AtChem2 v1.2

User Manual

R. Sommariva
S. Cox

December 29, 2019

Contents

Chapter 1

Introduction

AtChem2 is a modelling software designed to build and run atmospheric chemistry box-models using the Master Chemical Mechanism (MCM). It can also be used with other chemical mechanisms, as long as they are provided in the correct format (see Chemical Mechanism).

AtChem2 was developed from **AtChem-online** with the objective to create a software able to run large atmospheric chemistry models. AtChem-online is a web tool developed at the University of Leeds as part of the EUROCHAMP project. It was designed to facilitate the use of the MCM in the simulation of environmental chamber experiments. A tutorial for AtChem-online, with examples and exercises, is available on the MCM website. A help page with detailed instructions and description of the model parameters and variables is available here.

The latest stable version of AtChem2 can be found at the releases page. The development version can be downloaded from the master branch or obtained via **git**. To install and run AtChem2 follow the instructions in the Installation and Dependencies sections.

AtChem2 is open source, under **MIT license**. For information on how to cite the model in publications, see the `CITATION.md` file. The contributors and funders of **AtChem-online** and **AtChem2** are listed in the Acknowledgements and Credits chapter.

Bug reports, suggestions and contributions are welcome. In order to contribute to the model development, please follow the instructions in the Model Development chapter.

Chapter 2

Model Installation

2.1 Requirements

AtChem2 can be installed on Linux/Unix and macOS operating systems. A working knowledge of the **unix shell** and its basic commands is *required* to install and use the model. AtChem2 requires the following tools:

- Fortran – the model compiles with GNU `gfortran` (version 4.8.5) and with Intel `ifort` (version 17.0)
  default compiler: GNU `gfortran`
- Python
- cmake
- Ruby (optional)

Some or all of these tools may already be present on your system. Use the `which` command to find out (e.g.: `which python`, `which cmake`, etc...). Otherwise, check the local documentation or ask the system administrator. In addition, AtChem2 has the following dependencies:

- CVODE
- openlibm
- BLAS and LAPACK
- numdiff (optional)
- FRUIT (optional)

For detailed instructions on the installation and configuration of the dependencies go to Sect. 2.3.

2.2 Download

There are two versions of AtChem2: the stable version and the development version, also known as `master branch` (see Chapt. 5 for additional information). The source code can be obtained in two ways:

- with **git**:
  1. Open the terminal. Move to the directory where you want to install AtChem2.
  2. Execute `git clone https://github.com/AtChem/AtChem2.git` (if using HTTPS) or `git clone git@github.com:AtChem/AtChem2.git` (if using SSH). This method will download the development version and it is recommended if you want to contribute to the model development.
- with the **archive file**:
  1. Download the archive file (`*.tar.gz` or `*.zip`) of the stable version or of the development version to the directory where you want to install AtChem2.
  2. Open the terminal. Move to the directory where you downloaded the archive file.
  3. Execute `tar -zxfv v1.2.tar.gz` or `unzip -v master.zip` to unpack the archive file.

The downloaded source code is now in a directory called `AtChem2` (if git was used) or `AtChem2-1.2` (if the stable version was downloaded) or `AtChem2-master` (if the development version was downloaded). This directory, which can be renamed as you prefer, is the *`Main Directory`* of the model. In this manual we will assume that the *`Main Directory`* is `$HOME/AtChem2`.

2.3 Dependencies

AtChem2 has a number of dependencies (external tools and libraries): some are required, and without them the model cannot be installed or used, others are optional. It is recommended to use the same directory for all the dependencies of AtChem2: the *`Dependency Directory`* can be located anywhere and called as you prefer. In this manual we will assume that the *`Dependency Directory`* is `$HOME/atchem-lib/`.

Before installing the dependencies, make sure that a **Fortran** compiler, **Python**, **cmake** and (optionally) **Ruby** are installed on your system, as explained in Sect. 2.1.

2.3.1 Required dependencies

BLAS and LAPACK

BLAS and LAPACK are standard Fortran libraries for linear algebra. They are needed to install and compile the CVODE library (see below). Usually they are located in the `/usr/lib/` directory (e.g., `/usr/lib/libblas/` and `/usr/lib/lapack/`). The location may be different, especially if you are on a High Performance Computing (HPC) system, so check the local documentation or ask the system administrator.

CVODE

AtChem2 uses the CVODE library which is part of the open source SUNDIALS suite [Hindmarsh et al., 2005], to solve the system of ordinary differential equations (ODE).

The version of CVODE currently used in AtChem2 is v2.9.0 (part of SUNDIALS v2.7.0) and it can be installed using the `install_cvode.sh` script in the `tools/install/` directory.

1. Open the terminal. Move to the *Main Directory* (`cd ~/AtChem2`).
2. Open the installation script (`tools/install/install_cvode.sh`) with a text editor:
   - If LAPACK and BLAS are not in the default location on your system (see above), change the `LAPACK_LIBS` variable for your architecture (Linux or macOS) as appropriate.
   - If you are not using the `gcc` compiler (`gfortran`), change the line `-DCMAKE_C_COMPILER:FILEPATH=gcc \` accordingly.
3. Run the installation script (change the path to the *Dependency Directory* as needed):

```
./tools/install/install_cvode.sh ~/atchem-lib/
```

If the installation is successful, there should be a working CVODE installation at `~/atchem-lib/cvode/`. Take note of the path to the CVODE library (`~/atchem-lib/cvode/lib/`), as it will be needed later to complete the configuration of AtChem2 (see Sect. 2.4).

openlibm

openlibm is a portable version of the open source libm library. Installing openlibm and linking against it allows reproducible results by ensuring the same implementation of several mathematical functions across platforms.

The current version of openlibm is 0.4.1 and it can be installed using the `install_openlibm.sh` script in the `tools/install/` directory.

1. Open the terminal. Move to the *Main Directory* (`cd ~/AtChem2`).
2. Run the installation script (change the path to the *Dependency Directory* as needed):

```
./tools/install/install_openlibm.sh ~/atchem-lib/
```

If the installation is successful, there should be a working openlibm installation at `~/atchem-lib/openlibm-0.4.1/`. Take note of the path to openlibm, as it will be needed later to complete the configuration of AtChem2 (see Sect. 2.4).

2.3.2 Optional dependencies

numdiff

numdiff is a program used to compare files containing numerical fields. It is needed only if you want to run the Test Suite, a series of tests used to ensure that the model works properly and that changes to the code do not result in unintended behaviour. Installation of numdiff is recommended if you want to contribute to the development of AtChem2.

Use `which numdiff` to check if the program is already installed on your system. If not, ask the system administrator. Alternatively, numdiff can be installed locally (e.g., in the *Dependency Directory*) using the script `install_numdiff.sh` in the `tools/install/` directory.

1. Open the terminal. Move to the *Main Directory* (`cd ~/AtChem2`).
2. Run the installation script (change the path to the *Dependency Directory* as needed):

```
./tools/install/install_numdiff.sh ~/atchem-lib/
```

3. Move to your `$HOME` directory (`cd ~`). Open the `.bash_profile` file (or the `.profile` file, depending on your configuration) with a text editor. Add the following line at the bottom of the file (change the path to the *Dependency Directory* as needed):

```
PATH=$PATH:$HOME/atchem-lib/numdiff/bin
```

4. Close the terminal.
5. Reopen the terminal. Execute `which numdiff` to check that the program has been installed correctly.

FRUIT

FRUIT (FORTRAN Unit Test Framework) is a unit test framework for Fortran. It requires Ruby and is needed only if you want to run the unit tests (see Sect. 5.2). Installation of FRUIT is recommended if you want to contribute to the development of AtChem2.

The current version of FRUIT is 3.4.3 and it can be installed using the `install_fruit.sh` script in the `tools/install/` directory.

1. Open the terminal. Move to your `$HOME` directory (`cd ~`). Open the `.bash_profile` file (or the `.profile` file, depending on your configuration) with a text editor. Add the following lines at the bottom of the file:

   ```
   GEM_HOME=$HOME/.gem
   PATH=$PATH:$GEM_HOME/bin
   ```

2. Close the terminal.

3. Reopen the terminal. Move to the *Main Directory* (`cd ~/AtChem2`).

4. Run the installation script (change the path to the *Dependency Directory* as needed):

   ```
   ./tools/install/install_fruit.sh ~/atchem-lib/
   ```

If the installation is successful, there should be a working FRUIT installation at `~/atchem-lib/fruit_3.4.3/`. Take note of the path to FRUIT, as it will be needed later to complete the configuration of AtChem2 (see Sect. 2.4).

2.4 Install

To install AtChem2:

1. Open the terminal. Move to the *Main Directory* (`cd ~/AtChem2`). Install the Dependencies and take note of the names and paths of **CVODE**, **openlibm** and **FRUIT**.

2. Copy the example `Makefile` from the `tools/install/` directory to the *Main Directory*:

   ```
   cp ./tools/install/Makefile.skel ./Makefile
   ```

3. Open the `Makefile` with a text editor. Set the variables `$CVODELIB`, `$OPENLIBMDIR`, `$FRUITDIR` to the paths of CVODE, openlibm and FRUIT, respectively (see Sect. 2.3). Use full paths, as using relative paths may cause compilation errors (see issue #364). For example (change the path to the *Dependency Directory* as needed):

   ```
   CVODELIB = $(HOME)/atchem-lib/cvode/lib
   OPENLIBMDIR = $(HOME)/atchem-lib/openlibm-0.4.1
   FRUITDIR = $(HOME)/atchem-lib/fruit_3.4.3
   ```

   If FRUIT has not been installed, you can leave the default value for the `$FRUITDIR` variable.

4. Compile AtChem2 with the `build_atchem2.sh` script in the `build/` directory:

   ```
   ./build/build_atchem2.sh ./mcm/mechanism_test.fac
   ```

   This command compiles the model and creates an executable (`atchem2`) using the example mechanism file `mechanism_test.fac` in the `mcm/` directory.

5. Execute `./atchem2`. This command runs the model executable using the default configuration. If the model run completes successfully you should see a message similar to this:

   ```
   ------------------
    Final statistics
   ------------------
    No. steps = 546   No. f-s = 584   No. J-s = 912   No. LU-s = 56
    No. nonlinear iterations = 581
    No. nonlinear convergence failures = 0
    No. error test failures = 4

    Runtime = 0
    Deallocating memory.
   ```

If your are installing AtChem2 on a macOS system and receive an error message concerning `rpath`, check the **Note for macOS users** on the wiki.

AtChem2 is now ready to be used. Optionally, the Test Suite can be run to check that the model has been installed correctly (see Sect. 2.4.1). The directory structure and the organization of the AtChem2 model are described in Sect. 2.5. For information on how to compile, configure and execute an AtChem2 model go to Chapt. 3 and Chapt. 4.

2.4.1 Tests (optional)

The Test Suite can be used to verify that AtChem2 has been installed correctly and works as intended. It is recommended to run the Test Suite if you want to contribute to the development of the AtChem2 model. Note that in order to run the Test Suite the optional dependencies have to be installed.

To run the Test Suite, open the terminal and execute one of the following commands from the *Main Directory*:

- `make alltests`: runs all the tests (requires numdiff and FRUIT).
- `make tests`: runs only the build and behaviour tests (requires numdiff).
- `make unittests`: runs only the unit tests (requires FRUIT).

The command executes the requested tests, then prints to the terminal the tests output and a summary of the results.

2.5 Model Structure

AtChem2 is organized in several directories which contain the source code, the compilation files, the chemical mechanism, the model configuration and output, several scripts and utilities, and the Test Suite. The directory structure of AtChem2 is derived from that of AtChem-online, but it was substantially changed with the release of version 1.1 (November 2018). Table 2.1 shows the new directory structure and, for reference, the original one.

In AtChem2 v1.1 (and later versions) the directories `build/`, `mcm/`, `obj/` and `src/` contain, respectively, the compilation scripts, the MCM data files, the files generated by the compiler and the source code (see Sect. 4.3 for detailed information); the directory `travis/` contains the files and the scripts necessary to run the Test Suite.

For the majority of the users, the most important directories are `doc/`, which contains this manual and other documents, `tools/`, which contains the installation and plotting scripts (plus other utilities), and `model/`, which contains the model information (configuration, input, output and, usually, the chemical mechanism). For more information on these three directories, see the following sections.

2.5.1 The doc/ and tools/ directories

The `doc/` directory contains the pdf file of the AtChem2 user manual (this document: `AtChem2-Manual.pdf`), along with the corresponding LaTeX files and figures. An electronic copy of the poster presented at the 2018 Atmospheric Chemical Mechanisms Conference [Sommariva et al., 2018] is also included (`AtChem_poster_ACM2018.pdf`).

Table 2.1: Directory structure of AtChem2. "Original" refers to version 1.0 and earlier; "New" refers to version 1.1 and later.

Original	New (from v1.1)	Description
–	`build/`	scripts to build the model.
–	`doc/`	user manual and other documents.
–	`mcm/`	MCM data files and example `.fac` files.
`modelConfiguration/`	`model/configuration/`	chemical mechanism, shared library, model and solver configuration files.
`speciesConstraints/`	`model/constraints/species/`	model constraints: chemical species.
`environmentConstraints/`	`model/constraints/environment/`	model constraints: environment variables.
`environmentConstraints/`	`model/constraints/photolysis/`	model constraints: photolysis rates.
`modelOutput/`	`model/output/`	model output: chemical species, photolysis rates, environment variables, diagnostic variables.
`instantaneousRates/`	`model/output/reactionRates/`	model output: reaction rates for every reaction of the chemical mechanism.
`obj/`	`obj/`	files generated by the Fortran compiler.
`src/`	`src/`	Fortran source files.
`tools/`	`tools/`	various scripts and plotting tools.
`travis/`	`travis/`	scripts and files for the Test Suite.

The `tools/` directory contains some auxiliary scripts (e.g., `version.sh` which is used to update the version number of the model in each development cycle) and the following subdirectories:

- `install/`, which contains the example `Makefile` and the scripts to install the Dependencies.
- `plot/`, which contains basic plotting scripts in various programming languages.

The plotting scripts in `tools/plot/` are very simple and are only intended to give the user a quick overview of the model output for validation and diagnostic purposes. We suggest to use a proper data analysis software package (e.g., IDL, Igor, MATLAB, Origin, R, etc. . .) to process and analyze the model results.

The plotting scripts are written in different programming languages: gnuplot, Octave [1], Python (v2 and v3), R. One or more of these environments

[1]GNU Octave is an open source implementation of MATLAB. The script `plot-atchem2.m` is compatible with both Octave and MATLAB

is probably already installed on your system: check the local documentation or ask the system administrator. All plotting scripts require one argument – the directory containing the model output – and create one file called `atchem2_output.pdf` in the given directory. By default, the model output directory is `model/output/`, but see Sect. 2.5.2 for details.

To run a plotting script, open the terminal and execute one of the following commands from the *Main Directory* (change the path to the model output directory as needed):

- `gnuplot -c tools/plot/plot-atchem2.gp model/output/`
- `octave tools/plot/plot-atchem2.m model/output/`
- `python2 tools/plot/plot-atchem2_v2.py model/output/`
- `python3 tools/plot/plot-atchem2_v3.py model/output/`
- `Rscript --vanilla tools/plot/plot-atchem2.r model/output/`

2.5.2 The model/ directory

The `model/` directory is the most important for the user: it includes the model configuration files, the model constraints and the model output. Basically, all the information required to set up and run a box-model with AtChem2, together with the model results, is contained in this directory. In principle, the chemical mechanism (`.fac` file) could be located in another directory, although it is good practice to keep it together with the rest of the model configuration (see Sect. 3.1.4).

The `model/` directory can be given any name and can be located outside the *Main Directory*. Moreover, there can be multiple `model/` directories (with different names) in the same location. The paths to the required `model/` directory and/or to the chemical mechanism file are given as an argument to the build script (`build/build_atchem2.sh`) and to the `atchem2` executable, as explained in Sect. 4.3.

This approach gives the user the flexibility to run different versions of the same model (in terms of configuration and/or chemical mechanism) or different models (e.g., for separate projects) at the same time, without having to recompile the source code and create a different executable each time. Sensitivity studies and batch model runs are therefore easy to do, since all the parts of the model than need to be modified are contained in the same directory. For more information go to Chapt. 4.

Chapter 3

Model Setup

3.1 Chemical Mechanism

The chemical mechanism is the core element of an atmospheric chemistry model. In AtChem2, the mechanism file is written in FACSIMILE format and has the extension `.fac`. The FACSIMILE format is used to describe chemical reactions in the commercial FACSIMILE Kinetic Modelling Software; for historical reasons, the software and the format have been often used in conjunction with the MCM. The extraction tool on the MCM website can generate `.fac` files directly in FACSIMILE format (see Sect. 3.1.3).

3.1.1 The FACSIMILE format

Chemical reactions are described in FACSIMILE format using the following notation:

```
% k : A + B = C + D ;
```

where `k` is the rate coefficient, `A` and `B` are the reactants, `C` and `D` are the products. A reaction starts with the `%` character and ends with the `;` character. Comments can be inserted in the `.fac` file to document and annotate the chemical mechanism: in FACSIMILE format, comments are enclosed between the characters `*` and `;` and are ignored by the build scripts.

The rate coefficient (`k`) can be a constant number or, more commonly, can be calculated as a function of other variables, such as temperature (`TEMP`), air density (`M`), water vapour (`H2O`) and other environment variables (Sect. 3.4). A basic chemical mechanism, with comments and calculated rate coefficients, looks like this:

```
* Tropospheric O3-NOx cycle ;
* Kinetic data from Atkinson et al., ACP, 2004 ;
% J_NO2                      : NO2 = NO + O ;
```

```
% 5.6D-34*M*(TEMP/300)@-2.6  : O + O2 = O3 ;
% 1.4D-12*EXP(-1310/TEMP)    : NO + O3 = NO2 + O2 ;
```

The photolysis rate of NO_2 (J_NO2) in the example above is calculated by AtChem2 as function of latitude, longitude and solar zenith angle, as explained in detail in Sect. 3.5. Complex mathematical expressions can be used to calculate the rate coefficient, in which case they have to be defined before the chemical reactions that use them (typically, combination and dissociation reactions). For example:

```
* Formation of nitric acid (HNO3) in the gas-phase ;
*;
* Rate coefficient (Atkinson et al., ACP, 2004) ;
K80 = 3.3D-30*M*(TEMP/300)@-3.0 ;
K8I = 4.1D-11 ;
KR8 = K80/K8I ;
FC8 = 0.4 ;
NC8 = 0.75-1.27*(LOG10(FC8)) ;
F8 = 10@(LOG10(FC8)/(1+(LOG10(KR8)/NC8)**2)) ;
KMT08 = (K80*K8I)*F8/(K80+K8I) ;
*;
* Chemical Reaction ;
% KMT08 : OH + NO2 = HNO3 ;
```

Chemical reactions can be written in FACSIMILE format without reactants or products. This feature is useful to implement simple descriptions of non-chemical processes in a box-model. For example, dry deposition to the surface and direct emission of a chemical species can be parametrized as:

```
* Deposition velocity of O3 = 1.4 cm s-1 ;
% 1.4/BLHEIGHT  :  O3 =  ;

* Emission rate of NO2 = 1e8 molecule cm-3 s-1 ;
% 1D+8  :  = NO2  ;
```

More sophisticated approaches to describe non-chemical processes can be implemented either by defining complex mathematical expressions to calculate the corresponding "rate coefficients" (as shown above) or by writing the appropriate Fortran function(s) into the source code.

The `.fac` file is processed by a Python script (`mech_converter.py`, see Sect. 3.1.4); the script expects the chemical mechanism to have four sections:

Generic rate coefficients : contains the definitions of the generic rate coefficients used when experimental kinetic data are not available.

Complex reactions : contains the mathematical expressions to calculate complex rate coefficients (e.g., for combination and dissociation reactions).

Peroxy radicals : contains the calculation of the RO_2 sum – go to Sect. 3.1.2 for details.

Reaction definitions : contains the chemical reactions (and parametrized non-chemical processes) in FACSIMILE format.

For `mech_converter.py` to work, the beginning of each section must be delimited by a header constituted by a single comment line. The header must always be present, even though the corresponding sections can be empty. A minimal `.fac` file (`mechanism_skel.fac`) and an example chemical mechanism downloaded from the MCM website (`mechanism_test.fac`) are included in the `mcm/` directory for reference and testing.

3.1.2 The RO_2 sum

The sum of organic peroxy radicals (RO_2) is a key feature of the Master Chemical Mechanism. Organic peroxy radicals react with HO_2, with themselves and with other RO_2: given that there are over 1000 peroxy radicals in the MCM, the number of possible self and cross reactions is of the order of 10^6, which presents a significant computational challenge. The RO_2 sum is used in the MCM to keep the number of peroxy radicals permutation reactions to a reasonable number, as explained in the MCM protocol papers [Jenkin et al., 1997, Saunders et al., 2003].

AtChem2 is designed primarily to run models based upon the MCM, and therefore the `.fac` file must contain a section with the calculation of the RO_2 sum (Sect. 3.1.1). This section must have a header and has the following format:

```
* Peroxy radicals. ;
RO2 = RO2a + RO2b + RO2c + ... ;
```

where `RO2a`, `RO2b`, `RO2c`, are the organic peroxy radicals in the chemical mechanism. If there are no organic peroxy radicals in the chemical mechanism (or if the mechanism is not based upon the MCM), the RO_2 sum section must still be present in the `.fac` file, but it is left empty:

```
* Peroxy radicals. ;
RO2 = ;
```

The RO_2 sum is automatically generated from the chemical mechanism during the build process using the list of RO_2 extracted from the MCM database (Sect. 3.1.4). AtChem2 includes the complete list of all organic

peroxy radicals in the MCM v3.3.1 (`mcm/peroxy-radicals_v3.3.1`), which is the version used by default. Complete lists of all organic peroxy radicals in the other versions of the MCM are also included in the `mcm/` directory. Instructions on how to set up AtChem2 to use the previous versions of the MCM can be found in the file `mcm/INFO.md`. The value of the RO_2 sum is always output together with the environment variables (in `model/output/environmentVariables.output`).

It is important to ensure that the RO_2 sum is accurate, because many reactions in the MCM use the value of this parameter to calculate the reaction rate correctly. The hydroperoxyl radical (HO_2) is a peroxy radical but is not an organic molecule, and therefore it *should not* be included in the RO_2 sum.

3.1.3 MCM extraction

The MCM website provides a convenient tool that can be used to download the whole Master Chemical Mechanism (or subsets of it) in FACSIMILE format. Only a brief overview of the process will be given here: for more information go to the MCM website.

First, select the species of interest using the MCM browser and add the selection to the *Mark List*. Then proceed to the MCM extraction tool and select the option *FACSIMILE input format, suitable for inserting into a FACSIMILE model.*. Make sure that the following options are selected, so that the required headers (Sect. 3.1.1) will be included in the generated `.fac` file: :

```
[x] Include inorganic reactions?
[x] Include generic rate coefficients?
    FACSIMILE, FORTRAN and KPP formats only
```

Click on *Extract* to download the `.fac` file into a directory of choice – such as the `model/` directory, as discussed in Sect. 2.5. The downloaded `.fac` file is a simple text file and can be used directly in AtChem2. If modifications are required (e.g., if you want to add, delete, or change some chemical reactions) open the `.fac` file with a text editor and edit the chemical mechanism as needed.

3.1.4 The build process

AtChem2 is built using the scripts in the `build/` directory. Here, we only outline the build process; detailed instructions on how to compile the model can be found in Sect. 4.3.

The Python script `mech_converter.py`, which is automatically called by the `build_atchem2.sh` script, converts the chemical mechanism from the

FACSIMILE fortmat into a format that can be read by the Fortran code. In doing so, the Python script generates a number of files:

- `mechanism.f90` contains the equations, in Fortran code, to calculate the rate coefficients of each reaction of the chemical mechanism.
- `mechanism.so` is the shared library, i.e. the pre-compiled version of the chemical mechanism.
- `mechanism.species` contains the list of chemical species in the chemical mechanism. The file has no header. The first column is the *ID number* of the species, the second column is the name of the species:

```
1 O
2 O3
3 NO
4 NO2
```

- `mechanism.reac` and `mechanism.prod` contain the reactants and the products (respectively) in each reaction of the chemical mechanism. The files have a one line header with the number of species, the number of reactions and the number of equations in the *Generic rate coefficients* and *Complex reactions* sections (Sect. 3.1.1). The first column is the *ID number* of the reaction, the second column is the *ID number* of the chemical species which are reactants/products in that reaction:

```
29 71 139 numberOfSpecies numberOfReactions numberOfGenericComplex
1 1
2 1
3 1
3 2
```

- `mechanism.ro2` contains the organic peroxy radicals (RO_2). The file has a one line header formatted as a Fortran comment. The first column is the *ID number* of the peroxy radical (from `mechanism.species`), the second column is the name of the peroxy radical as a Fortran comment:

```
! Note that this file is generated by build/mech_converter.py
! based upon the file mcm/mechanism_test.fac
! Any manual edits to this file will be overwritten when
! calling build/mech_converter.py
23 !CH3O2
26 !C2H5O2
```

```
28 !IC3H7O2
29 !NC3H7O2
```

The directory containing the files generated by the build script is the *Shared Library Directory* which, by default, is `model/configuration/`: the location of the *Shared Library Directory* can be changed using the second argument of the build script, as explained in more detail in Sect. 4.3 (see also Sect. 2.5.2).

3.2 Model Parameters

The model parameters control the general setup of the model; they are set in the `model.parameters` file which, by default, is in the `model/configuration/` directory.

- **number of steps** and **step size**. The duration of the model run is determined by the number of steps and by the step size (in seconds). The step size controls the frequency of the model output for the chemical species listed in `outputSpecies.config` (Sect. 3.6), as well as for all the environment variables, the photolysis rates and the diagnostic variables. The step size is not related to the integration step which is controlled by CVODE.
  For example, a model runtime of 2 hours, with output every 5 minutes, requires 24 steps with a step size of 300 seconds (24x300 = 7200 sec = 2 hours).

- **species interpolation method** and **conditions interpolation method**. Interpolation method used for the constrained chemical species, and for the constrained environment variables and the photolysis rates, respectively (Sect. 4.2). Two interpolation methods are currently implemented in AtChem2: piecewise constant and piecewise linear. The default option is `2` (piecewise linear interpolation).

- **rates output step size**. Frequency (in seconds) of the model output for the production and loss rates of selected chemical species, i.e. those listed in `outputRates.config` (Sect. 3.6).

- **model start time**. Start time of the model (in seconds) calculated from midnight of the **day**, **month**, **year** parameters (see below). For example, a start time of 3600 means the model run starts at 1:00 in the morning and a start time of 46800 means the model run starts at 1:00 in the afternoon (13:00). The **model stop time** is automatically calculated by the model as:

```
model start time + (number of steps * step size)
```

N.B.: when one or more variables are constrained, the time interval between the model start time and the model stop time *must be* equal to or less than the time interval of the constrained data (Sect. 4.2).

- **jacobian output step size**. Frequency (in seconds) of the model output for the Jacobian matrix. If this parameter is set to `0` (default option), the Jacobian matrix is not output. Note that the `jacobian.output` file generated by the model can be very large, especially if the chemical mechanism has many reactions and/or the model runtime is long.

- **latitude** and **longitude**. Geographical coordinates (in degrees). Latitude North is positive and latitude South is negative; longitude East is negative and longitude West is positive [1]. Latitude and longitude are used to calculate the Earth-Sun angles, which are needed for the calculation of the photolysis rates (Sect. 3.5).

- **day** and **month** and **year**. Start date of the model simulation. The model time is in UTC (GMT timezone) and is calculated in seconds since midnight of the start date.

- **reaction rates output step size**. Frequency (in seconds) of the model output for the reaction rates of every reaction in the chemical mechanism. By default, the reaction rates are saved in the directory `model/output/reactionRates/` as one file for each model step, with the name of the file corresponding to the time in seconds. In previous versions of AtChem, this output was called **instantaneous rates**. This parameter is different from **rates output step size** (see above), which sets the frequency of a formatted output of reaction rates for selected species of interest. For more information, go to Sect. 3.6.

3.3 Solver Parameters

The solver parameters control the behaviour of the ordinary differential equations (ODE) solver; they are set in the `solver.parameters` file which, by default, is in the `model/configuration/` directory. A complete explanation of these parameters can be found in the documentation of CVODE.

- **atol** (positive real) and **rtol** (positive real): absolute and relative tolerance values for the solver.

[1] The preferred convention is that longitude East is positive and longitude West is negative. The current version of AtChem2 uses the opposite convention to maintain compatibility with previous versions.

- **delta main** (positive real): linear convergence tolerance factor of the GMRES linear solver.
- **lookback** (positive integer): maximum Krylov subspace dimension of the GMRES linear solver.
- **maximum solver step size** (positive real): maximum size of the timesteps that the solver is allowed to use (in seconds).
- **maximum number of steps in solver** (positive integer): maximum number of steps used by the solver before reaching `tout`, i.e. the next output time.
- **solver type** (integer): selection of the linear solver to use: `1` for GMRES, `2` for GMRES preconditioned with a banded preconditioner (default option), `3` for a dense solver.
- **banded preconditioner upper bandwidth** (integer): only used in the case that `solver type = 2`.
- **banded preconditioner lower bandwidth** (integer): only used in the case that `solver type = 2`.

3.4 Environment Variables

The environment variables define the physical parameters of the model, such as temperature, pressure, humidity, latitude, longitude, position of the sun, etc.... These variables are set in the `environmentVariables.config` file which, by default, is in the `model/configuration/` directory.

The environment variables can have a fixed (constant) value or can be constrained to measured values (`CONSTRAINED`), in which case the corresponding data file must be present in the `model/constraints/environment/` directory (see Sect. 4.2 for more information). Some environment variables can be calculated by the model (`CALC`) and some can be deactivated if they are not needed (`NOTUSED`).

By default, the environment variables are set to `NOTUSED` or to a fixed value. The AtChem2 environment variables are described below, together with their possible settings, units, and default values.

3.4.1 TEMP

Ambient Temperature (K).

- fixed value
- constrained

Default fixed value = 298.15

3.4.2 PRESS

Ambient Pressure (mbar).

- fixed value
- constrained

Default fixed value = 1013.25

3.4.3 RH

Relative Humidity (%). It is required only if H2O is set to `CALC`, otherwise it should be set to `NOTUSED`.

- fixed value
- constrained
- not used

Default = NOTUSED (-1)

3.4.4 H2O

Water Concentration (molecules cm-3). If it is set to `CALC`, then RH has to be set to a fixed value or `CONSTRAINED`.

- fixed value
- constrained
- calculated

Default fixed value = 3.91e+17

3.4.5 DEC

Sun Declination (radians) is the angle between the center of the Sun and Earth's equatorial plane.

- fixed value
- constrained
- calculated

Default fixed value = 0.41

3.4.6 BLHEIGHT

Boundary Layer Height. It is required only if the model includes non-chemical processes, such as emission or deposition of chemical species. The unit is typically cm or m, depending on how these processes are parametrized in the chemical mechanism (see Sect. 3.1.1 for details).

- fixed value
- constrained
- not used

Default = NOTUSED (-1)

3.4.7 DILUTE

Dilution rate (s-1). It is required only if the model includes dilution of the chemical species.

- fixed value
- constrained
- not used

Default value = NOTUSED (-1)

3.4.8 JFAC

Correction factor used to adjust the photolysis rates – e.g., to account for the presence of clouds. `JFAC` can have a value between `0` (photolysis rates are set to zero) and `1` (photolysis rates are not corrected). JFAC is *not applied* to constant or constrained photolysis rates. For more information go to Sect. 3.5.

- fixed value
- constrained
- calculated

Default fixed value = 1

3.4.9 ROOF

Flag to switch the photolysis rates. It is used mostly in simulations of environmental chamber experiments, where the roof of the chamber can be opened/closed or the lights can be turned on/off.

When `ROOF` is set to `CLOSED` all the photolysis rates are set to zero, including those that are constant or constrained: this is different than setting `JFAC` to `0`, which only applies to the calculated photolysis rates (see above and Sect. 3.5). `ROOF` is the only environment variable that cannot be set to `CONSTRAINED`.

Default value = OPEN

3.5 Photolysis Rates

The photolysis rates are identified in FACSIMILE format with the notation `J<n>`, where `n` is an integer assigned by the MCM to each photolysis reaction (or class of reactions). By default, AtChem2 calculates the photolysis rates using the MCM parametrization, described below. Each photolysis rate can also be set to a constant value or to constrained values. The following rules apply (for more detail, see Fig. 3.1):

1. If a photolysis rate is set as constant, it assumes the given value. The other photolysis rates are set to zero.

2. If a photolysis rate is set to constrained, it assumes the values given in the corresponding constraint file. All other photolysis rates are calculated.

3. If no photolysis rate is set to constant or to constrained, the model calculates all photolysis rates.

Additionally, the environment variable `ROOF` (Sect. 3.4) can be used to turn the photolysis rates ON/OFF, which is useful for simulations of some environmental chamber experiments.

3.5.1 Constant photolysis rates

The typical scenario for constant photolysis rates is a lamp or solar simulator in an environmental chamber. All the photolysis rates used in the mechanism need to be given a value (in `model/configuration/photolysisConstant.config`) otherwise they are set to zero (Fig. 3.1). This approach allows the user to model individual photolysis processes and/or to account for lamps that emit only in certain spectral windows. The format of the `photolysisConstant.config` file is described in Sect. 3.6.

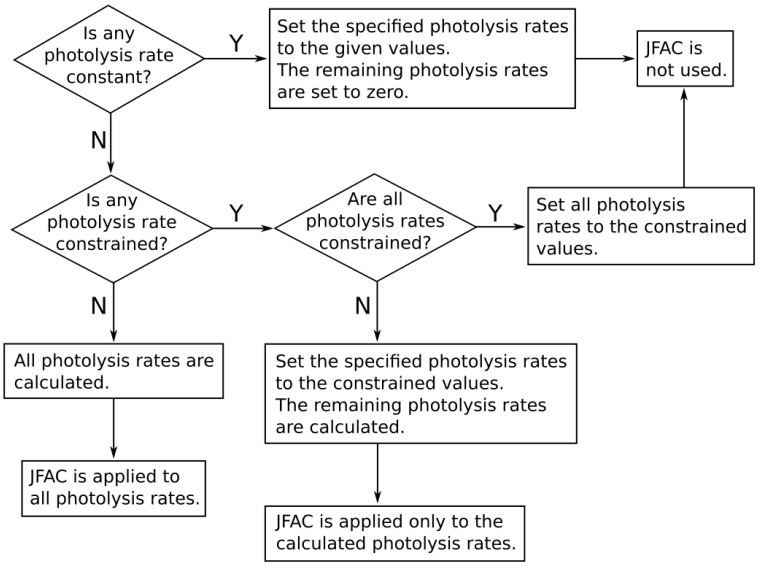

Figure 3.1: Treatment of photolysis rates in AtChem2.

3.5.2 Constrained photolysis rates

All photolysis rates can be constrained to measured values. In this case, the name of the constrained photolysis rate (e.g., `J2`) must be listed in the file `model/configuration/photolysisConstrained.config` (Sect. 3.6) and a corresponding file with the constraint data must be present in the directory `model/constraints/photolysis/` (Sect. 4.2.1). It is not always possible to measure – and therefore constrain – all the required photolysis rates. The photolysis rates that are not constrained (i.e., those not listed in `photolysisConstrained.config`) are calculated using the MCM parametrization (Fig. 3.1).

3.5.3 Calculated photolysis rates

AtChem2 implements the parametrization of photolysis rates used by the Master Chemical Mechanism. The MCM parametrization is described in detail in the protocol papers [Jenkin et al., 1997, Saunders et al., 2003]. Briefly, the photolysis rate of a reaction (`J`) is calculated as a function of the solar zenith angle with the equation:

```
J = l * (cosX)^m * exp(-n * secX) * tau
```

where `l`, `m`, `n` are empirical parameters, `cosX` is the cosine of the solar zenith angle, `secX` is the inverse of `cosX` (i.e., `secX = 1/cosX`) and `tau` is the transmission factor. The empirical parameters are different for each version of the MCM. AtChem2 includes the empirical parameters for the

MCM v3.3.1 (`mcm/photolysis-rates_v3.3.1`). The transmission factor `tau` can be used to account for the loss of natural or artificial light in some environmental chambers and is by default set to 1 (i.e., perfect trasmittance of light). It is possible to use previous versions of the MCM parametrization and to change the value of `tau`: see the file `mcm/INFO.md` for instructions.

The solar zenith angle (SZA) is the angle between the local vertical and the center of the Sun. The SZA is calculated by AtChem2 using the environment variables latitude, longitude, sun declination (Sect. 3.4), and the time of the day (Sect. 3.2). The calculation of the sun declination – if not constrained or set to a constant values – and of the solar zenith angle is described in Madronich [1993].

3.5.4 JFAC calculation

Measurements of ambient photolysis rates typically show short-term variability due to the changing meteorological conditions, such as clouds, rain, aerosol, etc.... This information is retained in the constrained photolysis rates, but it is lost in the calculated ones. To account for this, the calculated photolysis rates can be scaled by a constant or time dependent correction factor, the environment variable `JFAC` (Sect. 3.4).

`JFAC` is defined as the ratio between a measured and a calculated photolysis rate. Typically, `J4` (the photolysis rate of NO_2) is used for this purpose, as it is one of the most frequently measured photolysis rates:

```
JFAC = j(NO2)/J4
```

where `j(NO2)` is measured and `J4` is calculated with the MCM parametrization (Sect. 3.5.3).

`JFAC` is by default set to 1, meaning that the calculated photolyis rates are not scaled; it can be set to any value between 0 and 1 (Sect. 3.4.8) or it can be constrained (Sect. 4.2.1). Note that only the photolysis rates calculated with the MCM parametrization are scaled by `JFAC`, the constrained and the constant photolysis rates are not (Fig. 3.1).

`JFAC` can also be calculated at runtime. To do so, `JFAC` should be set to the name of the photolysis rate to be used as reference (e.g., `J4`) in the file `model/configuration/environmentVariables.config` and there should be a corresponding file with the constraint data in the directory `model/constraints/environment/` [2].

[2]This option is not working well in the current version of AtChem2, especially when `J4` is very variable. Therefore, we suggest to calculate `JFAC` offline and to constrain it (see issue #16)

3.6 Config Files

The configuration files contain the settings for the environment variables, the initial conditions of the chemical species, the model constraints and the model output. These files complement the settings of the model (in `model.parameters`) and of the solver (in `solver.parameters`), which are described in Sect. 3.2 and Sect. 3.3.

The configuration files have the extension `.config` and, by default, are in the `model/configuration/` directory. This directory also contains the files generated during the build process which describes the chemical mechanism. The location of the configuration and mechanism files can be modified by the user (Sect. 2.5.2 and Sect. 3.1.4, provided that all these files are kept in the same directory.

The content and format of the `.config` file are described below. Note that the names of some files have changed with the release of AtChem2 v1.1 (November 2018).

3.6.1 environmentVariables.config

This file contains the settings for the environment variables, which are described in detail in Sect. 3.4. If an environment variable is constrained, there must be a corresponding data file in the `model/constraints/environment/` directory (see Sect. 4.2).

3.6.2 initialConcentrations.config

This file contains the initial concentrations of the chemical species. The file has two columns: the first column is the list of initialized species, and the second column is the corresponding concentration at `t = 0` in **molecules cm-3**. For example:

```
NO      378473308.14
NO2     86893908168.9
O3      1.213e+12
CH4     4.938e+13
```

The chemical species that are not listed in `initialConcentrations.config` are automatically initialized to the default value `0`. It is not necessary to initialize the chemical species that are set to constant or constrained (i.e., those listed in `speciesConstant.config` and in `speciesConstrained.config`). The environment variables are set in `environmentVariables.config` (Sect. 3.6.1) and should not be included in this file.

3.6.3 outputRates.config

This file lists the chemical species for which detailed production rates and loss rates are required. In v1.0 and earlier, these species were listed in two different files called `productionRatesOutput.config` and `lossRatesOutput.config`. The file has one column, with one species per line. The frequency of this output is controlled by the **rates output step size** parameter in `model.parameters` (Sect. 3.2).

The corresponding output files – called `productionRates.output` and `lossRates.output` – are designed to facilitate the analysis of production and destruction rates for selected species of interests (rather than processing the files saved in the `model/output/reactionRates/` directory). The format of these output files is illustrated in Fig. 3.2.

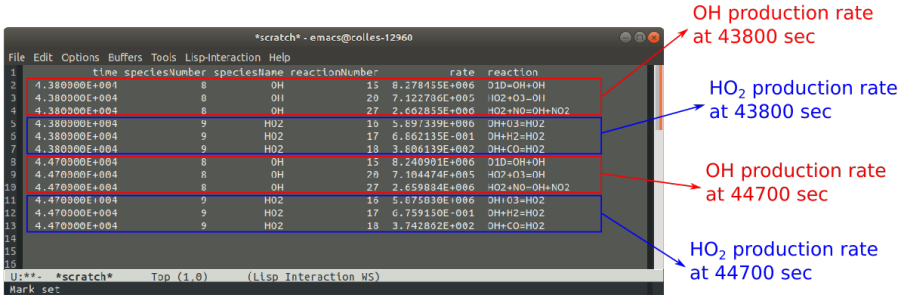

Figure 3.2: Format of the file `productionRates.output`. The file `lossRates.output` has a similar format.

3.6.4 outputSpecies.config

This file (called `concentrationOutput.config` in v1.0 and earlier) lists the chemical species for which the calculated concentration is required. The current version of AtChem2 limits the number of species that can be output to 100, although the user can modify the Fortran code to increase this number. The file has one column, with one species per line.

The frequency of this output is controlled by the **step size** parameter in `model.parameters` (Sect. 3.2). The constrained chemical species can be listed in this file; there is no need to list the photolysis rates, the environment variables and the RO_2 sum, which are always output (see Sect. 4.5).

3.6.5 photolysisConstant.config

This file lists the photolysis rates that are set to constant. The file has three columns: the first column is the number that identifies the photolysis rate (e.g., `1`), the second column is the value of the photolysis rate in **s-1** (e.g., `1e-5`), the third column is the name of the photolysis rate (e.g., `J1`). The

photolysis rates are named according to the MCM naming convention. If no photolysis rate is set to constant, the file should be empty.

If one or more photolysis rate is set to a constant value, the others (i.e., those not listed in `photolysisConstants.config`) are set to zero (Fig. 3.1). For more information go to Sect. 3.5.

3.6.6 photolysisConstrained.config

This file (called `constrainedPhotoRates.config` in v1.0 and earlier) lists the photolysis rates that are constrained. The file has one column, with one photolysis rate per line (e.g., `J1`). The photolysis rates are named according to the MCM naming convention.

If no photolysis rate is constrained, the file should be empty. If a photolysis rate is constrained, there must be a corresponding data file in the `model/constraints/photolysis/` directory (Sect. 4.2.1). The photolysis rates that are not listed in `photolysisConstrained.config` are calculated using the MCM parametrization (Fig. 3.1). For more information go to Sect. 3.5.

3.6.7 speciesConstant.config

hemical species is constrained, the file should be empty. If a chemical species is constrained, there must be a corresponding data file in the `model/constraints/species/` directory (Sect. 4.2.1).

When a chemical species is constrained, it does not need to be initialized: the values set in `speciesConstrained.config` override those set in `initialConcentrations.config`.

Chapter 4

Model Execution

4.1 The Model

AtChem2 is designed to build and run atmospheric chemistry box-models based upon the Master Chemical Mechanism (MCM). Other chemical mechanisms can be used, as long as they are provided in the correct format, as explained in Sect. 3.1.

This chapter explains how to set up, compile and run an AtChem2 atmospheric chemistry model. The directory structure of AtChem2 is described in Sect. 2.5. A working knowledge of the **unix shell** and its basic commands is *required* to use the AtChem2 model.

There are two sets of inputs to AtChem2 – the mechanism file, and the configuration files.

4.1.1 Mechanism file

AtChem2 requires a chemical mechanism in FACSIMILE format (`.fac`). The mechanism file can be downloaded from the MCM website using the extraction tool or it can be assembled manually. The user can modify the downloaded `.fac` file with a text editor.

The mechanism file is converted into a shared library (`.so`) and a number of related files during the build process (see Sect. 3.1.4).

4.1.2 Configuration files

The model configuration is set via a number of text files located in the `model/configuration/` directory:

- model and solver parameters – go to Model Parameters and Solver Parameters.
- environment variables settings – go to Environment Variables.

- photolysis rates settings – go to Photolysis Rates.
- initialization of chemical species, model output settings – go to Config Files.

All the configuration files can be modified with a text editor. Detailed information on the configuration files can be found in the Model Setup chapter. The model constraints – chemical species, environment variables, photolysis rates – are located in the `model/constraints/` directory: for more information go to Sect. 4.2.

4.2 Constraints

AtChem2 can be run in two modes:

- unconstrained: all variables are calculated by the model from the initial conditions, which are set in the model configuration files.
- constrained: one or more variables are constrained, meaning that the solver forces their value to a given value at each time step. The variables that are not constrained are calculated by the model.

The constrained values must be provided as one file for each constrained variable, with the format described below. By default, the files with the constraint data are in `model/constraints/species/` for the chemical species, `model/constraints/environment/` for the environment variables, and `model/constraints/photoly` for the photolysis rates. The default directories can be changed, as explained in Sect. 2.5.2.

4.2.1 Constrained variables

Environment variables

All environment variables, except `ROOF`, can be constrained. To do so, set the variable to `CONSTRAINED` in `model/configuration/environmentVariables.config` and create the file with the constraint data. The name of the file must be the same as the name of the variable, e.g., `TEMP` (without extension). See also Sect. 3.4.

Chemical species

Any chemical species in the chemical mechanism can be constrained. To do so, add the name of the species to `model/configuration/speciesConstrained.config` and create the file with the constraint data. The name of the file must be the same as the name of the chemical species, e.g., `CH3OH` (without extension). See also Sect. 3.6.

Photolysis rates

Any of the photolysis rates in the chemical mechanism can be constrained. The photolysis rates are identified as `J<n>`, where `n` is an integer (Sect. 3.5). To constrain a photolysis rate add its name (e.g., `J4`) to `model/configuration/photolysisConstrai` and create the file with the constraint data. The name of the file must be the same as the name of the photolysis rate, e.g., `J4` (without extension). See also Sect. 3.6.

4.2.2 Constraint files

The files with the constraint data are text files with two columns separated by spaces. The first column is the time in **seconds** from midnight of day/month/year (see Sect. 3.2), the second column is the value of the variable in the appropriate unit. For the chemical species the unit is **molecules cm-3** and for the photolysis rates the unit is **s-1**; for the environment variables see Sect. 3.4. For example:

```
-900    73.21
0       74.393
900     72.973
1800    72.63
2700    72.73
3600    69.326
4500    65.822
5400    63.83
6300    64.852
7200    64.739
```

The time in the first column of a constraint file can be negative. AtChem2 interprets negative times as "seconds *before* midnight of day/month/year" (see Sect. 3.2). This can be useful to allow correct interpolation of the variables at the beginning of the model run, as explained below.

The model constraints *must* cover the same amount of time, or preferably more, as the intended model runtime. For example: if the model starts at 42300 seconds and stops at 216000 seconds, the first and the last data points in a constraint file must have a time of 42300 seconds (or lower) and 21600 seconds (or higher), respectively.

4.2.3 Interpolation

Constraints can be provided at different timescales. Typically, the constraint data come from direct measurements and it is very common for different instruments to sample with different frequencies. For example, ozone and nitrogen oxides can be measured once every minute, but most organic compounds can be measured only once every hour.

The user can average the constraints so that they are all at the same timescale or can use the data with the original timestamps. Both approaches have advantages and disadvantages in terms of how much pre-processing work is required, and in terms of model accuracy and integration speed [Sommariva et al., 2019]. Whether all the constraints have the same timescale or not, the solver interpolates between data points using the interpolation method selected in `model/configuration/model.parameters` (Sect. 3.2). The default interpolation method is piecewise linear, but piecewise constant interpolation is also available.

The photolysis rates and the environment variables are evaluated by the solver when needed – each is interpolated individually, only when constrained. This happens each time the function `mechanism_rates()` is called from `FCVFUN()`, and is controlled by **CVODE** as it carries out the integration. In a similar way, the interpolation routine for the chemical species is called once for each of the constrained species in `FCVFUN()`, plus once when setting the initial conditions of each of the constrained species.

As mentioned above, the model start and stop time *must be* within the time interval of the constrained data to avoid interpolation errors or model crash. If data is not supplied for the full runtime interval, then the *final* value will be used for all times both *before the first data point* and *after the last data point*. This behaviour is likely to change in future versions, at least to avoid the situation where the last value is used for all times before the first (see issue #294). A warning is printed for all evaluations outside of the supplied time interval. It is good practice to supply data that cover a short time *beyond* the final model time, which may be used by the solver.

4.3 Build

The script `build_atchem2.sh` in the `build/` directory is used to process the chemical mechanism file (`.fac`) and to compile the model. The script generates one Fortran file (`mechanism.f90`), one shared library (`mechanism.so`) and four mechanism files (`mechanism.prod`, `mechanism.reac`, `mechanism.ro2`, `mechanism.species`) in the `model/configuration/` directory.

The `build/build_atchem2.sh` script must be run from the *`Main Directory`* and takes four arguments which must be passed in the exact order indicated below. This means that if – for example – the third argument needs to be specified, it is also necessary to specify the first and the second arguments, even if they have the default values. To avoid mistakes, the user can choose to always specify all the arguments. The four arguments, and their default values, are:

1. the path to the chemical mechanism file – no default (suggested: `model/`).
2. the path to the directory for the Fortran and mechanism files generated

from the `.fac` file – default:
`model/configuration/`.

3. the path to the directory containing the configuration files – default:
`model/configuration/`.

4. the path to the directory containing the MCM data files – default:
`mcm/`.

For example, if the chemical mechanism file is in the `model/` directory, the model is build using the command:

```
./build/build_atchem2.sh model/mechanism.fac model/configuration/
                         model/configuration/ mcm/
```

An installation of AtChem2 can have multiple `model/` directories, which may correspond to different models or different projects; this allows the user to run more than one model at the same time. In the following example, the *Main Directory* contains two `model/` directories with different names (`model_1` and `model_2`, each with their own chemical mechanism, configuration, constraints and output:

```
AtChem2/
        | mcm/
        | model_1/
            | configuration/
            | constraints/
            | output/
            | mechanism.fac
        | model_2/
            | configuration/
            | constraints/
            | output/
            | mechanism.fac
        | obj/
        | src/
        | tools/
        | travis/
```

The `model/` directories can also be located outside the *Main Directory*: as long as the correct paths are passed to the `build_atchem2.sh` script and to the exectuable (see below), the model will compile and run. For example:

```
./build/build_atchem2.sh model_1/mechanism.fac model_1/configuration/
                         model_1/configuration/
./build/build_atchem2.sh model_2/mechanism.fac model_2/configuration/
                         model_2/configuration/
```

Note that the fourth argument is not specified in the example above, so the default value (`mcm/`) will be used.

Compilation is required only once for a given `.fac` file. If the user changes the configuration files, there is no need to recompile the model. Likewise, if the constraints files are changed, there is no need to recompile the model. This is because the model configuration and the model constraints are read by the executable at runtime. However, if the user makes changes to the `.fac` file, then the shared library `model/configuration/mechanism.so` needs to be recompiled using the `build_atchem2.sh` script.

The user may also want, or need, to change the Fortran code (`src/*.f90`), in which case the model needs to be recompiled: if the `.fac` file has also been changed, use the `build_atchem2.sh` script. Otherwise, if only the Fortran code has been changed, executing `make` from the *Main Directory* is enough to recompile the model.

4.4 Execute

The compilation process creates an executable file called `atchem2` in the *Main Directory*. The executable file takes seven arguments, corresponding to the directories containing the model configuration and output:

1. the path to the directory for the model output – default:
   `model/output`
2. the path to the directory for the model output reaction rates - default:
   `model/output/reactionRates/`
3. the path to the directory with the configuration files – default:
   `model/configuration/`.
4. the path to the directory with the MCM data files – default:
   `mcm/`.
5. the path to the directory with the data files of constrained chemical species – default:
   `model/constraints/species/`
6. the path to the directory with the data files of constrained environment variables – default:
   `model/constraints/environment/`
7. the path to the directory with the data files of constrained photolysis rates – default:
   `model/constraints/photolysis/`

The model can be run simply by executing the `atchem2` command from the *Main Directory*, in which case the executable will use the default configuration and output directories. Otherwise, the configuration and output directories need to be specified. AtChem2 uses the following flags to pass the arguments to the executable: `--model`, `--output`, `--reactionRates`, `--configuration`, `--constraints`, `--env_constraints`, `--photo_constraints`, `--spec_constraints`, `--mcm`, and `--shared-lib`.

The command `atchem2 --help` displays a help message showing the usage of the command line arguments. For example, if the constraints are in the default directories (or not used), the model can be run by executing:

```
./atchem2 --output=model/output/
          --reactionRates=model/output/reactionRates/
          --configuration=model/configuration/
          --spec_constraints=model_1/constraints/species/
          --env_constraints=model_1/constraints/environment/
          --photo_constraints=model_1/constraints/photolysis/
          --mcm=mcm/
```

In the case of multiple `model/` directories, the directories corresponding to each model need to be passed as arguments to the `atchem2` executable. This allows the user to run two or more models simultaneously. For example:

```
./atchem2 --output=model_1/output/
          --configuration=model_1/configuration/
          --constraints=model_1/constraints/
./atchem2 --output=model_2/output/
          --configuration=model_2/configuration/
          --constraints=model_2/constraints/
```

As explained above, if the chemical mechanism (`.fac`) is changed, only the shared library needs to be recompiled. This allows the user to have only one base executable called `atchem2` in the *Main Directory*: when running multiple models at the same time the user can reuse this base executable while pointing each model to the correct shared library and configuration files.

4.5 Output

The model output is saved by default in the directory `model/output/`. The location can be modified by changing the arguments of the `atchem2` executable, as explained in Sect. 4.4.

The AtChem2 output files are space-delimited text files, with a header containing the names of the variables:

- values of environment variables and concentrations of chemical species: `environmentVariables.output`, `speciesConcentrations.output`.
- values of photolysis rates and related parameters: `photolysisRates.output`, `photolysisRatesParameters.output`.
- loss and production rates of selected species (see Sect. 3.6 for details): `lossRates.output`, `productionRates.output`.
- Jacobian matrix (if requested, see Sect. 3.2): `jacobian.output`.
- model diagnostic variables: `errors.output` `finalModelState.output`, `mainSolverParameters.output`.

In addition, the reaction rates of all the reactions in the chemical mechanism are saved in the directory `reactionRates/`: one file for each model step, with the filename corresponding to the time in seconds. While the model is running diagnostic information is printed to the terminal: this can be redirected to a log file using standard unix commands. A successfull model run completes with a message similar to the one shown in Sect. 2.4.

Chapter 5

Model Development

5.1 General Information

Two versions of AtChem2 are available:

1. the stable version, which is indicated by a version number (e.g., **v1.0**), and can be found here.

2. the development version: which is indicated by a version number with the suffix `-dev` (e.g., **v1.1-dev**), and can be downloaded from the master branch or obtained via **git**.

AtChem2 is under active development, which means that the `master branch` may sometimes be a few steps ahead of the latest stable release. The Test Suite is designed to ensure that changes to the code do not cause unintended behaviour or unexplained differences in the model results, so the development version is usually safe to use, although caution is advised.

The roadmap for the development of AtChem2 can be found here.

Feedback, bug reports, comments and suggestions are welcome. Please check this github page for a list of known and open issues.

If you want to contribute to the model development, the best way is to use **git**. The procedure to contribute code is described on this wiki page. A basic level of knowledge of git is *required*.

5.2 Test Suite

AtChem2 uses Travis CI for Continuous Integration testing. This programming approach ensures changes to the code do not modify the behaviour and the results of the software in an unintended fashion.

To begin using CI on code modifications, create a Pull Request on github from your own fork to `AtChem/AtChem2` (see previous section for instructions on how to set up **git**). Once the PR is created, Travis CI will automatically

run build, unit and behaviour tests on 2 architectures (linux and OSX). Pull requests should only be merged once the Travis CI has completed with passes on both architectures. This is indicated by the message: "All checks have passed".

In order to run the Test Suite on your local machine, call `make alltest` from the *Main Directory*. This will run each of the 3 classes of test in this order:

- unit tests: checks that small fragments of code generate the expected outputs;
- build test: checks that an example program builds and runs successfully;
- behaviour tests: builds each of a number of test setups in turn, and checks that they generate the expected outputs.

Each of the test classes outputs the results of their tests to the terminal screen. To perform just the unit tests, call `make unittests`. To run just the build and behaviour tests, call `make tests`.

The CI tester performs the following on each architecture:

- Install `gfortran`, `cvode`, and `numdiff`
  - linux: use `apt-get` for `gfortran`, `numdiff`, and `liplapack-dev` (a dependency of `cvode`). Install `cvode` from source (`apt-get` could also be used to install `sundials` (including `cvode`), but it doesn't currently hold `cvode 2.9`).
  - OSX: use Homebrew for `gfortran` and `numdiff`. Install `cvode` from source.
- Build and run unit tests. PASS if all unit tests pass.
- Build and run a single example of AtChem2. PASS if this exits with 0.
- Build and run several other examples of AtChem2, using different input files. PASS if no differences from the reference output files are found, otherwise FAIL. Every test must pass to allow the full CI to PASS.

5.2.1 Adding new unit tests

To add new unit tests, do the following:

- Navigate to `travis/unit_tests`. This contains several files with the ending `*_test.f90`. IF the new test to be added fits into an existing test file, edit that file – otherwise, make a new file, but it must follow that pattern of `*_test.f90`. It is suggested that unit tests covering functions from the source file `xFunctions.f90` should be named `x_test.f90`.
- The file must contain a module with the same name as the file, i.e., `*_test`. It must `use fruit`, and any other modules as needed.
- The module should contain subroutines with the naming scheme `test_*~`. These subroutines must take no arguments (and, crucially, not have any brackets for arguments either – subroutine `test_calc` is correct, but subroutine `test_calc()` is wrong).
- Each subroutine should call one or more assert functions (usually `assert_equals()`, `assert_not_equals()`, `assert_true()` or `assert_false()`). These assert functions act as the arbiters of pass or failure – each assert must pass for the subroutine to pass, and each subroutine must pass for the unit tests to pass.
- The assert functions have the following syntax:

```
call assert_true( a == b , "Test that a and b are equal")
call assert_false( a == b , "Test that a and b are not equal")
call assert_equals( a, b , "Test that a and b are equal")
call assert_not_equals( a, b , "Test that a and b are not equal")
```

It is useful to use the last argument as a *unique* and *descriptive* test message. If any unit tests fail, then this will be highlighted in the summary, and the message will be printed. Unique and descriptive messages enable faster and easier understanding of which test has failed, and perhaps why.

If these steps are followed, calling `make unittests` is enough to run all the unit tests, including new ones. To check that your new tests have indeed been run and passed, check the output summary – you should see a line associated to each of the `test*` subroutines in each file in the suite of unit tests.

5.2.2 Adding new behaviour tests

To add a new behaviour test called `$TESTNAME` to the Test Suite, you should provide the following:

Each input `$TESTNAME` should have a subdirectory `travis/tests/$TESTNAME/` containing the following files in the following structure (`*` indicates that this file/directory is optional dependent on the configuration used in the test,

while + indicates that this directory should be populated with the required files for the constraints declared in file in the `model/configuration` directory):

```
|- mcm
|  |- photolysis-rates_v3.3.1
|  |- peroxy-radicals_v3.3.1
|- model
|  |- configuration
|  |  |- $TESTNAME.fac
|  |  |- environmentVariables.config
|  |  |- mechanism.reac.cmp
|  |  |- mechanism.prod.cmp
|  |  |- mechanism.species.cmp
|  |  |- mechanism.ro2.cmp
|  |  |- model.parameters
|  |  |- outputSpecies.config
|  |  |- outputRates.config
|  |  |- *photolysisConstant.config
|  |  |- *photolysisConstrained.config
|  |  |- solver.parameters
|  |  |- *speciesConstrained.config
|  |  |- *speciesConstant.config
|  |  |- initialConcentrations.config
|  |  `- a .gitignore file containing
|  |
|  |       # Ignore everything in this directory
|  |       *
|  |       # Except the following
|  |       !*.config
|  |       !*.parameters
|  |       !.gitignore
|  `- constraints
|     |- *+environment (1)
|     |  `- a .gitignore file containing
|     |     # Ignore nothing in this directory
|     |
|     |         # Except this file
|     |         !.gitignore
|     |
|     |- *+photolysis (1)
|     |  `- a .gitignore file containing
|     |             # Ignore nothing in this directory
|     |
```

```
|     |         # Except this file
|     |         !.gitignore
|     |
|     `- *+species (1)
|        `- a .gitignore file containing
|               # Ignore nothing in this directory
|
|               # Except this file
|               !.gitignore
|- output
|  |- reactionRates/ (3)
|  |- concentration.output.cmp
|  |- environmentVariables.output.cmp
|  |- errors.output.cmp
|  |- finalModelState.output.cmp
|  |- initialConditionsSetting.output.cmp
|  |- jacobian.output.cmp
|  |- lossRates.output.cmp
|  |- mainSolverParameters.output.cmp
|  |- photolysisRates.output.cmp
|  |- photolysisRatesParameters.output.cmp
|  `- productionRates.output.cmp
|- $TESTNAME.out.cmp (2)
```

Notes on this structure:

- if any environment variables (resp. species, photolysis) are to be constrained by data from a file (as set in `model/configuration/environmentVariables.config`, `model/configuration/speciesConstrained.config`, `model/configuration/photolysisConstrained.config`), the subdirectories in `model/constraints/` (`environment/`, `species/`, `photolysis/`) should contain data files with filename equal to the constrained variable name.
- the file `$TESTNAME.out.cmp`, should contain a copy of the expected screen output;
- the subdirectory `reactionRates`, should contain a `.gitignore` file and a copy of each of the appropriate files normally outputted to `reactionRates`, with each suffixed by `.cmp`. The `.gitignore` file should contain

```
\# Ignore everything in this folder
\*
\# except files ending in .cmp
!*.cmp
```

New tests will be picked up by the Makefile automatically when running `make test`.

5.3 Style Guide

In order to make the code more readable, we attempt to use a consistent style of coding. Two scripts, `build/fix_style.py` and `build/fix_indent.py`, help with keeping the style of the Fortran code consistent:

- `build/fix_style.py` edits files in-place to try to be consistent with the style guide (passing two arguments sends the output to the second argument, leaving the input file untouched, and is thus the safer option). This script is by no means infallible.; therefore, when using the script (by invoking `python build/fix_style.py filename`), it is strongly recommended to have a backup of the file to revert to, in case this script wrongly edits.
  This script is also used in the Test Suite to check a few aspects of the styling. This works by running the script over the source file and outputting to a `.cmp` file: if the copy matches the original file, then the test passes.
- `build/fix_indent.py` works similarly, but checks and corrects the indentation level of each line of code. This is also used within the Test Suite.

5.3.1 Style recommendations

General principles

- All code should be within a module structure, except the main program. In our case, due to a complicating factor with linking to CVODE, we also place `FCVFUN()` and `FCVJTIMES()` within the main file `atchem.f90`.
- Code is write in free-form Fortran, so source files should end in `.f90`
- Use two spaces to indent blocks
- Comment each procedure with a high-level explanation of what that procedure does.
- Comment at the top of each file with author, date, purpose of code.
- Anything in comments is not touched by the style guide, although common sense rules, and any code within comments should probably follow the rules below.

Specific recommendations

- All **keywords** are lowercase, e.g., `if then`, `call`, `module`, `integer`, `real`, `only`, `intrinsic`. This also includes `(kind=XX)` and `(len=XX)` statements.
- All **intrinsic** function names are lowercase, e.g., `trim`, `adjustl`, `adjustr`.
- **Relational operators** should use $\geq$, `==` rather than `.GE.`, `.EQ.`, and surrounded by a single space.
- `=` should be surrounded by one space when used as assignment, except in the cases of `(kind=XX)` and `(len=XX)` where no spaces should be used.
- **Mathematical operators** should be surrounded by one space, e.g., `*`, `-`, `+`, `**`.
  - The case of scientific number notation requires no spaces around the `+` or `-`, e.g., `1.5e-9`.
- **Variables** begin with lowercase, while **procedures** (that is, subroutines and functions) begin with uppercase. An exception is **third-party functions**, which should be uppercase. Use either CamelCase or underscores to write multiple-word identifiers.
- **All procedures and modules** should include the 'implicit none' statement.
- All variable **declarations** should include the `::` notation.
- All procedure dummy arguments should include an **intent** statement in their declaration.
- **Brackets**:
  - Opening brackets always have no space before them, except for `read`, `write`, `open`, `close` statements.
  - `call` statements, and the definitions of all procedures should contain **one** space inside the brackets before the first argument and after the last argument, e.g., `call function_name( arg1, arg2 )`, `subroutine subroutine_name( arg1 )`
  - Functions calls, and array indices have **no such space** before the first argument or after the last argument.

Chapter 6

Credits and Acknowledgements

6.1 Credits

AtChem-online was developed at the University of Leeds in 2009-2012 by:

- Chris Martin
- Kasia Borońska
- Jenny Young
- Peter Jimack
- Mike Pilling

Model evaluation, development of test scenarios and model testing were done by Andrew Rickard (NCAS) and Monica Vázquez Moreno (CEAM/EUPHORE). Technical support was provided by David Waller (University of Leeds).

AtChem2 was developed from the **AtChem-online** codebase (rev.146) at the University of Leicester in 2017-2018 by:

- Sam Cox
- Roberto Sommariva (also at University of Birmingham)

6.2 Acknowledgements

Additional contributions by:

- Peter Bräuer
- Vasilis Matthaios

- Beth Nelson
- Mike Newland
- Marios Panagi
- Robert Woodward-Massey

Special thanks to Harald Stark (University of Colorado-Boulder, USA) for providing data to test the photolysis rates subroutines. And to J.-F. Doussin (Université Paris-Est Créteil, France). Bill Bloss and Paul Monks

6.3 Funding

Funding has been provided at different stages by:

- EUROCHAMP project.
- Natural Environment Research Council (NERC).
- University of Leicester Research Software Engineering Team (ReSET).
- NCAS

References

A. C. Hindmarsh, P. N. Brown, K. E. Grant, S. L. Lee, R. Serban, D. E. Shumaker, and C. S. Woodward. SUNDIALS: suite of nonlinear and differential/algebraic equation solvers. *ACM Transactions on Mathematical Software*, 31(3):363–396, 2005. doi: 10.1145/1089014.1089020.

M. E. Jenkin, S. M. Saunders, and M. J. Pilling. The tropospheric degradation of volatile organic compounds: a protocol for mechanism development. *Atmospheric Environment*, 31(1):81–104, 1997. doi: 10.1016/S1352-2310(96)00105-7.

S. Madronich. The atmosphere and UV-B radiation at ground level. In A. R. Young, J. Moan, L. O. Björn, and W. Nultsch, editors, *Environmental UV Photobiology*, pages 1–39. Springer, Boston, MA, 1993. ISBN 978-1-4899-2408-7. doi: 10.1007/978-1-4899-2406-3_1.

S. M. Saunders, M. E. Jenkin, R. G. Derwent, and M. J. Pilling. Protocol for the development of the Master Chemical Mechanism, MCM v3 (Part A): tropospheric degradation of non-aromatic volatile organic compounds. *Atmospheric Chemistry and Physics*, 3(1):161–180, 2003. doi: 10.5194/acp-3-161-2003. URL `https://www.atmos-chem-phys.net/3/161/2003/`.

R. Sommariva, S. Cox, C. Martin, K. Borońska, J. Young, P. Jimack, M. J. Pilling, W. J. Bloss, P. S. Monks, and A. R. Rickard. AtChem, an open source box-model for the Master Chemical Mechanism. In *Atmospheric Chemical Mechanisms Conference*, 2018.

R. Sommariva, S. Cox, C. Martin, K. Borońska, J. Young, P. Jimack, M. J. Pilling, V. N. Matthaios, M. J. Newland, M. Panagi, W. J. Bloss, P. S. Monks, and A. R. Rickard. AtChem, an open source box-model for the Master Chemical Mechanism. *Geoscientific Model Development Discussions*, pages 1–27, 2019. doi: 10.5194/gmd-2019-192.