KPP-for-GEOS-Chem

Release 2.3.1 $_{gC}$

GEOS-Chem Support Team

May 07, 2021

BASIC INFORMATION

KPP-for-GEOS-Chem is a clean implementation of the Kinetic Pre Processor (KPP) that has been customized for **GEOS-Chem** v11-01 and later versions. You can use **KPP-for-GEOS-Chem** to create custom **GEOS-Chem** chemistry mechanisms (or to edit existing mechanisms).

CHAPTER

ONE

OVERVIEW

1.1 About KPP

KPP (The Kinetic PreProcessor, by A. Sandu et al), translates a chemical mechanism specification from plain-text format to source code. You may select the numerical integrator to be used (Runge-Kutta, LSODES, Rosenbrock, etc.) and the language for the generated source code (Fortran-90, Fortran-77, C, Matlab). The resulting source code files may then be integrated into an atmospheric chemistry model, chemical box model, or other application.

The latest public release is **KPP 2.2.3_01**, which may be obtained as a tarball from the KPP web site. For convenience, the GEOS-Chem Support Team has created a Github repository that contains **KPP 2.2.3_01** in the `main` branch.

1.2 About KPP-for-GEOS-Chem

KPP-for-GEOS-Chem is a clean implementation of KPP that has been customized for **GEOS-Chem** v11-01 and later versions. You can use **KPP-for-GEOS-Chem** to modify existing **GEOS-Chem** chemistry mechanisms or create new mechanisms.

Current and previous **KPP-for-GEOS-Chem** releases are kept in the `GC_updates` branch of the KPP repository on Github.

> **Important:** The version of **KPP-for-GEOS-Chem** that you will need to use currently depends on the version of **GEOS-Chem** that you have. In the future we hope to make **KPP-for-GEOS-Chem** version a submodule of **GEOS-Chem**.

The **GEOS-Chem** source code module `flexchem_mod.F90` serves as the connection between the chemical mechanism solver files generated by **KPP-for-GEOS-Chem** and the species concentration array in **GEOS-Chem**. In `flexchem_mod.F90`, species concentrations, photoylsis rates, meteorology fields, and other relevant information are passed to the chemistry mechanism routines created by **KPP-for-GEOS-Chem**. These routines compute reaction rates, perform the forward integration, and pass the updated species concentration back to **GEOS-Chem**.

The main benefits of **KPP-for-GEOS-Chem** are:

1. Better documentation of chemical mechanisms;
2. Easy switching between chemical mechanisms.
3. Optimized chemistry computations; and
4. Removal of the **SMVGEAR** solver from **GEOS-Chem**.

CHAPTER

TWO

SYSTEM REQUIREMENTS

KPP-for-GEOS-Chem requires the following packages:

1. A C-language compiler (such as **gcc**, from the GNU Compiler Collection)
2. The **flex** lexical parser library. This is often installed on many computer systems by default. It can also be easily installed with a package manager such as **spack**.
3. Python (2.7.5 or greater). This is required in order to post-process source code created by KPP-for-GEOS-Chem to include modifications for the OH reactivity diagnostic.

Depending on your setup, you might have to load these packages with the **module load** or **spack load** commands. Ask your sysadmin for more information.

CHAPTER

THREE

KEY REFERENCES

3.1 KPP

1. V. Damian, A. Sandu, M. Damian, F. Potra, and G.R. Carmichael: *The Kinetic PreProcessor KPP – A Software Environment for Solving Chemical Kinetics*, Computers and Chemical Engineering, Vol. 26, No. 11, 1567-1579 (2002).
2. A. Sandu, D. Daescu, and G.R. Carmichael: *Direct and Adjoint Sensitivity Analysis of Chemical Kinetic Systems with KPP: I – Theory and Software Tools*, Atmospheric Environment, Vol. 37, 5083-5096 (2003).
3. D. Daescu, A. Sandu, and G.R. Carmichael: *Direct and Adjoint Sensitivity Analysis of Chemical Kinetic Systems with KPP: II – Validation and Numerical Experiments*, Atmospheric Environment, Vol. 37, p. 5097-5114 (2003).
4. A. Sandu and R. Sander. *Technical Note: Simulating chemical systems in Fortran90 and Matlab with the kinetic preprocessor KPP-2.1*, Atmospheric Chemistry and Physics, Vol. 6, 187-195, (2006).
5. The KPP 2.1 user manual.

3.2 GEOS-Chem

1. GEOS-Chem was first described in [[Bey et al., 2001]].
2. HEMCO is described in [[Keller et al., 2014]].
3. Columnar operators are described in [[Long et al., 2015]].
4. GEOS-Chem High Performance (GCHP) is described in [[Eastham et al., 2018]].
5. GCHP execution on the cloud and MPI considerations are described in [[Zhuang et al., 2020]].
6. Grid-stretching is described in [[Bindle et al., 2020]].

References

CHAPTER FOUR

INSTALLING KPP-FOR-GEOS-CHEM

Once you have made sure that your system meets all the *System requirements*, you may download and install **KPP-for-GEOS-Chem**.

4.1 Downloading

The following sections describe how to download **KPP-for-GEOS-Chem** with Git. The version of **KPP-for-GEOS-Chem** that you will need to use depends on the GEOS-Chem version that you have.

Important: Do not download the **KPP-for-GEOS-Chem** source code into your **GEOS-Chem** source code directory! This will avoid confusion with the KPP folder within **GEOS-Chem**. This folder contains Fortran-90 source code files that are generated by **KPP-for-GEOS-Chem**.

4.1.1 KPP-for-GEOS-Chem 2.3.2_gc – currently experimental

The current version of **KPP-for-GEOS-Chem 2.3.2_gc** is for use with ongoing GEOS-Chem development. It should not yet be used with any released GEOS-Chem versions.

To download this version, type at the command line:

```
$ git clone -b GC_updates https://github.com/geoschem/KPP.git
```

You may now proceed to our *Compiling* section below

4.1.2 KPP-for-GEOS-Chem 2.3.1.gc – for GEOS-Chem 13.1.0 and later

If you will be working with GEOS-Chem 13.1.0 and later, then you will need to download **KPP-for-GEOS-Chem 2.3.1_gc**. Type at the command line:

```
$ git clone -b 2.3.1_gc https://github.com/geoschem/KPP.git
$ git -C KPP branch 2.3.1_gc
```

You may now proceed to our *Compiling* section below.

4.1.3 KPP-for-GEOS-Chem 2.2.5_gc – for GEOS-Chem 13.0.2 and earlier

If you will be working with GEOS-Chem versions older than 13.1.0, then you will need to download **KPP-for-GEOS-Chem 2.2.5_gc**. Use the following commands:

```
$ git clone -b 2.2.5_gc https://github.com/geoschem/KPP.git
$ git -C KPP branch 2.2.5_gc
```

4.2 Compiling

Build the KPP-for-GEOS-Chem executable file with these commands:

```
$ cd KPP/kpp-code
$ make distclean
$ make all
```

If the build completes successfully, you will see the executable file `KPP/kpp-code/bin/kpp`.

4.3 Setting the path

Once have built **KPP-for-GEOS-Chem**, you must add the path to the executable file to your `PATH` environment variable.

If you use the bash Unix shell, add these lines to your `~/.bash_aliases` file. If you don't have a `~/.bash_aliases` file, you can add these lines to your `~/.bashrc` file instead.)

```
export PATH=$PATH:/PATH_TO_KPP/KPP/kpp-code/bin/
export KPP_HOME=PATH_TO_KPP/KPP/kpp-code`
```

If you use the csh or tcsh Unix shell, add these lines to your `~/.cshrc` file:

```
setenv PATH $PATH:/PATH_TO_KPP/KPP/kpp-code/bin/
setenv KPP_HOME=PATH_TO_KPP/KPP/kpp-code
```

Note:

- For example, if you installed FlexChem-KPP into your home directory, then `PATH_TO_KPP` would be `~/KPP`, etc.

CHAPTER

FIVE

CREATING FORTRAN-90 CHEMICAL MECHANISM MODULES FOR GEOS-CHEM

5.1 Navigate to the KPP folder in your GEOS-Chem source code

At this point you can now use **`KPP-for-GEOS-Chem`** to generate Fortran-90 source code files that will solve the chemical mechanism in an efficient manner.

Important: The configuration files that define the various GEOS-Chem chemical mechanisms are kept with the GEOS-Chem source code. For the purposes of this tutorial, we will assume that you have already downloaded GEOS-Chem (either as GEOS-Chem "Classic" or GCHP) to your computer system.

Navigate to this folder in your **`GEOS-Chem`** source code:

- GEOS-Chem 12.9.3 and prior versions: `KPP`
- GEOS-Chem 13.0.0 and later versions: `src/GEOS-Chem/KPP`
- GCHP 13.0.0 and later versions: `src/GCHP_GridComp/GEOSChem_GridComp/geos-chem/KPP`

Here you will find a sub-folder named `custom` and a script named `build_mechanism.sh`. You will also find a few other sub-folders with names such as `fullchem` (in GEOS-Chem 13), or `Standard`, `Tropchem`, etc. (in GEOS-Chem 12).

Important: These `fullchem`, `Standard`, `Tropchem`, etc. sub-folders contain Fortran-90 source code files that have been generated by **`KPP-for-GEOS-Chem`**. You should leave these files untouched.

The `custom` folder contains sample chemical mechanism specification files (`custom.eqn` and `gckpp.kpp`), which are copies of the default GEOS-Chem mechanism. You can edit these files to define your own custom mechanism (see subsequent sections for detailed instructions).

5.2 Run the build_mechanism.sh script

Once you are satisfied with your custom mechanism specification you may now use KPP-for-GEOS-Chem to build the source code files for GEOS-Chem.

Return to the KPP folder containing `build_mechanism.sh` and then type:

```
$ ./build_mechanism.sh custom
```

The `build_mechanism.sh` script runs the **`KPP-for-GEOS-Chem`** executable (which is named `gckpp`) on the `gckpp.kpp` configuration file. It also runs a python script to generate code for the OH reactivity diagnostic.

Once you run the `build_mechanism.sh` script, you will see output similar to this:

```
This is KPP-X.Y.Z_gc.
KPP is parsing the equation file.
KPP is computing Jacobian sparsity structure.
KPP is starting the code generation.
KPP is initializing the code generation.
KPP is generating the monitor data:
  - gckpp_Monitor
KPP is generating the utility data:
  - gckpp_Util
KPP is generating the global declarations:
  - gckpp_Main
KPP is generating the ODE function:
  - gckpp_Function
KPP is generating the ODE Jacobian:
  - gckpp_Jacobian
  - gckpp_JacobianSP
KPP is generating the linear algebra routines:
  - gckpp_LinearAlgebra
KPP is generating the utility functions:
  - gckpp_Util
KPP is generating the rate laws:
  - gckpp_Rates
KPP is generating the parameters:
  - gckpp_Parameters
KPP is generating the global data:
  - gckpp_Global
KPP is generating the driver from none.f90:
  - gckpp_Main
KPP is starting the code post-processing.

KPP has succesfully created the model "gckpp".

Reactivity consists of 172 reactions
Written to gckpp_Util.F90
```

where `X.Y.Z` denotes the **`KPP-for-GEOS-Chem`** version that you are using.

If this process is successful, the `custom` folder should now be populated with several `.F90` source code files:

```
CMakeLists.txt*       gckpp_Initialize.F90  gckpp_LinearAlgebra.F90  gckpp_Precision.
↪F90
custom.eqn            gckpp_Integrator.F90  gckpp.map                gckpp_Rates.F90
gckpp_Function.F90    gckpp_Jacobian.F90    gckpp_Model.F90          gckpp_Util.F90
```

(continues on next page)

(continued from previous page)

```
gckpp_Global.F90     gckpp_JacobianSP.F90  gckpp_Monitor.F90         Makefile_gckpp
gckpp_HetRates.F90@  gckpp.kpp             gckpp_Parameters.F90
```

These files contain optimized Fortran-90 instructions for solving the chemical mechanism that you have specified.

CHAPTER

SIX

TELLING GEOS-CHEM TO USE A CUSTOM MECHANISM

GEOS-Chem will always use the default mechanism (which is named `fullchem` or `Standard` depending on which version you have). To tell GEOS-Chem to use the `custom` mechanism, follow these steps.

6.1 For GEOS-Chem versions using CMake

If your **GEOS-Chem** version uses CMake, thenen navigate to your build directory.

Note: GEOS-Chem Classic run directories have a subdirectory named `build` in which you can configure and build GEOS-Chem. If you don't have a build directory, you can add one to your run directory.

For more information about the GEOS-Chem and GCHP configuration process, please see GEOS-Chem manual and gchp.readthedocs.io.

6.1.1 Configuring GEOS-Chem

From the build directory, type:

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

NOTE: In some later **GEOS-Chem 12** versions, the proper command is:

```
$ cmake ../CodeDir -DCHEM=custom
```

Please consult the GEOS-Chem manual for more information.

You should see output similar to this written to the screen:

```
-- General settings:
   * CUSTOMMECH:   **ON** OFF
```

This confirms that the custom mechanism has been selected.

6.1.2 Compiling GEOS-Chem

Once you have configured **GEOS-Chem** to use the `custom` mechanism, you may build the exectuable. Type:

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

The executable file (`gcclassic` or `gchp`, depending on which mode of GEOS-Chem that you are using) will be placed in the run directory.

6.2 For GEOS-Chem versions using GNU Make

If you are using an older version of **GEOS-Chem** that is only compatible with GNU Make, then use this command to compile the executable:

```
$ make -j CHEM=Custom ...etc other build options...
```

Please consult the GEOS-Chem manual for more information.

CHAPTER

SEVEN

USER-EDITABLE CONFIGURATION FILES

In the `KPP/custom` folder within the GEOS-Chem source code, you will find two files that define the chemical mechanism:

7.1 custom.eqn

The `custom.eqn` configuration file contains:

- List of active species
- List of inactive species
- Gas-phase reactions
- Heterogeneous reactions
- Photolysis reactions

7.2 gckpp.kpp

The `gckpp.kpp` configuration file contains:

- Solver options
- Production and loss family definitions
- Functions to compute reaction rates
- Global definitions

CHAPTER

EIGHT

ADDING SPECIES

8.1 Chemically-active species

List chemically-active (aka variable) species in the #DEFVAR section of custom.eqn, as shown below:

```
#DEFVAR
A3O2        = IGNORE; {CH3CH2CH2OO; Primary RO2 from C3H8}
ACET        = IGNORE; {CH3C(O)CH3; Acetone}
ACTA        = IGNORE; {CH3C(O)OH; Acetic acid}
...etc ...
```

8.2 Chemically-inactive species

List species whose concentrations do not change in the #DEFFIX of custom.eqn, as shown below:

```
#DEFFIX
H2          = IGNORE; {H2; Molecular hydrogen}
N2          = IGNORE; {N2; Molecular nitrogen}
O2          = IGNORE; {O2; Molecular oxygen}
... etc ...
```

Species may be listed in any order, but we have found it convenient to list them alphabetically.

CHAPTER

NINE

ADDING REACTIONS

9.1 Gas-phase reactions

List gas-phase reactions first in the `#EQUATIONS` section of `custom.eqn`.

```
#EQUATIONS
//
// Gas-phase reactions
//
...skipping over the comment header...
//
O3 + NO = NO2 + O2 :                          GCARR(3.00E-12, 0.0, -1500.0);
O3 + OH = HO2 + O2 :                          GCARR(1.70E-12, 0.0, -940.0);
O3 + HO2 = OH + O2 + O2 :                     GCARR(1.00E-14, 0.0, -490.0);
O3 + NO2 = O2 + NO3 :                         GCARR(1.20E-13, 0.0, -2450.0);
... etc ...
```

9.1.1 General form

No matter what reaction is being added, the general procedure is the same. A new line must be added to `custom.eqn` of the following form:

```
A + B = C + 2.000D : RATE_LAW_FUNCTION(ARG_A, ARG_B ...);
```

The denotes the reactants (A and B) as well as the products (C and D) of the reaction. If exactly one molecule is consumed or produced, then the factor can be omitted; otherwise the number of molecules consumed or produced should be specified with at least 1 decimal place of accuracy. The final section, between the colon and semi-colon, specifies the function `RATE_LAW_FUNCTION` and its arguments which will be used to calculate the reaction rate constant k. Rate-law functions are specified in the gckpp.kpp file.

For an equation such as the one above, the overall rate at which the reaction will proceed is determined by `k[A][B]`. However, if the reaction rate does not depend on the concentration of A or B, you may write it with a constant value, such as:

```
A + B = C + 2.000D : 8.95d-17
```

This will save the overhead of a function call.

9.1.2 Rates for two-body reactions according to the Arrhenius law

For many reactions, the calculation of k follows the Arrhenius law:

```
k = a0 + ( 300 / TEMP )**b0 + EXP( c0 / TEMP )
```

For example, the JPL chemical data evaluation (Feb 2017) specifies that the reaction O3 + NO produces NO2 and O2, and its Arrhenius parameters are A = 3.0x10^-12 and E/R = c0 = 1500.

To specify a two-body reaction whose rate follows the Arrhenius law, you can use the `GCARR` rate-law function, which is defined in `gckpp.kpp`. For example, the entry for the `O3 + NO = NO2 + O2` reaction can be written as in `custom.eqn` as:

```
O3 + NO = NO2 + O2 : GCARR(3.00E12, 0.0, -1500.0);
```

9.1.3 Other rate-law functions

The `gckpp.kpp` file contains other rate law functions, such as those required for three-body, pressure-dependent reactions. Any rate function which is to be referenced in the `custom.eqn` file must be available in `gckpp.kpp` prior to building the reaction mechanism.

9.1.4 Making your rate law functions computationally efficient

We recommend writing your rate-law functions so as to avoid explicitly casting variables from `REAL*4` to `REAL*8`. Code that looks like this:

```
REAL, INTENT(IN) :: A0, B0, C0
rate = DBLE(A0) + ( 300.0 / TEMP )**DBLE(B0) + EXP( DBLE(C0)/ TEMP )
```

Can be rewritten as:

```
REAL(kind=dp), INTENT(IN) :: A0, B0, C0
rate = A0 + ( 300.0d0 / TEMP )**B0 + EXP( C0/ TEMP )
```

Not only do casts lead to a loss of precision, but each cast takes a few CPU clock cycles to execute. Because these rate-law functions are called for each cell in the chemistry grid, wasted clock cycles can accumulate into a noticeable slowdown in execution.

You can also make your rate-law functions more efficient if you rewrite them to avoid computing terms that evaluate to 1. We saw *above* that the rate of the reaction `O3 + NO = NO2 + O2` can be computed according to the Arrhenius law. But because `b0 = 0`, term `(300/TEMP)**b0` evaluates to 1. We can therefore rewrite the computation of the reaction rate as:

```
k = 3.0x10^-12 + EXP( 1500 / TEMP )
```

Tip: The `EXP()` and `**` mathematical operations are among the most costly in terms of CPU clock cycles. Avoid calling them whenever necessary.

A recommended implementation would be to create separate rate-law functions that take different arguments depending on which parameters are nonzero. For example, the Arrhenius law function `GCARR` can be split into multiple functions:

1. `GCARR_abc(a0, b0, c0)`: Use when `a0 > 0` and `b0 > 0` and `c0 > 0`

2. `GCARR_ab(a0, b0)`: Use when `a0 > 0` and `b0 > 0`
3. `GCARR_ac(a0, c0)`: Use when `a0 > 0` and `c0 > 0`

Thus we can write the O3 + NO reaction in `custom.eqn` as:

```
O3 + NO = NO2 + O2 : GCARR_ac(3.00d12, -1500.0d0);
```

using the rate law function for when both `a0 > 0` and `c0 > 0`.

9.2 Heterogeneous reactions

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

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

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

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

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

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

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

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

9.3 Photolysis reactions

List photolysis reactions after the heterogeneous reactions, as shown below.

```
//
// Photolysis reactions
//
O3 + hv = O + O2 :                              PHOTOL(2);      {2014/02/03; Eastham2014;
↪ SDE}
O3 + hv = O1D + O2 :                            PHOTOL(3);      {2014/02/03; Eastham2014;
↪ SDE}
O2 + hv = 2.000O :                              PHOTOL(1);      {2014/02/03; Eastham2014;
↪ SDE}
... etc ...
NO3 + hv = NO2 + O :                            PHOTOL(12);     {2014/02/03; Eastham2014;
↪ SDE}
... etc ...
```

A photolysis reaction can be specified by giving the correct index of the `PHOTOL` array. This index can be determined by inspecting the file `FJX_j2j.dat`.

Tip: See the PHOTOLYSIS MENU section of input.geos to determine the folder in which `FJX_j2j.dat` is located.

For example, one branch of the NO3 photolysis reaction is specified in the `custom.eqn` file as

```
NO3 + hv = NO2 + O : PHOTOL(12)
```

Referring back to `FJX_j2j.dat` shows that reaction 12, as specified by the left-most index, is indeed NO3 = NO2 + O:

```
12 NO3        PHOTON     NO2        O                         0.886 /NO3    /
```

If your reaction is not already in `FJX_j2j.dat`, you may add it there. You may also need to modify `FJX_spec.dat` (in the same folder ast `FJX_j2j.dat`) to include cross-sections for your species. Note that if you add new reactions to `FJX_j2j.dat` you will also need to set the parameter `JVN_` in GEOS-Chem module `Headers/CMN_FJX_MOD.F90` to match the total number of entries.

If your reaction involves new cross section data, you will need to follow an additional set of steps. Specifically, you will need to:

1. Estimate the cross section of each wavelength bin (using the correlated-k method), and
2. Add this data to the `FJX_spec.dat` file.

For the first step, you can use tools already available on the Prather research group website. To generate the cross-sections used by Fast-JX, download the file UCI_fastJ_addX_73cx.tar.gz. You can then simply add your data to `FJX_spec.dat` and refer to it in `FJX_j2j.dat` as specified above. The following then describes how to generate a new set of cross-section data for the example of some new species MEKR:

To generate the photolysis cross sections of a new species, come up with some unique name which you will use to refer to it in the `FJX_j2j.dat` and `FJX_spec.dat` files - e.g. MEKR. You will need to copy one of the `addX_*.f` routines and make your own (say, `addX_MEKR.f`). Your edited version will need to read in whatever cross section data you have available, and you'll need to decide how to handle out-of-range information - this is particularly crucial if your cross section data is not defined in the visible wavelengths, as there have been some nasty problems in the past caused by implicitly assuming that the XS can be extrapolated (I would recommend buffering your data with zero values at the exact limits of your data as a conservative first guess). Then you need to compile that as a standalone code and run it; this will spit out a file fragment containing the aggregated 18-bin cross sections, based on a combination of your measured/calculated XS data and the non-contiguous bin subranges used by Fast-JX. Once that data has been generated, just add it to `FJX_spec.dat` and refer to it as above. There are examples in the addX files of how to deal with variations of cross section with temperature or pressure, but the main takeaway is that you will generate multiple cross section entries to be added to `FJX_spec.dat` with the same name.

Important: If your cross section data varies as a function of temperature AND pressure, you need to do something a little different. The acetone XS documentation shows one possible way to handle this; Fast-JX currently interpolates over either T or P, but not both, so if your data varies over both simultaneously then this will take some thought. The general idea seems to be that one determines which dependence is more important and uses that to generate a set of 3 cross sections (for interpolation), assuming values for the unused variable based on the standard atmosphere.

CHAPTER

TEN

ADDING PRODUCTION AND LOSS FAMILIES

Certain common families (e.g. POx, LOx) have been pre-defined for you. You will find the family definitions near the top of the `gckpp.kpp` file:

```
#FAMILIES
POx : O3 + NO2 + 2NO3 + PAN + PPN + MPAN + HNO4 + 3N2O5 + HNO3 + BrO + HOBr + BrNO2 +␣
↪2BrNO3 + MPN + ETHLN + MVKN + MCRHN + MCRHNB + PROPNN + R4N2 + PRN1 + PRPN + R4N1 +␣
↪HONIT + MONITS + MONITU + OLND + OLNN + IHN1 + IHN2 + IHN3 + IHN4 + INPB + INPD +␣
↪ICN + 2IDN + ITCN + ITHN + ISOPNOO1 + ISOPNOO2 + INO2B + INO2D + INA + IDHNBOO +␣
↪IDHNDOO1 + IDHNDOO2 + IHPNBOO + IHPNDOO + ICNOO + 2IDNOO + MACRNO2 + ClO + HOCl +␣
↪ClNO2 + 2ClNO3 + 2Cl2O2 + 2OClO + O + O1D + IO + HOI + IONO + 2IONO2 + 2OIO + 2I2O2␣
↪+ 3I2O3 + 4I2O4;
LOx : O3 + NO2 + 2NO3 + PAN + PPN + MPAN + HNO4 + 3N2O5 + HNO3 + BrO + HOBr + BrNO2 +␣
↪2BrNO3 + MPN + ETHLN + MVKN + MCRHN + MCRHNB + PROPNN + R4N2 + PRN1 + PRPN + R4N1 +␣
↪HONIT + MONITS + MONITU + OLND + OLNN + IHN1 + IHN2 + IHN3 + IHN4 + INPB + INPD +␣
↪ICN + 2IDN + ITCN + ITHN + ISOPNOO1 + ISOPNOO2 + INO2B + INO2D + INA + IDHNBOO +␣
↪IDHNDOO1 + IDHNDOO2 + IHPNBOO + IHPNDOO + ICNOO + 2IDNOO + MACRNO2 + ClO + HOCl +␣
↪ClNO2 + 2ClNO3 + 2Cl2O2 + 2OClO + O + O1D + IO + HOI + IONO + 2IONO2 + 2OIO + 2I2O2␣
↪+ 3I2O3 + 4I2O4;
PCO : CO;
LCO : CO;
PSO4 : SO4;
LCH4 : CH4;
PH2O2 : H2O2;
```

Note: The POx, LOx, PCO, and LCO families are used for computing budgets in the GEOS-Chem benchmark simulations. PSO4 is required for simulations using TOMAS aerosol microphysics.

To add a new prod/loss family, add a new line to the `#FAMILIES` section with the format

```
FAM_NAME : MEMBER_1 + MEMBER_2 + ... + MEMBER_N;
```

The family name must start with `P` or `L` to indicate whether KPP should calculate a production or a loss rate.

The maximum number of families allowed by KPP is currently set to 300. Depending on how many prod/loss families you add, you may need to increase that to a larger number to avoid errors in KPP. You can change the number for `MAX_FAMILIES` in `KPP/kpp-code/src/gdata.h` and then rebuild KPP.

```
#define MAX_EQN         1500    /* KPP 2.3.0_gc, Bob Yantosca (11 Feb 2021)   */
#define MAX_SPECIES     1000    /* KPP 2.3.0_gc, Bob Yantosca (11 Feb 2021)   */
#define MAX_SPNAME        30
#define MAX_IVAL          40
#define MAX_EQNTAG        12    /* Max length of equation ID in eqn file      */
```

(continues on next page)

(continued from previous page)

```
#define MAX_K            150    /* Max length of rate expression in eqn file */
#define MAX_ATOMS         10
#define MAX_ATNAME        10
#define MAX_ATNR         250
#define MAX_PATH         120
#define MAX_FILES         20
#define MAX_FAMILIES     300
#define MAX_MEMBERS      150
#define MAX_EQNLEN       200
```

Important: When adding a prod/loss family or changing any of the other settings in `gckpp.kpp`, you must re-run KPP to produce new Fortran-90 files for GEOS-Chem (*as described in a previous chapter*).

Production and loss families are archived via the HISTORY diagnostics. For more information, please see the Guide to GEOS_Chem History diagnostics on the GEOS-Chem wiki.

CHAPTER

ELEVEN

CHANGING THE NUMERICAL INTEGRATOR

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

```
#INTEGRATOR rosenbrock
#LANGUAGE Fortran90
#DRIVER none
#HESSIAN off
#MEX off
#STOICMAT off
```

The `#INTEGRATOR` tag specifies the choice of numerical integrator that you wish to use with your chemical mechanism. The Rosenbrock solver is used by default.

Important: We do not recommend changing the value of `#INTEGRATOR`.

However, if you wish to use a different integrator for research purposes, you may select from one of the following options:

- exponential
- gillespie
- kpp_dvode
- kpp_lsode
- kpp_radau5
- kpp_sdirk4
- kpp_seulex
- none
- rosenbrock
- rosenbrock_adj
- rosenbrock_split
- rosenbrock_tlm
- runge_kutta
- runge_kutta_adj
- runge_kutta_tlm
- sdirk

- sdirk_adj
- sdirk_tlm
- tau_leap

The `#LANGUAGE` setting should be set to `Fortran90`.

The other options should be left as they are, as they are not relevant to **GEOS-Chem**.

For more information about **KPP** settings, please see the KPP 2.1 user manual.

CHAPTER

TWELVE

KNOWN BUGS

This page links to known bugs in **KPP-for-GEOS-Chem**. See the GitHub issues for updates on their status.

Version 2.3.1_gc:

- https://github.com/geoschem/KPP/issues/1

CHAPTER

THIRTEEN

SUPPORT GUIDELINES

GEOS-Chem support is maintained by the GEOS-Chem Support Team (GCST). The GCST members are based at Harvard University and Washington University in St. Louis.

We track bugs, user questions, and feature requests through GitHub issues. Please help out as you can in response to issues and user questions.

13.1 How to report a bug

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

13.2 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 "ask a question" template.

13.3 How to submit changes

Please see "Contributing Guidelines".

13.4 How to request an enhancement

Please see "Contributing Guidelines".

CHAPTER FOURTEEN

CONTRIBUTING GUIDELINES

Thank you for looking into contributing to GEOS-Chem! GEOS-Chem is a grass-roots model that relies on contributions from community members like you. Whether you're new to GEOS-Chem or a longtime user, you're a valued member of the community, and we want you to feel empowered to contribute.

14.1 We use GitHub and ReadTheDocs

We use GitHub to host the KPP-for-GEOS-Chem source code, to track issues, user questions, and feature requests, and to accept pull requests: https://github.com/geoschem/KPP. Please help out as you can in response to issues and user questions.

We use ReadTheDocs to host the KPP-for-GEOS-Chem user documentation: https://kpp.readthedocs.io.

14.2 How to submit changes

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

As the author you are responsible for:

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

14.3 Coding conventions

The GEOS-Chem codebase dates back several decades and includes contributions from many people and multiple organizations. Therefore, some inconsistent conventions are inevitable, but we ask that you do your best to be consistent with nearby code.

14.4 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.

14.5 How to report a bug

Please see "Support Guidelines".

14.6 Where can I ask for help?

Please see "Support Guidelines".

CHAPTER

FIFTEEN

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.

15.1 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`.

15.2 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.

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

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

[Long et al., 2015] Long, M. S., Yantosca, R., Nielsen, J. E., Keller, C. A., da Silva, A., Sulprizio, M. P., Pawson, S., and Jacob, D. J. Development of a grid-independent GEOS-Chem chemical transport model (v9-02) as an atmospheric chemistry module for Earth system models. *Geoscientific Model Development*, 8(3):595–602, March 2015. doi:10.5194/gmd-8-595-2015.

[Eastham et al., 2018] Eastham, S. D., Long, M. S., Keller, C. A., Lundgren, E., Yantosca, R. M., Zhuang, J., Li, C., Lee, C. J., Yannetti, M., Auer, B. M., Clune, T. L., Kouatchou, J., Putman, W. M., Thompson, M. A., Trayanov, A. L., Molod, A. M., Martin, R. V., and Jacob, D. J. GEOS-Chem High Performance (GCHP v11-02c): a next-generation implementation of the GEOS-Chem chemical transport model for massively parallel applications. *Geoscientific Model Development*, 11(7):2941–2953, July 2018. doi:10.5194/gmd-11-2941-2018.

[Zhuang et al., 2020] Zhuang, J., Jacob, D. J., Lin, H., Lundgren, E. W., Yantosca, R. M., Gaya, J. F., Sulprizio, M. P., and Eastham, S. D. Enabling High-Performance Cloud Computing for Earth Science Modeling on Over a Thousand Cores: Application to the GEOS-Chem Atmospheric Chemistry Model. *Journal of Advances in Modeling Earth Systems*, May 2020. doi:10.1029/2020MS002064.

[Bindle et al., 2020] Bindle, L., Martin, R. V., Cooper, M. J., Lundgren, E. W., Eastham, S. D., Auer, B. M., Clune, T. L., Weng, H., Lin, J., Murray, L. T., Meng, J., Keller, C. A., Pawson, S., and Jacob, D. J. Grid-Stretching Capability for the GEOS-Chem 13.0.0 Atmospheric Chemistry Model. *Geoscientific Model Development*, December 2020. doi:10.5194/gmd-2020-398.

INDEX