install command. Otherwise, Spack will try to build a new version of HDF5 with default options (which is not what we want).

4.3.3 ncview

Ncview is a convenient viewer for browsing netCDF files. Install it with:

```
$ spack install ncview%gcc@10.2.0 target=x86_64 ^
↪hdf5+cxx+fortran+hl+pic+shared+threadsafe
$ spack load ncview%gcc@10.2.0
```

4.3.4 nco (The netCDF Operators)

The netCDF operators (**nco**) are useful programs for manipulating netCDF files and attributes. Install (**nco**) with:

```
$ spack install nco%gcc@10.2.0 target=x86_64 ^
↪hdf5+cxx+fortran+hl+pic+shared+threadsafe
$ spack load nco%gcc@10.2.0
```

4.3.5 cdo (The Climate Data Operators)

The Climate Data Operators (**cdo**) are utilities for processing data in netCDF files.

```
$ spack install cdo%gcc@10.2.0 target=x86_64 ^
↪hdf5+cxx+fortran+hl+pic+shared+threadsafe
$ spack load cdo%gcc@10.2.0
```

4.3.6 flex

The **flex** library is a lexical parser. It is a dependency for The Kinetic PreProcessor (KPP).

```
$ spack install flex%gcc@10.2.0 target=x86_64
$ spack load flex%gcc10.2.0
```

gdb and cgdb

Gdb is the GNU Debugger. **Cgdb** is a visual, user-friendly interface for **gdb**.

```
$ spack install gdb@9.1%gcc@10.2.0 target=x86_64
$ spack load gdb%10.2.0

$ spack install cgdb%gcc@10.2.0 target=x86_64
$ spack load cgdb%gcc@10.2.0
```

cmake and gmake

Cmake and **gmake** are used to build source code into executables.

```
$ spack install cmake%gcc@10.2.0 target=x86_64
$ spack load cmake%gcc@10.2.0

$ spack install gmake%gcc@10.2.0 target=x86_64
$ spack load gmake%gcc@10.2.0
```

4.4 Installing optional packages

These packages are useful not strictly necessary for GEOS-Chem.

4.4.1 OpenJDK (Java)

Some programs might need the **openjdk** Java Runtime Environment:

```
$ spack install openjdk%gcc@10.2.0
$ spack load openjdk%gcc@10.2.0
```

4.4.2 TAU performance profiler

The Tuning and Analysis Utilities (;program:*tau*) lets you profile **GEOS-Chem** and **HEMCO** in order to locate computational bottlenecks:

```
$ spack install tau%gcc@10.2.0 +pthread+openmp~otf2
$ spack load tau%gcc@10.2.0
```

4.5 Loading Spack packages at startup

4.5.1 Creating an environment file for Spack

Once you have finished installing libraries with **Spack**, you can create an environment file to load the Spack libraries whenever you start a new Unix shell. Here is a sample environment file that can be used (or modified) to load the Spack libraries described above.

```
#==============================================================================
# %%%%% Clear existing environment variables %%%%%
#==============================================================================
unset CC
unset CXX
unset EMACS_HOME
unset FC
unset F77
unset F90
unset NETCDF_HOME
unset NETCDF_INCLUDE
unset NETCDF_LIB
unset NETCDF_FORTRAN_HOME
unset NETCDF_FORTRAN_INCLUDE
unset NETCDF_FORTRAN_LIB
unset OMP_NUM_THREADS
unset OMP_STACKSIZE
unset PERL_HOME

#==============================================================================
# %%%%% Load Spack packages %%%%%
#==============================================================================
echo "Loading gfortran 10.2.0 and related libraries ..."

# Initialize Spack
# In the examples above /path/to/spack was ${HOME}/spack
export SPACK_ROOT=/path/to/spack
source $SPACK_ROOT/share/spack/setup-env.sh

# List each Spack package that you want to load
# (add the backslash after each new package that you add)
pkgs=(                          \
  gcc@10.2.0                    \
  cmake%gcc@10.2.0              \
  openmpi%gcc@10.2.0            \
  netcdf-fortran%gcc@10.2.0     \
  netcdf-c%gcc@10.2.0           \
  hdf5%gcc@10.2.0               \
  gdb%gcc@10.2.0                \
  flex%gcc@10.2.0               \
  openjdk%gcc@10.2.0            \
  cdo%gcc@10.2.0                \
  nco%gcc@10.2.0                \
  ncview%gcc@10.2.0             \
  perl@5.30.3%gcc@10.2.0        \
  tau%gcc@10.2.0                \
)

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

#==============================================================================
# %%%%% Settings for OpenMP parallelization %%%%%
#==============================================================================
```

(continues on next page)

(continued from previous page)

```
# Max out the stack memory for OpenMP
# Asking for a huge number will just give you the max availble
export OMP_STACKSIZE=500m

# By default, set the number of threads for OpenMP parallelization to 1
export OMP_NUM_THREADS=1

# Redefine number threads for OpenMP parallelization
# (a) If in a SLURM partition, set OMP_NUM_THREADS = SLURM_CPUS_PER_TASK
# (b) Or, set OMP_NUM_THREADS to the optional first argument that is passed
if [[ -n "${SLURM_CPUS_PER_TASK+1}" ]]; then
  export OMP_NUM_THREADS=${SLURM_CPUS_PER_TASK}
elif [[ "$#" -eq 1 ]]; then
  if [[ "x$1" != "xignoreeof" ]]; then
    export OMP_NUM_THREADS=${1}
  fi
fi
echo "Number of OpenMP threads: $OMP_NUM_THREADS"

#=============================================================================
# %%%%% Define relevant environment variables %%%%%
#=============================================================================

# Compiler environment variables
export FC=gfortran
export F90=gfortran
export F77=gfortran
export CC=gcc
export CXX=g++

# Machine architecture
export ARCH=`uname -s`

# netCDF paths
export NETCDF_HOME=`spack location -i netcdf-c%gcc@10.2.0`
export NETCDF_INCLUDE=${NETCDF_HOME}/include
export NETCDF_LIB=${NETCDF_HOME}/lib

# netCDF-Fortran paths
export NETCDF_FORTRAN_HOME=`spack location -i netcdf-fortran%gcc@10.2.0`
export NETCDF_FORTRAN_INCLUDE=${NETCDF_FORTRAN_HOME}/include
export NETCDF_FORTRAN_LIB=${NETCDF_FORTRAN_HOME}/lib

# Other important paths
export GCC_HOME=`spack location -i gcc@10.2.0`
export MPI_HOME=`spack location -i openmpi%gcc@10.2.0`
export TAU_HOME=`spack location -i tau%gcc@10.2.0`

#=============================================================================
# %%%%% Echo relevant environment variables %%%%%
#=============================================================================
echo
echo "Important environment variables:"
echo "CC  (C compiler)       : $CC"
echo "CXX (C++ compiler)     : $CXX"
echo "FC  (Fortran compiler) : $FC"
```

(continues on next page)

(continued from previous page)

```
echo "NETCDF_HOME             : $NETCDF_HOME"
echo "NETCDF_INCLUDE          : $NETCDF_INCLUDE"
echo "NETCDF_LIB              : $NETCDF_LIB"
echo "NETCDF_FORTRAN_HOME     : $NETCDF_FORTRAN_HOME"
echo "NETCDF_FORTRAN_INCLUDE  : $NETCDF_FORTRAN_INCLUDE"
echo "NETCDF_FORTRAN_LIB      : $NETCDF_FORTRAN_LIB"
```

Save this to your home folder with a name such as `~/.spack_env`. The `.` in front of the name will make it a hidden file like your `.bashrc` or `.bash_aliases`.

4.5.2 Loading Spack-built libraries

Whenever you start a new Unix session (either by opening a terminal window or running a new job), your `.bashrc` and `.bash_aliases` files will be sourced, and the commands contained within them applied. You should then load the Spack modules by typing at the terminal prompt:

```
$ source ~/.spack.env
```

You can also add some code to your `.bash_aliases` so that this will be done automatically:

```
if [[ -f ~/.spack.env ]]; then
    source ~/.spack.env
fi
```

In either case, this will load the modules for you. You should see output similar to:

```
Loading gfortran 10.2.0 and related libraries ...
Loading gcc@10.2.0
Loading cmake%gcc@10.2.0
Loading openmpi%gcc@10.2.0
Loading netcdf-fortran%gcc@10.2.0
Loading netcdf-c%gcc@10.2.0
Loading hdf5%gcc@10.2.0
Loading gdb%gcc@10.2.0
Loading flex%gcc@10.2.0
Loading openjdk%gcc@10.2.0
Loading cdo%gcc@10.2.0
Loading nco%gcc@10.2.0
Loading ncview%gcc@10.2.0
Loading perl@5.30.3%gcc@10.2.0
Loading tau%gcc@10.2.0
Number of OpenMP threads: 1

Important environment variables:
CC  (C compiler)        : gcc
CXX (C++ compiler)      : g++
FC  (Fortran compiler) : gfortran
NETCDF_HOME             : /net/seasasfs02/srv/export/seasasfs02/share_root/ryantosca/
 ↪spack/opt/spack/linux-centos7-x86_64/gcc-10.2.0/netcdf-c-4.7.4-
 ↪22bkbtqledcaipqc2zrgun4qes7kkm5q
NETCDF_INCLUDE          : /net/seasasfs02/srv/export/seasasfs02/share_root/ryantosca/
 ↪spack/opt/spack/linux-centos7-x86_64/gcc-10.2.0/netcdf-c-4.7.4-
 ↪22bkbtqledcaipqc2zrgun4qes7kkm5q/include
NETCDF_LIB              : /net/seasasfs02/srv/export/seasasfs02/share_root/ryantosca/
 ↪spack/opt/spack/linux-centos7-x86_64/gcc-10.2.0/netcdf-c-4.7.4-
 ↪22bkbtqledcaipqc2zrgun4qes7kkm5q/lib
```

(continues on next page)

(continued from previous page)

```
NETCDF_FORTRAN_HOME    : /net/seasasfs02/srv/export/seasasfs02/share_root/ryantosca/
↪spack/opt/spack/linux-centos7-x86_64/gcc-10.2.0/netcdf-fortran-4.5.3-
↪mtuoejjcl3ozbvd6prgqm44k5jre3hne
NETCDF_FORTRAN_INCLUDE : /net/seasasfs02/srv/export/seasasfs02/share_root/ryantosca/
↪spack/opt/spack/linux-centos7-x86_64/gcc-10.2.0/netcdf-fortran-4.5.3-
↪mtuoejjcl3ozbvd6prgqm44k5jre3hne/include
NETCDF_FORTRAN_LIB     : /net/seasasfs02/srv/export/seasasfs02/share_root/ryantosca/
↪spack/opt/spack/linux-centos7-x86_64/gcc-10.2.0/netcdf-fortran-4.5.3-
↪mtuoejjcl3ozbvd6prgqm44k5jre3hne/lib
```

Once you see this output, you can then start using programs that rely on these Spack-built libraries.

4.5.3 Setting the number of cores for OpenMP

If you type:

```
$ source ~/.spack.env
```

by itself, this will set the `OMP_NUM_THREADS` variable to 1. This variable sets the number of computational cores that OpenMP should use.

You can change this with, e.g.

```
source ~/.spack.env 6
```

which will set `OMP_NUM_THREADS` to 6. In this case, GEOS-Chem Classic (and other programs that use OpenMP parallelization) will parallelize with 6 cores.

If you are using the SLURM scheduler and are source `.spack.env` in your job script, then `OMP_NUM_THREADS` will be automatically set to `SLURM_CPUS_PER_TASK`, which is then number of cores requested. If you are not using SLURM then you should add e.g.

```
export OMP_NUM_THREADS=6
```

(or however many cores you have requested) in your SLURM job script.

CHAPTER

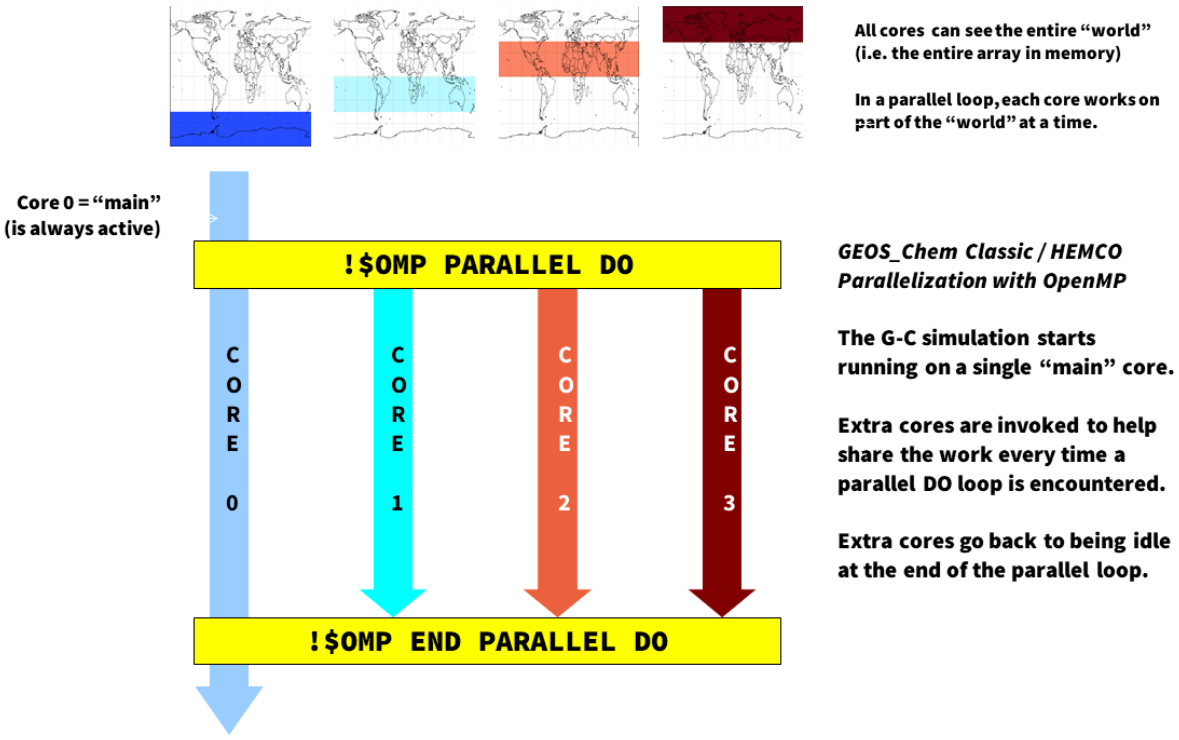
FIVE

PARALLELIZE GEOS-CHEM AND HEMCO SOURCE CODE

Single-node paralellization in GEOS-Chem Classic and HEMCO is acheieved with OpenMP. OpenMP directives, which are included in every modern compiler, allow you to divide the work done in DO loops among several computational cores. In this Guide, you will learn more about how GEOS-Chem Classic and HEMCO utilize OpenMP.

5.1 Overview of OpenMP parallelization

Most GEOS-Chem and HEMCO arrays represent quantities on a geospatial grid (such as meteorological fields, species concentrations, production and loss rates, etc.). When we parallelize the GEOS-Chem and HEMCO source code, we give each computational core its own region of the "world" to work on, so to speak. However, all cores can see the entire "world" (i.e. the entire memory on the machine) at once, but is just restricted to working on its own region of the "world".



It is important to remember that OpenMP is **loop-level parallelization**. That means that only commands within selected DO loops will execute in parallel. GEOS-Chem Classic and HEMCO (when running within GEOS-Chem Classic, or as the HEMCO standalone) start off on a single core (known as the "main core"). Upon entering a parallel

DO loop, other cores will be invoked to share the workload within the loop. At the end of the parallel DO loop, the other cores return to standby status and the execution continues only on the "main" core.

One restriction of using OpenMP parallelization is that simulations may use only as many cores that share by the same memory. In practice, this limits GEOS-Chem Classic and HEMCO standalone simulations to using 1 node (typically less than 64 cores) of a shared computer cluster.

We should also note that GEOS-Chem High Performance (aka GCHP) uses a *different type of parallelization (MPI)*. This allows GCHP to use hundreds or thousands of cores across several nodes of a computer cluster. We encourage you to consider using GCHP for hour high-resolution simulations.

5.2 Example using OpenMP directives

Consider the following nested loop that has been parallelized with OpenMP directives:

```
!$OMP PARALLEL DO              &
!$OMP SHARED( A              ) &
!$OMP PRIVATE( I, J, B       ) &
!$OMP COLLAPSE( 2            ) &
!$OMP SCHEDULE( DYNAMIC, 4 )
DO J = 1, NY
DO I = 1, NX
   B = A(I,J)
   A(I,J) = B * 2.0
ENDDO
ENDDO
!$OMP END PARALLEL DO
```

This loop will assign different `(I,J)` pairs to different computational cores. The more cores specified, the less time it will take to do the operation.

Let us know look at the important features of this loop.

`!$OMP` `PARALLEL DO`
: This is known as a **loop sentinel**. It tells the compiler that the following DO-loop is to be executed in parallel. The **clauses** following the sentinel specify further options for the parallelization. These clauses may be spread across multiple lines by using a continuation command (`&`) at the end of the line.

`!$OMP` `SHARED( A )`
: This clause tells the compiler that all computational cores can write to `A` simultaneously. This is OK because each core will recieve a unique set of `(I,J)` pairs. Thus data corruption of the `A` array will not happen. We say that `A` is a **SHARED** variable.

Note: We recommend using the clause `!$OMP DEFAULT( SHARED )`, which will declare all varaiables as shared, unless they are explicitly placed in an `!$OMP PRIVATE` clause.

`!$OMP` `PRIVATE( I, J, B )`
: Because different cores will be handling different `(I,J)` pairs, each core needs its own private copy of variables `I` and `J`. The compiler creates these temporary copies of these variables in memory "under the hood".

  If the `I` and `J` variables were not declared `PRIVATE`, then all of the computational cores could simultaneously write to `I` and `J`. This would lead to data corruption. For the same reason, we must also place the variable `B` within the `!$OMP PRIVATE` clause.

`!$OMP` `COLLAPSE( 2 )`
: By default, OpenMP will parallelize the outer loop in a set of nested loops. To gain more efficiency, we can

vectorize the loop. "Under the hood", the compiler can convert the two nested loops over `NX` and `NY` into a single loop of size `NX * NY`, and then parallelize over the single loop. Because we wish to collapse 2 loops together, we use the `!$OMP COLLAPSE( 2 )` statement.

`!$OMP` `SCHEDULE( DYNAMIC, 4 )`
Normally, OpenMP will evenly split the domain to be parallelized (i.e. `(NX, NY)`) evenly between the cores. But if some computations take longer than others (i.e. photochemistry at the day/night boundary), this static scheduling may be inefficient.

The `SCHEDULE( DYNAMIC, 4 )` will send groups of 4 grid boxes to each core. As soon as a core finishes its work, it will immediately receive another group of 4 grid boxes. This can help to achieve better load balancing.

`!$OMP` `END PARALLEL DO`
This is a sentinel that declares the end of the parallel DO loop. It may be omitted. But we encourage you to include them, as defining both the beginning and end of a parallel loop is good programming style.

5.3 Environment variable settings for OpenMP

Please see Set environment variables for parallelization to learn which environment variables you must add to your login environment to control OpenMP parallelization.

5.4 OpenMP parallelization FAQ

Here are some frequently asked questions about parallelizing GEOS-Chem and HEMCO code with OpenMP:

5.4.1 How can I tell what should go into the !$OMP PRIVATE clause?

Here is a good rule of thumb:

> All variables that appear on the left side of an equals sign, and that have lower dimensionality than the dimensionality of the parallel loop must be placed in the `!$OMP PRIVATE` clause.

In the *example shown above*, `I`, `J`, and `B` are scalars, so their dimensionality is 0. But the parallelization occurs over two DO loops (`1..NY` and `1..NX`), so the dimensionality of the parallelization is 2. Thus `I`, `J`, and `B` must go inside the `!$OMP PRIVATE` clause.

Tip: You can also think of dimensionality as the number of indices a variable has. For example `A` has dimensionality 0, but `A(I)` has dimensionality 1, `A(I,J)` has dimensionality 2, etc.

5.4.2 Why do the !$OMP statements begin with a comment character?

This is by design. In order to invoke the parallel procesing commands, you must use a specific compiler command (such as `-openmp`, `-fopenmp`, or similar, depending on the compiler). If you omit these compiler switches, then the parallel processing directives will be considered as Fortran comments, and the associated DO-loops will be executed on a single core.

5.4.3 Do subroutine variables have to be declared PRIVATE?

Consider this subroutine:

```
SUBROUTINE mySub( X, Y, Z )

   ! Dummy variables for input
   REAL, INTENT(IN)  :: X, Y

   ! Dummy variable for output
   REAL, INTENT(OUT) :: Z

   ! Add X + Y to make Z
   Z = X + Y

END SUBROUTINE mySub
```

which is called from within a parallel loop:

```
INTEGER :: N
REAL    :: A, B, C

!$OMP PARALLEL DO           &
!$OMP DEFAULT( SHARED     ) &
!$OMP PRIVATE( N, A, B, C )
DO N = 1, nIterations

   ! Get inputs from some array
   A = Input(N,1)
   B = Input(N,2)

   ! Add A + B to make C
   CALL mySub( A, B, C )

   ! Save the output in an array
   Output(N) = C
ENDDO
!$OMP END PARALLEL DO
```

Using the *rule of thumb described above*, because `N`, `A`, `B`, and `C` are scalars (having dimensionality = 0), they must be placed in the `!$OMP PRIVATE` clause.

But note that the variables `X`, `Y`, and `Z` do not need to be placed within a `!$OMP PRIVATE` clause within subroutine `mySub`. This is because each core calls `mySub` in a separate thread of execution, and will create its own private copy of `X`, `Y`, and `Z` in memory.

5.4.4 What does the THREADPRIVATE statement do?

Let's modify the *above example* slightly. Let's now suppose that subroutine `mySub` from the prior example is now part of a Fortran-90 module, which looks like this:

```
MODULE myModule

  ! Module variable:
  REAL, PUBLIC :: Z
```

(continues on next page)

(continued from previous page)

```
CONTAINS

  SUBROUTINE mySub( X, Y )

   ! Dummy variables for input
   REAL, INTENT(IN)  :: X, Y

   ! Add X + Y to make Z
   ! NOTE that Z is now a global variable
   Z  = X + Y

  END SUBROUTINE mySub

END MODULE myModule
```

Note that `Z` is now a global scalar variable with dimensionality = 0. Let's now use the same parallel loop (dimensionality = 1) as before:

```
! Get the Z variable from myModule
USE myModule, ONLY : Z

INTEGER :: N
REAL    :: A, B, C

!$OMP PARALLEL DO           &
!$OMP DEFAULT( SHARED     ) &
!$OMP PRIVATE( N, A, B, C )
DO N = 1, nIterations

   ! Get inputs from some array
   A = Input(N,1)
   B = Input(N,2)

   ! Add A + B to make C
   CALL mySub( A, B )

   ! Save the output in an array
   Output(N) = Z

ENDDO
!$OMP END PARALLEL DO
```

Because `Z` is now a global variable with lower dimensionality than the loop, we must try to place it within an `!$OMP PRIVATE` clause. However, `Z` is defined in a different program unit than where the parallel loop occurs, so we cannot place it in an `!$OMP PRIVATE` clause for the loop.

In this case we must place `Z` into an `!$OMP THREADPRIVATE` clause within the module where it is declared, as shown below:

```
MODULE myModule

  ! Module variable:
  ! This is global and acts as if it were in a F77-style common block
  REAL, PUBLIC :: Z
  !$OMP THREADPRIVATE( Z )

  ... etc ...
```

This tells the computer to create a separate private copy of `Z` in memory for each core.

Important: When you place a variable into an `!$OMP PRIVATE` or `!$OMP THREADPRIVATE` clause, this means that the variable will have no meaning outside of the parallel loop where it is used. So you should not rely on using the value of `PRIVATE` or `THREADPRIVATE` variables elsewhere in your code.

Most of the time you won't have to use the `!$OMP THREADPRIVATE` statement. You may need to use it if you are trying to parallelize code that came from someone else.

5.4.5 Can I use pointers within an OpenMP parallel loop?

You may use pointer-based variables (including derived-type objects) within an OpenMP parallel loop. But you must make sure that you point to the target within the parallel loop section AND that you also nullify the pointer within the parallel loop section. For example:

INCORRECT:

```
! Declare variables
REAL, TARGET  :: myArray(NX,NY)
REAL, POINTER :: myPtr  (:    )

! Declare an OpenMP parallel loop
!$OMP PARALLEL DO                    ) &
!$OMP DEFAULT( SHARED                ) &
!$OMP PRIVATE( I, J, myPtr, ...etc... )
DO J = 1, NY
DO I = 1, NX

   ! Point to a variable.
   !This must be done in the parallel loop section.
   myPtr => myArray(:,J)

   . . . do other stuff . . .

ENDDO
!$OMP END PARALLEL DO

! Nullify the pointer.
! NOTE: This is incorrect because we nullify the pointer outside of the loop.
myPtr => NULL()
```

CORRECT:

```
! Declare variables
REAL, TARGET  :: myArray(NX,NY)
REAL, POINTER :: myPtr  (:    )

! Declare an OpenMP parallel loop
!$OMP PARALLEL DO                    ) &
!$OMP DEFAULT( SHARED                ) &
!$OMP PRIVATE( I, J, myPtr, ...etc... )
DO J = 1, NY
DO I = 1, NX

   ! Point to a variable.
```

(continues on next page)

(continued from previous page)

```
   !This must be done in the parallel loop section.
   myPtr => myArray(:,J)

   . . . do other stuff . . .

   ! Nullify the pointer within the parallel loop
   myPtr => NULL()

ENDDO
!$OMP END PARALLEL DO
```

In other words, pointers used in OpenMP parallel loops only have meaning within the parallel loop.

5.4.6 How many cores may I use for GEOS-Chem or HEMCO?

You can use as many computational cores as there are on a single node of your cluster. With OpenMP parallelization, the restriction is that all of the cores have to see all the memory on the machine (or node of a larger machine). So if you have 32 cores on a single node, you can use them. We have shown that run times will continue to decrease (albeit asymptotically) when you increase the number of cores.

5.4.7 Why is GEOS-Chem is not using all the cores I requested?

The number of threads for an OpenMP simulation is determined by the environment variable OMP_NUM_THREADS. You must define `OMP_NUM_THREADS` in your environment file to specify the desired number of computational cores for your simulation. For the `bash` shell, use4 this command to request 8 cores:

```
export OMP_NUM_THREADS=8
```

5.5 MPI parallelization

The *OpenMP parallelization* used by GEOS-Chem Classic and HEMCO standalone is an example of **shared memory parallelization** (also known as **serial parallelization**). As we have seen, we are restricted to using a single node of a computer cluster. This is because all of the cores need to talk with all of the memory on the node.

On the other hand, MPI (Message Passing Interface) parallelzation is an example of **distributed parallelization**. An MPI library installation is required for passing memory from one physical system to another (i.e. across nodes).

GEOS-Chem High Performance (GCHP) uses Earth System Model Framework (ESMF) and MAPL libraries to implement MPI parallelization. For detailed information, please see gchp.readthedocs.io.

CHAPTER

SIX

DEBUG GEOS-CHEM AND HEMCO ERRORS

If your **GEOS-Chem** or **HEMCO** simulation dies unexpectedly with an error or takes much longer to execute than it should, the most important thing is to try to isolate the source of the error or bottleneck right away. Below are some debugging tips that you can use.

6.1 Check if a solution has been posted to Github

We have migrated support requests from the GEOS-Chem wiki to **Github issues**. A quick search of Github issues (both open and closed) might reveal the answer to your question or provide a solution to your problem.

You should also feel free to open a new issue at one of these Github links:

- GEOS-Chem Classic new issues page
- GCHP new issues page
- HEMCO new issues page

If you are new to Github, we recommend viewing our Github tutorial videos at our GEOS-Chem Youtube site.

6.2 Check if your computational environment is configured properly

Many **GEOS-Chem** and **HEMCO** errors occur due to improper configuration settings (i.e. missing libraries, incorrectly-specified environment variables, etc.) in your computational environment. Take a moment and refer back to these manual pages (on ReadTheDocs) for information on configuring your environment:

- GEOS-Chem Classic manual
- GCHP manual
- HEMCO manual

6.3 Check any code modifications that you have added

If you have made modifications to a "fresh out-of-the-box" **GEOS-Chem** or **HEMCO** version, look over your code edits to search for sources of potential error.

You can also use Git to revert to the last stable version, which is always in the **main** branch.

6.4 Check if your runs exceeded time or memory limits

If you are running **GEOS-Chem** or **HEMCO** on a shared computer system, you will probably have to use a **job scheduler** (such as **SLURM**) to submit your jobs to a computational queue. You should be aware of the run time and memory limits for each of the queues on your system.

If your job uses more memory or run time than the computational queue allows, it can be cancelled by the scheduler. You will usually get an error message printed out to the stderr stream, and maybe also an email stating that the run was terminated. Be sure to check all of the log files created by your jobs for such error messages.

To solve this issue, try submitting your **GEOS-Chem** or **HEMCO** simulations to a queue with larger run-time and memory limits. You can also try splitting up your long simulations into several smaller stages (e.g. monthly) that take less time to run to completion.

6.5 Send debug printout to the log files

If your **GEOS-Chem** simulation stopped with an error, but you cannot tell where, turn on the the `debug_printout` option. This is found in the **Simulation Settings** section of `geoschem_config.yml`:

```
#============================================================================
# Simulation settings
#============================================================================
simulation:
  name: fullchem
  start_date: [20190701, 000000]
  end_date: [20190801, 000000]
  root_data_dir: /path/to/ExtData
  met_field: MERRA2
  species_database_file: ./species_database.yml
  debug_printout: false  # <---- set this to true
  use_gcclassic_timers: false
```

This will send additional output to the **GEOS-Chem** log file, which may help you to determine where the simulation stopped.

If your **HEMCO** simulation stopped with an error, turn on debug printout by editing the `Verbose` and `Warnings` settings at the top of the `HEMCO_Config.rc` configuration file:

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

ROOT:                         /path/to/ExtData/HEMCO
METDIR:                       MERRA2
GCAP2SCENARIO:                none
GCAP2VERTRES:                 none
Logfile:                      HEMCO.log
DiagnFile:                    HEMCO_Diagn.rc
DiagnPrefix:                  ./OutputDir/HEMCO_diagnostics
DiagnFreq:                    Monthly
Wildcard:                     *
Separator:                    /
Unit tolerance:               1
Negative values:              0
Only unitless scale factors: false
```

(continues on next page)

(continued from previous page)

```
Verbose:                    0       # <---- set this to 3
Warnings:                   1       # <---- set this to 3
```

Both `Verbose` and `Warnings` settings can have values from 0 to 3. The higher the number, the more information will be printed out to the `HEMCO.log` file. A value of 0 disables debug printout.

Having this extra debug printout in your log file output may provide insight as to where your simulation is halting.

6.6 Look at the traceback output

An **error traceback** will be printed out whenever a **GEOS-Chem** or **HEMCO** simulation halts with an error. This is a list of routines that were called when the error occurred.

An sample error traceback is shown here:

```
forrtl: severe (174): SIGSEGV, segmentation fault occurred

Image              PC                Routine            Line        Source
gcclassic          0000000000C82023  Unknown               Unknown  Unknown
libpthread-2.17.s  00002AACE8015630  Unknown               Unknown  Unknown
gcclassic          000000000095935E  error_mod_mp_erro         437  error_mod.F90
gcclassic          000000000040ABB7  MAIN__                    422  main.F90
gcclassic          0000000000406B92  Unknown               Unknown  Unknown
libc-2.17.so       00002AACE8244555  __libc_start_main     Unknown  Unknown
gcclassic          0000000000406AA9  Unknown               Unknown  Unknown
```

The top line with a valid routine name and line number printed is the routine that exited with an error (`error_mod.F90`, line 437). You might also have to look at the other listed files as well to get some more information about the error (e.g. `main.F90`, line 422).

6.7 Identify whether the error happens consistently

If your **GEOS-Chem** or **HEMCO** error always happens at the same model date and time, this could indicate corrupted meteorology or emissions input data files. In this case, you may be able to fix the issue simply by re-downloading the files to your disk space.

If the error happened only once, it could be caused by a network problem or other such transient condition.

6.8 Isolate the error to a particular operation

If you are not sure where a **GEOS-Chem** error is occurring, turn off operations (such as transport, chemistry, dry deposition, etc.) one at a time in the `geoschem_config.yml` configuration file, and rerun your simulation.

Similarly, if you are debugging a **HEMCO** error, turn off different emissions inventories and extensions one at a time in the `HEMCO_Config.rc` file, and rerun your simulation.

Repeating this process should eventually lead you to the source of the error.

6.9 Compile with debugging options

You can compile **GEOS-Chem** or **HEMCO** in debug mode. This will activate several additional error run-time error checks (such as looking for assignments that go outside of array bounds or floating point math errors) that can give you more insight as to where your simulation is dying.

Configure your code for debug mode with the **-DCMAKE_RELEASE_TYPE=Debug** option. From your run directory, type these commands:

```
cd build
cmake ../CodeDir -DCMAKE_RELEASE_TYPE=Debug -DRUNDIR=..
make -j
make -j install
cd ..
```

Attention: Compiling in debug mode will add a significant amount of computational overhead to your simulation. Therefore, we recommend to activate these additional error checks only in short simulations and not in long production runs.

6.10 Use a debugger

You can save yourself a lot of time and hassle by using a debugger such as **gdb** (the GNU debugger). With a debugger you can:

- Examine data when a program stops
- Navigate the stack when a program stops
- Set break points

To run **GEOS-Chem** or **HEMCO** in the **gdb** debugger, you should first *compile in debug mode*. This will turn on the `-g` compiler flag (which tells the compiler to generate symbolic information for debugging) and the `-O0` compiler flag (which shuts off all optimizations. Once the executable has been created, type one of the following commands, which will start **gdb**:

```
$ gdb gcclassic    # for GEOS-Chem Classic
$ gdb gchp         # for GCHP
$ gdb hemco        # for HEMCO standalone
```

At the **gdb** prompt, type one of these commands:

```
(gdb) run                      # for GEOS-Chem Classic or GCHP
(gdb) run HEMCO_sa_Config.rc   # for HEMCO standalone
```

With **gdb**, you can also go directly to the point of the error without having to re-run GEOS-Chem or HEMCO. When your GEOS-Chem or HEMCO simulation dies, it will create a **corefile** such as `core.12345`. The `12345` refers to the process ID assigned to your executable by the operating system; this number is different for each running process on your system.

Typing one of these commands:

```
$ gdb gcclassic core.12345          # for GEOS-Chem Classic
$ gdb gchp core.12345               # for GCHP
$ gdb hemco_standalone core.12345   # for HEMCO standalone
```

will open **gdb** and bring you immediately to the point of the error. If you then type at the `(gdb)` prompt:

```
(gdb) where
```

You will get a *traceback* listing.

To exit **gdb**, type `quit`.

6.11 Print it out if you are in doubt!

Add `print*,` statements to write values of variables in the area of the code where you suspect the error is occurring. Also add the `call flush(6)` statement to flush the output to the screen and/or log file immediately after printing. Maybe you will see something wrong in the output.

You can often detect numerical errors by adding debugging print statements into your source code:

1. Use `MINVAL` and `MAXVAL` functions to get the minimum and maximum values of an array:

   ```
   PRINT*, '### Min, Max: ', MINVAL( ARRAY ), MAXVAL( ARRAY )
   CALL FLUSH( 6 )
   ```

2. Use the `SUM` function to check the sum of an array:

   ```
   PRINT*, '### Sum of X : ', SUM( ARRAY )
   CALL FLUSH( 6 )
   ```

6.12 Use the brute-force method when all else fails

If the bug is difficult to locate, then comment out a large section of code and run your **GEOS-Chem** or **HEMCO** simulation again. If the error does not occur, then uncomment some more code and run again. Repeat the process until you find the location of the error. The brute force method may be tedious, but it will usually lead you to the source of the problem.

6.13 Identify poorly-performing code with a profiler

If you think your **GEOS-Chem** or **HEMCO** simulation is taking too long to run, consider using profiling tools to generate a list of the time that is spent in each routine. This can help you identify badly written and/or poorly-parallelized code. For more information, please see our Profiling GEOS-Chem wiki page.

CHAPTER

SEVEN

MANAGE A DATA ARCHIVE WITH BASHDATACATALOG

If you need to download a large amount of input data for **GEOS-Chem** or **HEMCO** (e.g. in support of a large user group at your institution) you may find **bashdatacatalog** helpful.

7.1 What is bashdatacatalog?

The **bashdatacatalog** is a command-line tool (written by Liam Bindle) that facilitates synchronizing local data collections with a remote data source. With the **bashdatacatalog**, you can run queries on your local data collections to answer questions like "What files am I missing?" or "What files aren't bitwise identical to remote data?". Queries can include a date range, in which case collections with temporal assets are filtered-out accordingly. The **bashdatacatalog** can format the results of queries as: a URL download list, a Globus transfer list, an rsync transfer list, or simply a file list.

The **bashdatacatalog** was written to facilitate downloading input data for users of the GEOS-Chem atmospheric chemistry model. The canonical GEOS-Chem input data repository has >1 M files and >100 TB of data, and the input data required for a simulation depends on the model version and simulation parameters such as start and end date.

7.2 Usage instructions

For detailed instructions on using **bashdatacatalog**, please see the bashdatacatalog wiki on Github.

Also see our input-data-catalogs Github repository for comma-separated input lists of GEOS-Chem data, separated by model version.

CHAPTER

EIGHT

WORK WITH NETCDF FILES

On this page we provide some useful information about working with data files in netCDF format.

8.1 Useful tools

There are many free and open-source software packages readily available for visualizing and manipulating netCDF files.

cdo
: **Climate Data Operators**: Highly-optimized command-line tools for manipulating and analyzing netCDF files. Contains features that are especially useful for Earth Science applications.

  See: https://code.zmaw.de/projects/cdo

GCPy
: **GEOS-Chem Python toolkit**: Python package for visualizing and analyzing GEOS-Chem output. Used for creating the GEOS-Chem benchmark plots. Also contains some useful routines for creating single-panel plots and multi-panel difference plots.

  See: https://gcpy.readthedocs.io

ncdump
: Generates a text representation of netCDF data and can be used to quickly view the variables contained in a netCDF file. **ncdump** is installed to the `bin/` folder of your netCDF library distribution.

  See: https://www.unidata.ucar.edu/software/netcdf/workshops/2011/utilities/Ncdump.html

nco
: **netCDF operators**: Highly-optimized command-line tools for manipulating and analyzing netCDF files.

  See: http://nco.sourceforge.net

ncview
: Visualization package for netCDF files. **Ncview** has limited features, but is great for a quick look at the contents of netCDF files.

  See: http://meteora.ucsd.edu/~pierce/ncview_home_page.html

netcdf-scripts
: Our repository of useful netCDF utility scripts for GEOS-Chem.

  See: https://github.com/geoschem/netcdf-scripts

Panoply
: Java-based data viewer for netCDF files. This package offers an alternative to ncview. From our experience, Panoply works nicely when installed on the desktop, but is slow to respond in the Linux environment.

See: https://www.giss.nasa.gov/tools/panoply/

xarray
Python package that lets you read the contents of a netCDF file into a data structure. The data can then be further manipulated or converted to numpy or dask arrays for further procesing.

See: https://xarray.readthedocs.io

Some of the tools listed above, such as **ncdump** and **ncview** may come pre-installed on your system. Others may need to be installed or loaded (e.g. via the **module load** command). Check with your system administrator or IT staff to see what is available on your system.

8.2 Examine the contents of a netCDF file

An easy way to examine the contents of a netCDF file is to use **ncdump** as follows:

```
$ ncdump -ct GEOSChem.SpeciesConc.20190701_0000z.nc4
```

You will see output similar to this:

```
netcdf GEOSChem.SpeciesConc.20190701_0000z {
dimensions:
     time = UNLIMITED ; // (1 currently)
     lev = 72 ;
     ilev = 73 ;
     lat = 46 ;
     lon = 72 ;
     nb = 2 ;
variables:
     double time(time) ;
             time:long_name = "Time" ;
             time:units = "minutes since 2019-07-01 00:00:00" ;
             time:calendar = "gregorian" ;
             time:axis = "T" ;
     double lev(lev) ;
             lev:long_name = "hybrid level at midpoints ((A/P0)+B)" ;
             lev:units = "level" ;
             lev:axis = "Z" ;
             lev:positive = "up" ;
             lev:standard_name = "atmosphere_hybrid_sigma_pressure_coordinate" ;
             lev:formula_terms = "a: hyam b: hybm p0: P0 ps: PS" ;
     double ilev(ilev) ;
             ilev:long_name = "hybrid level at interfaces ((A/P0)+B)" ;
             ilev:units = "level" ;
             ilev:positive = "up" ;
             ilev:standard_name = "atmosphere_hybrid_sigma_pressure_coordinate" ;
             ilev:formula_terms = "a: hyai b: hybi p0: P0 ps: PS" ;
     double lat_bnds(lat, nb) ;
             lat_bnds:long_name = "Latitude bounds (CF-compliant)" ;
             lat_bnds:units = "degrees_north" ;
     double lat(lat) ;
             lat:long_name = "Latitude" ;
             lat:units = "degrees_north" ;
             lat:axis = "Y" ;
             lat:bounds = "lat_bnds" ;
     double lon_bnds(lon, nb) ;
             lon_bnds:long_name = "Longitude bounds (CF-compliant)" ;
```

(continues on next page)

(continued from previous page)

```
            lon_bnds:units = "degrees_east" ;
    double lon(lon) ;
            lon:long_name = "Longitude" ;
            lon:units = "degrees_east" ;
            lon:axis = "X" ;
            lon:bounds = "lon_bnds" ;
    double hyam(lev) ;
            hyam:long_name = "hybrid A coefficient at layer midpoints" ;
            hyam:units = "hPa" ;
    double hybm(lev) ;
            hybm:long_name = "hybrid B coefficient at layer midpoints" ;
            hybm:units = "1" ;
    double hyai(ilev) ;
            hyai:long_name = "hybrid A coefficient at layer interfaces" ;
            hyai:units = "hPa" ;
    double hybi(ilev) ;
            hybi:long_name = "hybrid B coefficient at layer interfaces" ;
            hybi:units = "1" ;
    double P0 ;
            P0:long_name = "reference pressure" ;
            P0:units = "hPa" ;
    float AREA(lat, lon) ;
            AREA:long_name = "Surface area" ;
            AREA:units = "m2" ;
    float SpeciesConc_RCOOH(time, lev, lat, lon) ;
            SpeciesConc_RCOOH:long_name = "Dry mixing ratio of species RCOOH" ;
            SpeciesConc_RCOOH:units = "mol mol-1 dry" ;
            SpeciesConc_RCOOH:averaging_method = "time-averaged" ;
    float SpeciesConc_O2(time, lev, lat, lon) ;
            SpeciesConc_O2:long_name = "Dry mixing ratio of species O2" ;
            SpeciesConc_O2:units = "mol mol-1 dry" ;
            SpeciesConc_O2:averaging_method = "time-averaged" ;
    float SpeciesConc_N2(time, lev, lat, lon) ;
            SpeciesConc_N2:long_name = "Dry mixing ratio of species N2" ;
            SpeciesConc_N2:units = "mol mol-1 dry" ;
            SpeciesConc_N2:averaging_method = "time-averaged" ;
    float SpeciesConc_H2(time, lev, lat, lon) ;
            SpeciesConc_H2:long_name = "Dry mixing ratio of species H2" ;
            SpeciesConc_H2:units = "mol mol-1 dry" ;
            SpeciesConc_H2:averaging_method = "time-averaged" ;
    float SpeciesConc_O(time, lev, lat, lon) ;
            SpeciesConc_O:long_name = "Dry mixing ratio of species O" ;
            SpeciesConc_O:units = "mol mol-1 dry" ;

            ... etc ...

// global attributes:
            :title = "GEOS-Chem diagnostic collection: SpeciesConc" ;
            :history = "" ;
            :format = "not found" ;
            :conventions = "COARDS" ;
            :ProdDateTime = "" ;
            :reference = "www.geos-chem.org; wiki.geos-chem.org" ;
            :contact = "GEOS-Chem Support Team (geos-chem-support@g.harvard.edu)" ;
            :simulation_start_date_and_time = "2019-07-01 00:00:00z" ;
            :simulation_end_date_and_time = "2019-07-01 01:00:00z" ;
data:
```

(continues on next page)

(continued from previous page)

```
 time = "2019-07-01 00:30" ;

 lev = 0.99250002413, 0.97749990013, 0.962499776, 0.947499955, 0.93250006,
    0.91749991, 0.90249991, 0.88749996, 0.87249996, 0.85750006, 0.842500125,
    0.82750016, 0.8100002, 0.78750002, 0.762499965, 0.737500105, 0.7125001,
    0.6875001, 0.65625015, 0.6187502, 0.58125015, 0.5437501, 0.5062501,
    0.4687501, 0.4312501, 0.3937501, 0.3562501, 0.31279158, 0.26647905,
    0.2265135325, 0.192541016587707, 0.163661504087706, 0.139115, 0.11825,
    0.10051436, 0.085439015, 0.07255786, 0.06149566, 0.05201591, 0.04390966,
    0.03699271, 0.03108891, 0.02604911, 0.021761005, 0.01812435, 0.01505025,
    0.01246015, 0.010284921, 0.008456392, 0.0069183215, 0.005631801,
    0.004561686, 0.003676501, 0.002948321, 0.0023525905, 0.00186788,
    0.00147565, 0.001159975, 0.00090728705, 0.0007059566, 0.0005462926,
    0.0004204236, 0.0003217836, 0.00024493755, 0.000185422, 0.000139599,
    0.00010452401, 7.7672515e-05, 5.679251e-05, 4.0142505e-05, 2.635e-05,
    1.5e-05 ;

 ilev = 1, 0.98500004826, 0.969999752, 0.9549998, 0.94000011, 0.92500001,
    0.90999981, 0.89500001, 0.87999991, 0.86500001, 0.85000011, 0.83500014,
    0.82000018, 0.80000022, 0.77499982, 0.75000011, 0.7250001, 0.7000001,
    0.6750001, 0.6375002, 0.6000002, 0.5625001, 0.5250001, 0.4875001,
    0.4500001, 0.4125001, 0.3750001, 0.3375001, 0.28808306, 0.24487504,
    0.208152025, 0.176930008175413, 0.150393, 0.127837, 0.108663, 0.09236572,
    0.07851231, 0.06660341, 0.05638791, 0.04764391, 0.04017541, 0.03381001,
    0.02836781, 0.02373041, 0.0197916, 0.0164571, 0.0136434, 0.0112769,
    0.009292942, 0.007619842, 0.006216801, 0.005046801, 0.004076571,
    0.003276431, 0.002620211, 0.00208497, 0.00165079, 0.00130051, 0.00101944,
    0.0007951341, 0.0006167791, 0.0004758061, 0.0003650411, 0.0002785261,
    0.000211349, 0.000159495, 0.000119703, 8.934502e-05, 6.600001e-05,
    4.758501e-05, 3.27e-05, 2e-05, 1e-05 ;

 lat = -89, -86, -82, -78, -74, -70, -66, -62, -58, -54, -50, -46, -42, -38,
    -34, -30, -26, -22, -18, -14, -10, -6, -2, 2, 6, 10, 14, 18, 22, 26, 30,
    34, 38, 42, 46, 50, 54, 58, 62, 66, 70, 74, 78, 82, 86, 89 ;

 lon = -180, -175, -170, -165, -160, -155, -150, -145, -140, -135, -130,
    -125, -120, -115, -110, -105, -100, -95, -90, -85, -80, -75, -70, -65,
    -60, -55, -50, -45, -40, -35, -30, -25, -20, -15, -10, -5, 0, 5, 10, 15,
    20, 25, 30, 35, 40, 45, 50, 55, 60, 65, 70, 75, 80, 85, 90, 95, 100, 105,
    110, 115, 120, 125, 130, 135, 140, 145, 150, 155, 160, 165, 170, 175 ;
}
```

You can also use **ncdump** to display the data values for a given variable in the netCDF file. This command will display the values in the `SpeciesRst_O3` variable to the screen:

```
$ ncdump -v SpeciesConc_O3 GEOSChem.SpeciesConc.20190701_0000z.nc4 | less
```

Or you can redirect the output to a file:

```
$ ncdump -v SpeciesConc_O3 GEOSChem.SpeciesConc.20190701_0000z.nc4 > log
```

8.3 Read the contents of a netCDF file

8.3.1 Read data with Python

The easiest way to read a netCDF file is to use the xarray Python package.

```
#!/usr/bin/env python

# Imports
import numpy as np
import xarray as xr

# Read a restart file into an xarray Dataset object
ds = xr.open_dataset("GEOSChem.SpeciesConc.20190701_0000z.nc4")

# Print the contents of the DataSet
print(ds)

# Print units of data
print(f"\nUnits of SpeciesRst_O3: {ds['SpeciesConc_O3'].units}")

# Print the sum, max, and min of the data
# NOTE .values returns a numpy ndarray so that we can use
# other numpy functions like np.sum() on the data
print(f"Sum of SpeciesRst_O3: {np.sum(ds['SpeciesConc_O3'].values)}")
print(f"Max of SpeciesRst_O3: {np.max(ds['SpeciesConc_O3'].values)}")
print(f"Min of SpeciesRst_O3: {np.min(ds['SpeciesConc_O3'].values)}")
```

This above script will print the following output:

```
<xarray.Dataset>
Dimensions:               (ilev: 73, lat: 46, lev: 72, lon: 72, nb: 2, time: 1)
Coordinates:
  * time                  (time) datetime64[ns] 2019-07-01T00:30:00
  * lev                   (lev) float64 0.9925 0.9775 ... 2.635e-05 1.5e-05
  * ilev                  (ilev) float64 1.0 0.985 0.97 ... 3.27e-05 2e-05 1e-05
  * lat                   (lat) float64 -89.0 -86.0 -82.0 ... 82.0 86.0 89.0
  * lon                   (lon) float64 -180.0 -175.0 -170.0 ... 170.0 175.0
Dimensions without coordinates: nb
Data variables: (12/315)
    lat_bnds              (lat, nb) float64 ...
    lon_bnds              (lon, nb) float64 ...
    hyam                  (lev) float64 ...
    hybm                  (lev) float64 ...
    hyai                  (ilev) float64 ...
    hybi                  (ilev) float64 ...
    ...                    ...
    SpeciesConc_AONITA    (time, lev, lat, lon) float32 ...
    SpeciesConc_ALK4      (time, lev, lat, lon) float32 ...
    SpeciesConc_ALD2      (time, lev, lat, lon) float32 ...
    SpeciesConc_AERI      (time, lev, lat, lon) float32 ...
    SpeciesConc_ACTA      (time, lev, lat, lon) float32 ...
    SpeciesConc_ACET      (time, lev, lat, lon) float32 ...
Attributes:
    title:                          GEOS-Chem diagnostic collection: Species...
    history:
    format:                         not found
```

(continues on next page)

(continued from previous page)

```
    conventions:                    COARDS
    ProdDateTime:
    reference:                      www.geos-chem.org; wiki.geos-chem.org
    contact:                        GEOS-Chem Support Team (geos-chem-suppor...
    simulation_start_date_and_time: 2019-07-01 00:00:00z
    simulation_end_date_and_time:   2019-07-01 01:00:00z

Units of SpeciesRst_O3: mol mol-1 dry
Sum of SpeciesRst_O3: 0.4052325189113617
Max of SpeciesRst_O3: 1.012129541777 54e-05
Min of SpeciesRst_O3: 3.758645839013752e-09
```

8.3.2 Read data from multiple files in Python

The xarray package will also let you read data from multiple files into a single Dataset object. This is done with the open_mfdataset (open multi-file-dataset) function as shown below:

```
#!/usr/bin/env python

# Imports
import xarray as xr

# Create a list of files to open
filelist = [
    'GEOSChem.SpeciesConc.20160101_0000z.nc4',
    'GEOSChem.SpeciesConc_20160201_0000z.nc4',
    ...
]

# Read a restart file into an xarray Dataset object
ds = xr.open_mfdataset(filelist)
```

8.4 Determining if a netCDF file is COARDS-compliant

All netCDF files used as input to GEOS-Chem and/or HEMCO must adhere to the *COARDS netCDF conventions*. You can use the isCoards script (from our netcdf-scripts repository at GitHub) to determine if a netCDF file adheres to the COARDS conventions.

Run the `isCoards` script at the command line on any netCDF file, and you will receive a report as to which elements of the file do not comply with the COARDS conventions.

```
$ isCoards myfile.nc

================================================================================
Filename: myfile.nc
================================================================================

The following items adhere to the COARDS standard:
--------------------------------------------------------------------------------
-> Dimension "time" adheres to standard usage
-> Dimension "lev" adheres to standard usage
-> Dimension "lat" adheres to standard usage
```

(continues on next page)

(continued from previous page)

```
-> Dimension "lon" adheres to standard usage
-> time(time)
-> time is monotonically increasing
-> time:axis = "T"
-> time:calendar = "gregorian"
-> time:long_name = "Time"
-> time:units = "hours since 1985-1-1 00:00:0.0"
-> lev(lev)
-> lev is monotonically decreasing
-> lev:axis = "Z"
-> lev:positive = "up"
-> lev:long_name = "GEOS-Chem levels"
-> lev:units = "sigma_level"
-> lat(lat)
-> lat is monotonically increasing
-> lat:axis = "Y"
-> lat:long_name = "Latitude"
-> lat:units = "degrees_north"
-> lon(lon)
-> lon is monotonically increasing
-> lon:axis = "X"
-> lon:long_name = "Longitude"
-> lon:units = "degrees_east"
-> OH(time,lev,lat,lon)
-> OH:long_name = "Chemically produced OH"
-> OH:units = "kg/m3"
-> OH:long_name = 1.e+30f
-> OH:missing_value = 1.e+30f
-> conventions: "COARDS"
-> history: "Mon Apr  3 08:26:19 2017"
-> title: "COARDS/netCDF file created by BPCH2COARDS (GAMAP v2-17+)"
-> format: "NetCDF-3"

The following items DO NOT ADHERE to the COARDS standard:
------------------------------------------------------------------------------
-> time[0] != 0 (problem for GCHP)

The following optional items are RECOMMENDED:
------------------------------------------------------------------------------
-> Consider adding the "references" global attribute
```

8.5 Edit variables and attributes

As discussed *in the preceding section*, you may find that you need to edit your netCDF files for COARDS-compliance. Below are several useful commands for editing netCDF files. Many of these commands utilize the `nco` and `cdo` utilities.

1. Display the header and coordinate variables of a netCDF file, with the time variable displayed in human-readable format. Also show status of file *compression and/or chunking*.

   ```
   $ ncdump -cts file.nc
   ```

2. *Compress a netCDF file*. This can considerably reduce the file size!

```
# No deflation
$ nccopy -d0 myfile.nc tmp.nc
$ mv tmp.nc myfile.nc

# Minimum deflation (good for most applications)
$ nccopy -d1 myfile.nc tmp.nc
$ mv tmp.nc myfile.nc

# Medium deflation
$ nccopy -d5 myfile.nc tmp.nc
$ mv tmp.nc myfile.nc

# Maximum deflation
$ nccopy -d9 myfile.nc tmp.nc
$ mv tmp.nc myfile.nc
```

3. Change variable name from `SpeciesConc_NO` to `NO`:

```
$ ncrename -v SpeciesConc_NO,NO myfile.nc
```

4. Set all missing values to zero:

```
$ cdo setemisstoc,0 myfile.nc tmp.nc
$ mv tmp.nc myfile.nc
```

5. Add/change the long-name attribute of the vertical coordinates (lev) to "GEOS-Chem levels". This will ensure that HEMCO recognizes the vertical levels of the input file as GEOS-Chem model levels.

```
$ ncatted -a long_name,lev,o,c,"GEOS-Chem levels" myfile.nc
```

6. Add/change the axis and positive attributes to the vertical coordinate (lev):

```
$ ncatted -a axis,lev,o,c,"Z" myfile.nc
$ ncatted -a positive,lev,o,c,"up" myfile.nc
```

7. Add/change the `units` attribute of the latitude (lat) coordinate to `degrees_north`:

```
$ ncatted -a units,lat,o,c,"degrees_north" myfile.nc
```

8. Convert the `units` attribute of the CHLA variable from `mg/m3` to `kg/m3`

```
$ ncap2 -v -s "CHLA=CHLA/1000000.0f" myfile.nc tmp.nc
$ ncatted -a units,CHLA,o,c,"kg/m3" tmp.nc
$ mv tmp.nc myfile.nc
```

9. Add/change the `references`, `title`, and `history` global attributes

```
$ ncatted -a references,global,o,c,"www.geos-chem.org; wiki.geos-chem.org" myfile.
↪nc
$ ncatted -a history,global,o,c,"Tue Mar  3 12:18:38 EST 2015" myfile.nc
$ ncatted -a title,global,o,c,"XYZ data from ABC source" myfile.nc
```

10. Remove the `references` global attribute:

```
$ ncatted -a references,global,d,, myfile.nc
```

11. Add a `time` dimension to a file that does not have one:

```
$ ncap2 -h -s 'defdim("time",1);time[time]=0.0;time@long_name="time";
↪time@calendar="standard";time@units="days since 2007-01-01 00:00:00"' -O myfile.
↪nc tmp.nc
$ mv tmp.nc myfile.nc
```

12. Add a `time` dimension to a variable:

```
# Assume myVar has lat and lon dimensions to start with
$ ncap2 -h -s 'myVar[$time,$lat,$lon]=myVar;' myfile.nc tmp.nc
$ mv tmp.nc myfile.nc
```

13. Make the `time` dimension unlimited:

```
$ ncks --mk_rec_dmn time myfile.nc tmp.nc
$ mv tmp.nc myfile.nc
```

14. Change the file reference date and time (i.e. `time:units`) from 1 Jan 1985 to 1 Jan 2000:

```
$ cdo setreftime,2000-01-01,00:00:00 myfile.nc tmp.nc
$ mv tmp.nc myfile.nc
```

15. Shift all time values ahead or back by 1 hour in a file:

```
# Shift ahead 1 hour
$ cdo shifttime,1hour myfile.nc tmp.nc
$ mv tmp.nc myfile.nc

# Shift back 1 hour
$ cdo shiftime,-1hour myfile.nc tmp.nc
$ mv tmp.nc myfile.nc
```

16. Set the date of all variables in the file. (Useful for files that have only one time point.)

```
$ cdo setdate,2019-07-02 myfile.nc tmp.nc
$ mv tmp.nc myfile.nc
```

Tip: The following **cdo** commands are similar to **cdo setdate**, but allow you to manipulate other time variables:

```
$ cdo settime,03:00:00 ...   # Sets time to 03:00 UTC
$ cdo setday,26, ...         # Sets day of month to 26
$ cdo setmon,10, ...         # Sets month to 10 (October)
$ cdo setyear,1992, ...      # Sets year to 1992
```

See the cdo user manual for more information.

8.6 Concatenate netCDF files

There are a couple of ways to concatenate multiple netCDF files into a single netCDF file, as shown in the sections below.

8.6.1 Concatenate with the netCDF operators

You can use the **ncrcat** utility (from *nco*) to concatenate the individual netCDF files into a single netCDF file.

Let's assume we want to combine 12 monthy data files (e.g. `month_01.nc`, `month_02.nc`, .. `month_12.nc` into a single file called `annual_data.nc`.

First, make sure that each of the `month_*nc` files has an unlimited `time` dimension. Type this at the command line:

```
$ ncdump -ct month_01.nc | grep "time"
```

Then you should see this as the first line in the output:

```
time = UNLIMITED ; // (1 currently)
```

This indicates that the time dimension is unlimited. If on the other hand you see this output:

```
time = 1 ;
```

Then it means that the time dimension is fixed. If this is the case, you will have to use the **ncks** command to make the time dimension unlimited, as follows:

```
$ ncks --mk_rec_dmn time month_01.nc tmp.nc
$ mv tmp.nc month_01.nc
... etc for the other files ...
```

Then use **ncrcat** to combine the monthly data along the time dimension, and save the result to a single netCDF file:

```
$ ncrcat -h0 month_*nc annual_data.nc
```

You may then discard the `month_*.nc` files if so desired.

8.6.2 Concatenate with Python

You can use the xarray Python package to create a single netCDF file from multiple files. Click HERE to view a sample Python script that does this.

8.7 Regrid netCDF files

The following tools can be used to regrid netCDF data files (such as GEOS-Chem restart files and GEOS-Chem diagnostic files.

8.7.1 Regrid with cdo

cdo includes several tools for regridding netCDF files. For example:

```
# Apply conservative regridding
$ cdo remapcon,gridfile infile.nc outfile.nc
```

For `gridfile`, you can use the files here. Also see this reference.

Issue with cdo remapdis regridding tool

GEOS-Chem user **Bram Maasakkers** wrote:

> I have noticed a problem regridding GEOS-Chem diagnostic file to 2x2.5 using **cdo** version 1.9.4. When I use:
>
> ```
> $ cdo remapdis,geos.2x25.grid GEOSChem.Restart.4x5.nc GEOSChem.Restart.2x25.
> ↪nc
> ```
>
> The last latitudinal band (-89.5) remains empty and gets filled with the standard missing value of cdo, which is really large. This leads to immediate problems in the methane simulation as enormous concentrations enter the domain from the South Pole. For now I've solved this problem by just using bicubic interpolation
>
> ```
> $ cdo remapbic,geos.2x25.grid GEOSChem.Restart.4x5.nc GEOSChem.Restart.2x25.
> ↪nc
> ```

You can also use conservative regridding:

```
$ cdo remapcon,geos.2x25.grid GEOSChem.Restart.4x5.nc GEOSChem.Restart.2x25.nc
```

8.7.2 Regrid with nco

nco also includes several regridding utilities. See the Regridding section of the NCO User Guide for more information.

8.7.3 Regrid with xESMF

xESMF is a universal regridding tool for geospatial data, which is written in Python. It can be used to regrid data not only on cartesian grids, but also on cubed-sphere and unstructured grids.

Note: **xESMF** only handles horizontal regridding.

8.7.4 Regrid with xarray

The xarray Python package has a built-in capability for 1-D interpolation. It wraps the SciPy interpolation module. This functionality can also be used for vertical regridding.

8.7.5 Regrid with gridspec and sparselt

Please see this chapter at gcpy.readthedocs.io for more information about this method of regridding.

8.8 Crop netCDF files

If needed, a netCDF file can be cropped to a subset of the globe with the **nco** or **cdo** utilities (cf. *Useful tools*).

For example, **cdo** has a **selbox** operator for selecting a box by specifying the lat/lon bounds:

```
$ cdo sellonlatbox,lon1,lon2,lat1,lat2 myfile.nc tmp.nc
$ mv tmp.nc myfile.nc
```

See the cdo guide for more information.

8.9 Add a new variable to a netCDF file

You have a couple of options for adding a new variable to a netCDF file (for example, when having to add a new species to an existing GEOS-Chem restart file).

1. You can use **cdo** and **nco** utilities to copy the data from one variable to another variable. For example:

```
#!/bin/bash

# Extract field SpeciesRst_PMN from the original restart file
cdo selvar,SpeciesRst_PMN initial_GEOSChem_rst.4x5_standard.nc NPMN.nc4

# Rename selected field to SpeciesRst_NPMN
ncrename -h -v SpeciesRst_PMN,Species_Rst_NPMN NMPN.nc4

# Append new species to existing restart file
ncks -h -A -M NMPN.nc4 initial_GEOSChem_rst.4x5_standard.nc
```

2. **Sal Farina** wrote a simple Python script for adding a new species to a netCDF restart file:

```
#!/usr/bin/env python

import netCDF4 as nc
import sys
import os

for nam in sys.argv[1:]:
    f = nc.Dataset(nam,mode='a')
    try:
        o = f['SpeciesRst_OCPI']
    except:
        print "SpeciesRst_OCPI not defined"
    f.createVariable('SpeciesRst_SOAP',o.datatype,dimensions=o.dimensions,fill_
↪value=o._FillValue)
```

(continues on next page)

(continued from previous page)

```
    soap = f['SpeciesRst_SOAP']
    soap[:] = 0.0
    soap.long_name= 'SOAP species'
    soap.units =  o.units
    soap.add_offset = 0.0
    soap.scale_factor = 1.0
    soap.missing_value = 1.0e30
    f.close()
```

3. Bob Yantosca wrote this Python script to insert a fake species into GEOS-Chem Classic and GCHP restart files (13.3.0)

```
#!/usr/bin/env python
"""
Adds an extra DataArray for into restart files:
Calling sequence:
    ./append_species_into_restart.py
"""
# Imports
import gcpy.constants as gcon
import xarray as xr
from xarray.coding.variables import SerializationWarning
import warnings

# Suppress harmless run-time warnings (mostly about underflow or NaNs)
warnings.filterwarnings("ignore", category=RuntimeWarning)
warnings.filterwarnings("ignore", category=UserWarning)
warnings.filterwarnings("ignore", category=SerializationWarning)

def main():
    """
    Appends extra species to restart files.
    """
    # Data vars to skip
    skip_vars = gcon.skip_these_vars
    # List of dates
    file_list = [
        'GEOSChem.Restart.fullchem.20190101_0000z.nc4',
        'GEOSChem.Restart.fullchem.20190701_0000z.nc4',
        'GEOSChem.Restart.TOMAS15.20190701_0000z.nc4',
        'GEOSChem.Restart.TOMAS40.20190701_0000z.nc4',
        'GCHP.Restart.fullchem.20190101_0000z.c180.nc4',
        'GCHP.Restart.fullchem.20190101_0000z.c24.nc4',
        'GCHP.Restart.fullchem.20190101_0000z.c360.nc4',
        'GCHP.Restart.fullchem.20190101_0000z.c48.nc4',
        'GCHP.Restart.fullchem.20190101_0000z.c90.nc4',
        'GCHP.Restart.fullchem.20190701_0000z.c180.nc4',
        'GCHP.Restart.fullchem.20190701_0000z.c24.nc4',
        'GCHP.Restart.fullchem.20190701_0000z.c360.nc4',
        'GCHP.Restart.fullchem.20190701_0000z.c48.nc4',
        'GCHP.Restart.fullchem.20190701_0000z.c90.nc4'
    ]
    # Keep all netCDF attributes
    with xr.set_options(keep_attrs=True):
        # Loop over dates
        for f in file_list:
```

(continues on next page)

(continued from previous page)

```
            # Input and output files
            infile = '../' + f
            outfile = f
            print("Creating " + outfile)

            # Open input file
            ds = xr.open_dataset(infile, drop_variables=skip_vars)
            # Create a new DataArray from a given species (EDIT ACCORDINGLY)
            if "GCHP" in infile:
                dr = ds["SPC_ETO"]
                dr.name = "SPC_ETOO"
            else:
                dr = ds["SpeciesRst_ETO"]
                dr.name = "SpeciesRst_ETOO"

            # Update attributes (EDIT ACCORDINGLY)
            dr.attrs["FullName"] = "peroxy radical from ethene"
            dr.attrs["Is_Gas"] = "true"
            dr.attrs["long_name"] = "Dry mixing ratio of species ETOO"
            dr.attrs["MW_g"] = 77.06
            # Merge the new DataArray into the Dataset
            ds = xr.merge([ds, dr], compat="override")

            # Create a new file
            ds.to_netcdf(outfile)

            # Free memory by setting ds to a null dataset
            ds = xr.Dataset()

if __name__ == "__main__":
    main()
```

8.10 Chunk and deflate a netCDF file to improve I/O

We recommend that you **chunk** the data in your netCDF file. Chunking specifies the order in along which the data will be read from disk. The Unidata web site has a good overview of why chunking a netCDF file matters.

For GEOS-Chem with the high-performance option (aka GCHP), the best file I/O performance occurs when the file is split into one chunk per level (assuming your data has a lev dimension). This allows each individual vertical level of data to be read in parallel.

You can use the **nccopy** command of *nco* to do the chunking. For example, say you have a netCDF file called `myfile.nc` with these dimensions:

```
dimensions:
        time = UNLIMITED ; // (12 currently)
        lev = 72 ;
        lat = 181 ;
        lon = 360 ;
```

Then you can use the **nccopy** command to apply the optimal chunking along levels:

```
$ nccopy -c lon/360,lat/181,lev/1,time/1 -d1 myfile.nc tmp.nc
$ mv tmp.nc myfile.nc
```

This will create a new file called `tmp.nc` that has the proper chunking. We then replace `myfile.nc` with this temporary file.

You can specify the chunk sizes that will be applied to the variables in the netCDF file with the **`-c`** argument to **`nccopy`**. To obtain the optimal chunking, the `lon` chunksize must be identical to the number of values along the longitude dimension (e.g. `lon/360` and the `lat` chunksize must be equal to the number of points in the latitude dimension (e.g. `lat/181`).

We also recommend that you **`deflate`** (i.e. compress) the netCDF data variables at the same time you apply the chunking. Deflating can substantially reduce the file size, especially for emissions data that are only defined over the land but not over the oceans. You can deflate the data in a netCDF file by specifying the -d argumetnt to nccopy. There are 10 possible deflation levels, ranging from 0 (no deflation) to 9 (max deflation). For most purposes, a deflation level of 1 (**`d1`**) is sufficient.

The GEOS-Chem Support Team has created a Perl script named nc_chunk.pl (contained in the netcdf-scripts repository at GitHub) that will automatically chunk and compress data for you.

```
$ nc_chunk.pl myfile.nc    # Chunk netCDF file
$ nc_chunk.pl myfile.nc 1  # Chunk and compress file using deflate level 1
```

You can use the **`ncdump -cts myfile.nc`** command to view the chunk size and deflation level in the file. After applying the chunking and compression to myfile.nc, you would see output such as this:

```
dimensions:
        time = UNLIMITED ; // (12 currently)
        lev = 72 ;
        lat = 181 ;
        lon = 360 ;
variables:
        float PRPE(time, lev, lat, lon) ;
                PRPE:long_name = "Propene" ;
                PRPE:units = "kgC/m2/s" ;
                PRPE:add_offset = 0.f ;
                PRPE:scale_factor = 1.f ;
                PRPE:_FillValue = 1.e+15f ;
                PRPE:missing_value = 1.e+15f ;
                PRPE:gamap_category = "ANTHSRCE" ;
                PRPE:_Storage = "chunked" ;
                PRPE:_ChunkSizes = 1, 1, 181, 360 ;
                PRPE:_DeflateLevel = 1 ;
                PRPE:_Endianness = "little" ;\
        float CO(time, lev, lat, lon) ;
                CO:long_name = "CO" ;
                CO:units = "kg/m2/s" ;
                CO:add_offset = 0.f ;
                CO:scale_factor = 1.f ;
                CO:_FillValue = 1.e+15f ;
                CO:missing_value = 1.e+15f ;
                CO:gamap_category = "ANTHSRCE" ;
                CO:_Storage = "chunked" ;
                CO:_ChunkSizes = 1, 1, 181, 360 ;
                CO:_DeflateLevel = 1 ;
                CO:_Endianness = "little" ;\
```

The attributes that begin with a _ character are "hidden" netCDF attributes. They represent file properties instead of user-defined properties (like the long name, units, etc.). The "hidden" attributes can be shown by adding the **`-s`** argument to **`ncdump`**.

CHAPTER

NINE

PREPARE COARDS-COMPLIANT NETCDF FILES

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

9.1 The COARDS netCDF standard

The Harmonized Emissions Compionent (HEMCO) reads data stored in the netCDF file format, which is a common data format used in atmospheric and climate sciences. NetCDF files contain **data arrays** as well as **metadata**, which is a description of the data.

Several netCDF conventions have been developed in order to facilitate data exchange and visualization. The Co-operative Ocean Atmosphere Research Data Service (COARDS) standard defines regular conventions for naming dimensions as well as the attributes describing the data. You will find more information about these conventions in the sections below. HEMCO requires its input data to be adhere to the COARDS standard.

Our *our "Work with netCDF files" supplemental guide* contains detailed instructions on how you can check a netCDF file for COARDS compliance.

9.2 COARDS dimensions

The **dimensions** of a netCDF file define how many grid boxes there are along a given direction. While the COARDS standard does not require any specific n

ames for dimensions, accepted practice is to use these names for rectilinear grids:

`time`
: Specifies the number of points along the time (`T`) axis.

  The *`time`* dimension must always be specified. When you create the netCDF file, you may declare *`time`* to be `UNLIMITED` and then later define its size. This allows you to append further time points into the file later on.

`lev`
: Specifies the number of points along the vertical level (`Z`) axis.

  This dimension may be omitted none of the data arrays in the netCDF file have a vertical dimension.

`lat`
: Specifies the number of points along the latitude (`Y`) axis.

`lon`
: Specifies the number of points along the longitude (`X`) axis.

Note: For non-rectilinear grids (e.g. cubed-sphere), the *lat* and *lon* dimensions may be named NY and NX instead.

9.3 COARDS coordinate vectors

Coordinate vectors (aka **index variables** or **axis variables**) are 1-dimensional arrays that define the values along each axis.

The only COARDS requirement for coordinate vectors are these:

1. Each coordinate vector must be given the same name as the dimension that is used to define it.
2. All of the values contained within a coordinate vector must be either monotonically increasing or monotonically decreasing.

9.3.1 time

A COARDS-compliant *time* coordinate vector will have these features:

```
dimensions
        time = UNLIMITED ; // (12 currently)
. . .
variables
        double time(time) ;
                  time:long_name = "time" ;
                  time:units = "hours since 2010-01-01 00:00:00" ;
                  time:calendar = "standard" ;
                  time:axis = "T";
```

Note: The above was generated by the **ncdump** command.

As you can see, *time* is an 8-byte floating point (aka REAL*8 with 12 time points.

The *time* coordinate vector has following attributes:

time:long_name
A detailed description of the contents of this array. This is usually set to time or Time.

time:units
Specifies the number of hours, minutes, seconds, etc. that has elapsed with respect to a reference datetime YYYY-MM-DD hh:mn:ss. Set this to one of the folllowing values:

- "days since YYYY-MM-DD hh:mn:ss"
- "hours since YYYY-MM-DD hh:mn:ss"
- "minutes since YYYY-MM-DD hh:mn:ss"
- "seconds since YYYY-MM-DD hh:mn:ss"

Tip: We recommend that you choose the reference datetime to correspond to the first time value in the file (i.e. time(0) = 0).

time:calendar
Specifies the calendar used to define the time system. Set this to one of the following values:

standard
Synonym for *gregorian*.

gregorian
Selects the Gregorian calendar system.

time:axis
Identifies the axis (X,Y,Z,T) corresponding to this coordinate vector. Set this to T.

Special considerations for time vectors

1. We have noticed that netCDF files having a *time:units* reference datetime prior to 1900/01/01 00:00:00 may not be read properly when using HEMCO or GCHP within an ESMF environment. We therefore recommend that you use reference datetime values after 1900 whenever possible.
2. Weekly data must contain seven time slices in increments of one day. The first entry must represent Sunday data, regardless of the real weekday of the assigned datetime. It is possible to store weekly data for more than one time interval, in which case the first weekday (i.e. Sunday) must hold the starting date for the given set of (seven) time slices.
   - For instance, weekly data for every month of a year can be stored as 12 sets of 7 time slices. The reference datetime of the first entry of each set must fall on the first day of every month, and the following six entries must be increments of one day.

   Currently, weekly data from netCDF files is not correctly read in an ESMF environment.

9.3.2 lev

A COARDS-compliant *lev* coordinate vector will have these features:

```
dimensions:
        lev = 72 ;
. . .
variables:
        double lev(lev) ;
                lev:long_name = "level" ;
                lev:units = "level" ;
                lev:positive = "up" ;
                lev:axis = "Z" ;
```

Here, *lev* is an 8-byte floating point (aka REAL*8) with 72 levels.

The *lev* coordinate vector has the following attributes:

lev:long_name
A detailed description of the contents of this array. You may set this to values such as:

- "level"
- "GEOS-Chem levels"
- "Eta centers"
- "Sigma centers"

lev:units
(Required) Specifies the units of vertical levels. Set this to one of the following:

- "levels"
- "eta_level"
- "sigma_level"

Important: If you set long_name: to level as well, then HEMCO will be able to regrid between GEOS-Chem vertical grids.

lev:axis
Identifies the axis (X,Y,Z,T) corresponding to this coordinate vector. Set this to Z.

lev:positive
Specifies the direction in which the vertical dimension is indexed. Set this to one of these values:

- "up" (Level 1 is the surface, and level indices increase upwards)
- "down" (Level 1 is the atmosphere top, and level indices increase downwards)

For emisisons and most other data sets, you can set *lev:positive* to "up".

Important: GCHP and the NASA GEOS-ESM use a vertical grid where *lev:positive* is "down".

Additional considerations for lev vectors:

When using GEOS-Chem or HEMCO in a non-ESMF environment, data is interpolated onto the simulation levels if the input data is on vertical levels other than the HEMCO model levels (see HEMCO vertical regridding).

Data on non-model levels must be on a hybrid sigma pressure coordinate system. In order to properly determine the vertical pressure levels of the input data, the file must contain the surface pressure values and the hybrid coefficients (a, b) of the coordinate system. Furthermore, the level variable must contain the attributes standard_name and formula_terms (the attribute positive is recommended but not required). A header excerpt of a valid netCDF file is shown below:

```
float lev(lev) ;
    lev:standard_name = ”atmosphere_hybrid_sigma_pressure_coordinate” ;
    lev:units = ”level” ;
    lev:positive = ”down” ;
    lev:formula_terms = ”ap: hyam b: hybm ps: PS” ;
float hyam(nhym) ;
    hyam:long_name = ”hybrid A coefficient at layer midpoints” ;
    hyam:units = ”hPa” ;
float hybm(nhym) ;
    hybm:long_name = ”hybrid B coefficient at layer midpoints” ;
    hybm:units = ”1” ;
float time(time) ;
    time:standard_name = ”time” ;
    time:units = ”days since 2000-01-01 00:00:00” ;
    time:calendar = ”standard” ;
float PS(time, lat, lon) ;
    PS:long_name = ”surface pressure” ;
    PS:units = ”hPa” ;
float EMIS(time, lev, lat, lon) ;
    EMIS:long_name = ”emissions” ;
    EMIS:units = ”kg m-2 s-1” ;
```

9.3.3 lat

A COARDS-compliant *lat* coordinate vector will have these features:

```
dimensions:
        lat = 181 ;
variables:``
        double lat(lat) ;
                lat:long_name = "Latitude" ;
                lat:units = "degrees_north" ;
                lat:axis = "Y" ;
```

Here, *lat* is an 8-byte floating point (aka `REAL*8`) with 181 values.

The *lat* coordinate vector has the following attributes:

lat:long_name
A detailed description of the contents of this array. Set this to `Latitude`.

lat:units
Specifies the units of latitude. Set this to `degrees_north`.

lat:axis
Identifies the axis `(X,Y,Z,T)` corresponding to this coordinate vector. Set this to `Y`.

9.3.4 lon

A COARDS-compliant *lat* coordinate vector will have these features:

```
dimensions:
        lon = 360 ;
variables:``
        double lon(lon) ;
                lon:long_name = "Longitude" ;
                lon:units = "degrees_east" ;
                lon:axis = "X" ;
```

Here, *lon* is an 8-byte floating point (aka `REAL*8`) with 360 values.

The *lon* coordinate vector has following attributes:

lon:long_name
A detailed description of the contents of this array. Set this to `Longitude`.

lon:units
Specifies the units of latitude. Set this to `degrees_east`.

lon:axis
Identifies the axis `(X,Y,Z,T)` corresponding to this coordinate vector. Set this to `X`.

Longitudes may be represented modulo 360. For example, -180, 180, and 540 are all valid representations of the International Dateline and 0 and 360 are both valid representations of the Prime Meridian. Note, however, that the sequence of numerical longitude values stored in the netCDF file must be monotonic in a non-modulo sense.

Practical guidelines:

1. If your grid begins at the International Dateline (-180°), then place your longitudes into the range -180..180.
2. If your grid begins at the Prime Meridian (0°), then place your longitudes into the range 0..360.

9.4 COARDS data arrays

A COARDS-compliant netCDF file may contain several **data arrays**. In our example file shown above, there are two data arrays:

```
dimensions:
        time = UNLIMITED ; // (12 currently)
        lev = 72 ;
        lat = 181 ;
        lon = 360 ;
variables:``
        float PRPE(time, lev, lat, lon) ;
                PRPE:long_name = "Propene" ;
                PRPE:units = "kgC/m2/s" ;
                PRPE:add_offset = 0.f ;
                PRPE:missing_value = 1.e+15f ;
        float CO(time, lev, lat, lon) ;``
                CO:long_name = "CO" ;
                CO:units = "kg/m2/s" ;
                CO:_FillValue = 1.e+15f ;
                CO:missing_value = 1.e+15f ;
```

These arrays contain emissions for species tracers PRPE (lumped < C3 alkenes) and CO.

9.4.1 Attributes for data arrays

long_name
Gives a detailed description of the contents of the array.

units
Specifies the units of data contained within the array. SI units are preferred.

Special usage for HEMCO:

- Use `kg/m2/s` or `kg m-2 s-1` for emission fluxes of species
- Use `kg/m3` or `kg m-3` for concentration data;
- Use `1` for dimensionless data instead of `unitless`. HEMCO will recognize `unitless`, but it is non-standard and not recommended.

missing_value
Specifies the value that should represent missing data. This should be set to a number that will not be mistaken for a valid data value.

_FillValue
Synonym for *missing_value*. It is recommended to set both *missing_value* and *_FillValue* to the same value. Some data visualization packages look for one but not the other.

9.4.2 Ordering of the data

2D and 3D array variables in netCDF files must have specific dimension order. If the order is incorrect you will encounter netCDF read error "start+count exceeds dimension bound". You can check the dimension ordering of your arrays by using the **ncdump** command as shown below:

```
$ ncdump file.nc -h
```

Be sure to check the dimensions listed next to the array name rather than the ordering of the dimensions listed at the top of the **ncdump** output.

The following dimension orders are acceptable:

```
array(time,lat,lon)
array(time,lat,lon,lev)
```

The rest of this section explains why the dimension ordering of arrays matters.

When you use **ncdump** to examine the contents of a netCDF file, you will notice that it displays the dimensions of the data in the opposite order with respect to Fortran. In our sample file, **ncdump** says that the CO and PRPE arrays have these dimensions:

```
CO(time,lev,lat,lon)
PRPE(time,lev,lat,lon)
```

But if you tried to read this netCDF file into GEOS-Chem (or any other program written in Fortran), you must use data arrays that have these dimensions:

```
CO(lon,lat,lev,time)
PRPE(lon,lat,lev,time)
```

Here's why:

Fortran is a **column-major** language, which means that arrays are stored in memory by columns first, then by rows. If you have declared an arrays such as:

```
INTEGER             :: I, J, L, T
INTEGER, PARAMETER  :: N_LON  = 360
INTEGER, PARAMETER  :: N_LAT  = 181
INTEGER, PARAMETER  :: N_LEV  = 72
INTEGER, PARAMTER   :: N_TIME = 12
REAL*4              :: CO   (N_LON,N_LAT,N_LEV,N_TIME)
REAL*4              :: PRPE(N_LON,N_LAT,N_LEV,N_TIME)
```

then for optimal efficiency, the leftmost dimension (`I`) needs to vary the fastest, and needs to be accessed by the innermost DO-loop. Then the next leftmost dimension (`J`) should be accessed by the next innermost DO-loop, and so on. Therefore, the proper way to loop over these arrays is:

```
DO T = 1, N_TIME
DO L = 1, N_LEV
DO J = 1, N_LAT
DO I = 1, N_LON
   CO   (I,J,L,N) = ...
   PRPE(I,J,L,N) = ...
ENDDO
ENDDO
ENDDO
ENDDO
```

Note that the `I` index is varying most often, since it is the innermost DO-loop, then `J`, `L`, and `T`. This is opposite to how a car's odometer reads.

If you loop through an array in this fashion, with leftmost indices varying fastest, then the code minimizes the number of times it has to load subsections of the array into cache memory. In this optimal manner of execution, all of the array elements sitting in the cache memory are read in the proper order before the next array subsection needs to be loaded into the cache. But if you step through array elements in the wrong order, the number of cache loads is proportionally increased. Because it takes a finite amount of time to reload array elements into cache memory, the more times you have to access the cache, the longer it will take the code to execute. This can slow down the code dramatically.

On the other hand, C is a **row-major** language, which means that arrays are stored by rows first, then by columns. This means that the outermost do loop (`I`) is varying the fastest. This is identical to how a car's odometer reads.

If you use a Fortran program to write data to disk, and then try to read that data from disk into a program written in C, then unless you reverse the order of the DO loops, you will be reading the array in the wrong order. In C you would have to use this ordering scheme (using Fortran-style syntax to illustrate the point):

```
DO I = 1, N_LON
DO J = 1, N_LAT
DO L = 1, N_LEV
DO T = 1, N_TIME
   CO(T,L,J,I)   = ...
   PRPE(T,L,J,I) = ...
ENDDO
ENDDO
ENDDO
ENDDO
```

Because **ncdump** is written in C, the order of the array appears opposite with respect to Fortran. The same goes for any other code written in a row-major programming language.

9.5 COARDS Global attributes

Global attributes are netCDF attributes that contain information about a netCDF file, as opposed to information about an individual data array.

From our example in the *Examine the contents of a netCDF file*, the output from **ncdump** showed that our sample netCDF file has several global attributes:

```
// global attributes:
            :Title = "COARDS/netCDF file containing X data"
            :Contact = "GEOS-Chem Support Team (geos-chem-support@as.harvard.edu)" ;
            :References = "www.geos-chem.org; wiki.geos-chem.org" ;
            :Conventions = "COARDS" ;
            :Filename = "my_sample_data_file.1x1"
            :History = "Mon Mar 17 16:18:09 2014 GMT" ;
            :ProductionDateTime = "File generated on: Mon Mar 17 16:18:09 2014 GMT" ;
            :ModificationDateTime = "File generated on: Mon Mar 17 16:18:09 2014 GMT"␣
↪;
            :VersionID = "1.2" ;
            :Format = "NetCDF-3" ;
            :Model = "GEOS5" ;
            :Grid = "GEOS_1x1" ;
            :Delta_Lon = 1.f ;
            :Delta_Lat = 1.f ;
            :SpatialCoverage = "global" ;
```

(continues on next page)

(continued from previous page)

```
            :NLayers = 72 ;
            :Start_Date = 20050101 ;
            :Start_Time = 00:00:00.0 ;
            :End_Date = 20051231 ;
            :End_Time = 23:59:59.99999 ;
```

Title (or title)
: Provides a short description of the file.

Contact (or contact)
: Provides contact information for the person(s) who created the file.

References (or references)
: Provides a reference (citation, DOI, or URL) for the data contained in the file.

Conventions (or conventions)
: Indicates if the netCDF file adheres to a standard (e.g. COARDS or CF).

Filename (or filename)
: Specifies the name of the file.

History (or history)
: Specifies the datetime of file creation, and of any subsequent modifications.

Note: If you edit the file with **nco** or **cdo**, then this attribute will be updated to reflect the modification that was done.

Format (or format)
: Specifies the format of the netCDF file (such as `netCDF-3` or `netCDF-4`).

9.6 For more information

Please see our *Work with netCDF files* Supplemental Guide for more information about commands that you can use to combine, edit, or maniuplate data in netCDF files.

CHAPTER

TEN

VIEW RELATED DOCUMENTATION

Table 1: **GEOS-Chem web, wiki and Youtube channel**

Site	Link
GEOS-Chem web site	geos-chem.org
GEOS-Chem wiki	wiki.geos-chem.org
Video tutorials on Youtube (various)	youtube.com/c/geoschem

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

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

CHAPTER

ELEVEN

CONTRIBUTING GUIDELINES

Thank you for looking into contributing to HEMCO! HEMCO is a grass-roots model that relies on contributions from community members like you. Whether you're new to HEMCO or a longtime user, you're a valued member of the community, and we want you to feel empowered to contribute.

11.1 We use GitHub and ReadTheDocs

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

We use ReadTheDocs to host the HEMCO user documentation: https://hemco.readthedocs.io.

11.2 How to submit changes

We use GitHub Flow, so all changes happen through pull requests. This workflow is described here.

As the author you are responsible for:

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

11.3 Coding conventions

The HEMCO 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.

11.4 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 might be relevant, including the motivation for the feature.

11.5 How to report a bug

Please see "Support Guidelines".

11.6 Where can I ask for help?

Please see "Support Guidelines".

CHAPTER

TWELVE

SUPPORT GUIDELINES

HEMCO 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.

12.1 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 simulation log files and instructions for replicating the bug.

12.2 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.

12.3 What type of support can I expect?

We will be happy to assist you in resolving bugs and technical issues that arise when compiling or running HEMCO.

Even though we can assist in several ways, we cannot possibly do everything. We rely on HEMCO users being resourceful and willing to try to resolve problems on their own to the greatest extent possible.

12.4 How to submit changes

Please see "Contributing Guidelines".

12.5 How to request an enhancement

Please see "Contributing Guidelines".

CHAPTER

THIRTEEN

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.

13.1 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`.

13.2 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.

13.3 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.

BIBLIOGRAPHY

[Bey et al., 2001] Bey, I., Jacob, D. J., Yantosca, R. M., Logan, J. A., Field, B. D., Fiore, A. M., Li, Q., Liu, H. Y., Mickley, L. J., and Schultz, M. G. Global modeling of tropospheric chemistry with assimilated meteorology: Model description and evaluation. *J. Geophys. Res.*, 106(D19):23073–23095, Oct 2001. doi:10.1029/2001JD000807.

[Ginoux et al., 2001] Ginoux, P., Chin, M., I. Tegen, Prospero, J., Hoben, B., Dubovik, O., and Lin, S.J. Sources and distributions of dust aerosols simulated with the gocart model. *J. Geophys. Res.*, 106(D17):20255–20273, 2001.

[Gong 2003] Gong, S.L. A parameterization of sea-salt aerosol source function for sub- and super-micron particles. *Global Biogeochem. Cycles*, 17:1097ff, 2003. doi:10.1029/2003GB002079.

[Guenther et al., 2012] Guenther, A.B., Jiang, X., Heald, C.L., Sakulyanontvittaya, T., Duhl, T., Emmons, L. K., and Wang, X. The model of emissions of gases and aerosols from nature version 2.1 (megan2.1): an extended and updated framework for modeling biogenic emissions. *Geosci. Model Dev.*, 5:1471—1492, 2012. doi:10.5194/gmd-5-1471-2012.

[Hudman et al., 2012] Hudman, R.C., Moore, N.E., Mebust, A.K., Martin, R.V., Russell, A.R., Valin, L.C., and Cohen, R.C. Steps towards a mechanistic model of global soil nitric oxide emissions: implementation and space based-constraints. *Atmos. Chem. Phys.*, 12:7779—7795, 2012. doi:10.5194/acp-12-7779-2012.

[Jacob et al., 1997] Jacob, D.J., Prather, M.J., and Rasch, P.J. e. al. Evaluation and intercomparison of global atmospheric transport models using rn-222 and other short-lived tracers. *J. Geophys. Res*, 102(D5):5953–5970, 1997.

[Jaegle et al., 2011] Jaeglé, L., Quinn, P.K., Bates, T.S., Alexander, B., and Lin, J.-T. Global distribution of sea salt aerosols: new constraints from in situ and remote sensing observations. *Atmos. Chem. Phys.*, 2011. doi:10.5194/acp-11-3137-2011.

[Johnson 2010] Johnson, M. T. A numerical scheme to calculate temperature and salinity dependent air-water transfer velocities for any gas. *Ocean Sci.*, 6:913–922, 2010. doi:10.5194/os-6-913-2010.

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

[Lamarque et al., 2010] Lamarque, J.-F., Bond, T. C., Eyring, V., Granier, C., Heil, A., Klimont, Z., Lee, D., Liousse, C., Mieville, A., Owen, B., Schultz, M. G., Shindell, D., Smith, S. J., Stehfest, E., Van Aardenne, J., Cooper, O. R., Kainuma, M., Mahowald, N., McConnell, J. R., Naik, V., Riahi, K., and van Vuuren, D. P. Historical (1850–2000) gridded anthropogenic and biomass burning emissions of reactive gases and aerosols: methodology and application. *Atm. Chem. Phys.*, 10:7017—7039, 2010.

[Lin et al., 2021] Lin, H., Jacob, D. J., Lundgren, E. W., Sulprizio, M. P., Keller, C. A., Fritz, T. M., Eastham4, S. D., Emmons, L. K., Campbell, P. C., Baker, B., Saylor, R. D., and Montuoro, R. Harmonized emissions component (hemco) 3.0 as a versatile emissions component for atmospheric models: application in

the geos-chem, nasa geos, wrf-gc, cesm2, noaa gefs-aerosol, and noaa ufs models. *Geosci. Model. Dev.*, 14:5487–5506, 2021. doi:0.5194/gmd-14-5487-2021.

[Luo et al., 2020] Luo, G., Yu, F., and Moch, J. Further improvement of wet process treatments in geos-chem v12.6.0: impact on global distributions of aerosols and aerosol precursors. *Geosci. Model. Dev.*, 13:2879–2903, 2020. doi:10.5194/gmd-13-2879-2020.

[Murray et al., 2012] Murray, L.T., Jacob, D.J., Logan, J.A., Hudman, R.C., and Koshak, W.J. Optimized regional and interannual variability of lightning in a global chemical transport model constrained by lis/otd satellite data. *J. Geophys. Res.-Atmos*, 2012. doi:10.1029/2012JD017934.

[Nightingale et al., 2000] Nightingale, P.D., Malin, G., Law, C.S., Watson, A.J., Liss, P.S., Liddicoat, M.I., Boutin, J., and Upstill-Goddard, R.C. In situ evaluation of air-sea gas exchange parameterizations using novel conservative and volatile tracers. *Global Biogeochem. Cycles*, 14:373—387, 2000. doi:10.1029/1999GB900091.

[Stettler et al., 2011] Stettler, M., Eastham, S., and Barrett, S. Air quality and public health impacts of uk airports. part i: emissions. *Atmos. Env.*, 45:5415–5424, 2011.

[van der Werf et al., 2010] van der Werf, G.R., Randerson, J.T., Giglio, L., Collatz, G. J., Mu, M., Kasibhatla, P.S., Morton, D.C., DeFries, R.S., Y., J., and van Leeuwen, T. T. Global fire emissions and the contribution of deforestation, savanna, forest, agricultural, and peat fires (1997–2009). *Atm. Chem. Phys.*, 10:11707–11735, 2010.

[Vestreng et al., 2009] Vestreng, V., Ntziachristos, L., Semb, A., Reis, S., Isaksen, I.S.A., and Tarrasón, L. Evolution of nox emissions in europe with focus on road transport control measures. *Atm. Chem. Phys.*, 9:1503—1520, 2009.

[Vinken et al., 2011] Vinken, G.C.M., Boersma, K.F., Jacob, D.J., and Meijer, E.W. Accounting for non-linear chemistry of ship plumes in the geos-chem global chemistry transport model. *Atmos. Chem. Phys.*, 11:11707–11722, 2011. doi:10.5194/acp-11-11707-2011.

[Wiedinmyer et al., 2014] Wiedinmyer, C., Yokelson, R.J, and Gullett, B.K. Global emissions of trace gases, particulate matter, and hazardous air pollutants from open burning of domestic waste. *Env. Sci. Tech*, 16:9523—9530, 2014.

[Zender et al., 2003] Zender, C.S., Bian, H., and Newman, D. Mineral dust entrainment and deposition (dead) model: description and 1990s dust climatology. *J. Geophys. Res.-Atmos*, 108:4416ff, 2003. doi:10.1029/2002JD002775.

[Zhang et al., 2021] Zhang, B., Liu, H., Crawford, J.H., G. Chen, Fairlie, T.D., Chambers, S., Kang, C.-H., Williams, A.G., Zhang, K., Considine, D.B., Sulprizio, M.P., and Yantosca, R.M. Simulation of radon-222 with the geos-chem global model: emissions, seasonality, and convective transport. *Atm. Chem. Phys.*, 21:1861–1887, 2021. doi:10.5194/acp-21-1861-2021.

INDEX

D

E

F

G

H

I

J

L

M

N

O

P

R

S

T

U

V

W

X

Y

Z