AtChem2
v1.2-dev

User Manual

R. Sommariva
S. Cox

April 27, 2020

Contents

Chapter 1

Introduction

AtChem2 is an open source modelling tool for atmospheric chemistry. It is designed to build and run zero-dimensional box-models using the Master Chemical Mechanism (MCM). The **MCM** is a near-explicit chemical mechanism which describes the gas-phase oxidation of primary emitted Volatile Organic Compounds (VOC) to carbon dioxide (CO_2) and water (H_2O). The MCM protocol is detailed in Jenkin et al. [1997] and subsequent updates [Saunders et al., 2003, Jenkin et al., 2003, Bloss et al., 2005, Jenkin et al., 2015]. Although it is meant to be used with the MCM, AtChem2 can use any other chemical mechanism, as long as it is in the correct format (Sect. 3.1).

AtChem2 is a development of **AtChem-online**, a modelling web tool created to facilitate the use of the MCM in the simulation of laboratory and environmental chamber experiments within the EUROCHAMP community [Martin, 2009]. AtChem-online runs as a web service – provided by the University of Leeds –and can be accessed at https://atchem.leeds.ac.uk/webapp/: in order to use AtChem-online, a user needs only a text editor, file compression software, a web browser, and an internet connection. A tutorial – with examples and exercises – is available on the MCM website.

AtChem-online is easy to use even for inexperienced users but has a number of limitations, mostly related to its nature as a web application. AtChem2 was developed from AtChem-online with the objective to provide an *offline* modelling tool capable of running long simulations of computationally intensive models, as well as batch simulations for sensitivity studies. In addition, AtChem2 implements a continuous integration workflow, coupled with a comprehensive suite of tests and version control software (git), which makes it robust, reliable and traceable.

The codebase of AtChem2 is structured into four components (or layers) – as illustrated in Fig. 1.1 – and is written mostly in Fortran 90/95. Installation, compilation and execution of AtChem2 are semi-automated via a number of shell and Python scripts that require minimal input from the user. This document (`AtChem2-Manual.pdf`) is the AtChem2 user manual

and contains all the information required to install, set up and use the current version of AtChem2. A summary of these instructions, and additional information, can be found on the wiki.

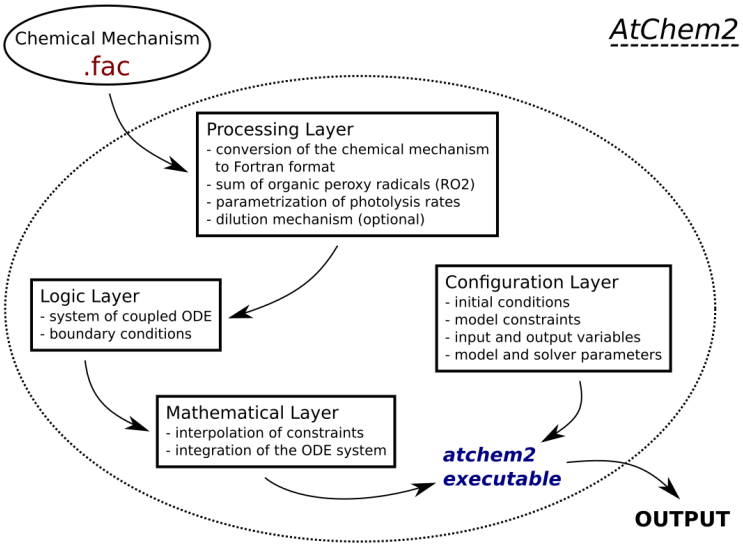

Figure 1.1: Architecture of AtChem2.

1.1 License and citation

AtChem2 is open source – released under **MIT license** – and is hosted at https://github.com/AtChem/AtChem2. A copy of the license can be found in the `LICENSE.md` file.

The model is free to use, compatible with the terms of the MIT license; if used in a publication please include a citation to the following paper:

R. Sommariva, S. Cox, C. Martin, K. Borońska, J. Young, P. K. Jimack, M. J. Pilling, V. N. Matthaios, B. S. Nelson, M. J. Newland, M. Panagi, W. J. Bloss, P. S. Monks, and A. R. Rickard. **AtChem (version 1), an open-source box model for the Master Chemical Mechanism**. *Geoscientific Model Development*, 13, 1, 169–183, 2020. doi: 10.5194/gmd-13-169-2020.

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: `gfortran`
- Python [1]
- cmake
- Ruby (optional)

Some or all of these tools may already be present on your operating 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.

[1] All Python scripts used in AtChem2 work equally with Python v2 and v3. Support for Python v2 may be removed in future versions of AtChem2.

2.2 Download

Two versions of AtChem2 are available for download: the stable version and the development version, also known as `master branch` (see Sect. 5.1 for details). 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 or of the development version to the directory where you want to install AtChem2.
  2. Open the terminal. Move to the directory where the archive file was downloaded.
  3. Execute `tar -zxfv *.tar.gz` or `unzip -v *.zip` (depending on which archive file was downloaded) to unpack the archive file.

The AtChem2 source code is now in a directory called `AtChem2` (if git was used) or `AtChem2-1.x` (if the stable version was downloaded) or `AtChem2-master` (if the development version was downloaded). This directory, which can be given any name, is the *`Main Directory`* of the model. In this manual, the *`Main Directory`* is assumed to be: `$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 given any name. In this manual, the *`Dependency Directory`* is assumed to be: `$HOME/atchem-lib/`.

Before installing the dependencies, make sure that a Fortran compiler, Python, cmake and (optionally) Ruby are installed on the operating 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 using a High Performance Computing (HPC) system: 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 (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 (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 (Sect. 2.4).

2.3.2 Optional dependencies

numdiff

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

```
export 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 it is needed only if you want to run the unit tests (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 the `$HOME` directory (`cd ~`). Open the `.bash_profile` file (or the `.profile` file, depending on your configuration) with a text editor. Add the following lines to the bottom of the file:

   ```
   export GEM_HOME=$HOME/.gem
   export PATH=$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 (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, as indicated in Sect. 2.3.1. Use full paths, because using relative paths may cause compilation errors [2]. 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 (it is optional), leave the default value for `$FRUITDIR`.

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 (called `atchem2`) using the example mechanism file `mechanism_test.fac` included in the `mcm/` directory.

5. Execute `./atchem2` from the *Main Directory*. This command runs the model executable using the default configuration. If the model run completes successfully, the following message (or similar) will be printed to the terminal:

   ```
   ------------------
    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.
   ```

When AtChem2 is run for the first time on a macOS system, it may abort with an error message concerning `rpath`: if this happens, see the **Note for macOS users** on the wiki for a solution.

[2] See issue #364.

AtChem2 is now ready to be used. Optionally, the Test Suite can be run to check that the model has been installed correctly: go to Sect. 2.4.1 for more information. The directory structure and the organization of AtChem2 are described in Sect. 2.5 (see also Fig. 1.1). For detailed instructions on how to set up, configure, build and execute an AtChem2 box-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 runs the requested tests, then prints the tests output and a summary of the results to the terminal.

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 the directory structure of AtChem-online, but it was substantially changed with the release of version 1.1 (November 2018). Tab. 2.1 shows the new directory structure and, for reference, the original one.

In AtChem2 version 1.1 (and later versions) the directories `build/`, `mcm/`, `obj/` and `src/` contain the build scripts, the MCM data files, the files generated by the compiler, the source code; 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 the user 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). Detailed information on these three directories can be found in the following sections.

Table 2.1: Directory structure of AtChem2. "Original" refers to version 1.0 and earlier (including AtChem-online); "New" refers to version 1.1 and later.

Original	New	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 files, 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, environment variables, photolysis rates, production and loss rates, diagnostic information.
`instantaneousRates/`	`model/output/reactionRates/`	model output: reaction rates of every reaction in 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.

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`).

The `tools/` directory contains an auxiliary script (`version.sh`, used to update the version number of the model in each development cycle) and the following subdirectories:

- `install/`: contains the example `Makefile` (`Makefile.skel`) and the scripts to install the Dependencies.
- `plot/`: contains basic plotting scripts in various programming languages.

In earlier versions of AtChem2, the `tools/` directory also included the build scripts, which have been moved to the `build/` directory in AtChem2 version 1.2.

The plotting scripts in `tools/plot/` are very simple and produce identical outputs. They are only intended to give the user a quick overview of the model results for validation and diagnostic purposes. It is recommended 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 [3], Python, and R. One or more of these environments 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 produce the same output [4]: one file called `atchem2_output.pdf` in the given directory.

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/`
- `python tools/plot/plot-atchem2-numpy.py model/output/`
- `python tools/plot/plot-atchem2-pandas.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 from the point of view of 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 but it is good practice to keep it together with the rest of the model configuration.

The `model/` directory can be given any name and can even 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 and in Sect. 4.4.

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

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

[4] See issue #269 for a list of known problems of the plotting scripts.

to recompile the source code and create a different executable every time. Sensitivity studies and batch model runs are therefore easy to do, since all the parts of the model that have to be modified are contained in the same directory. Further information can be found in Sect. 3.1.4.

Important Note: whenever the `model/` directory is mentioned in this manual, it is implied that its name and location may be different than the default name (`model/`) and location (*`Main Directory`*). All scripts that require this information, as well as the `atchem2` executable, allow the user to specify the path to the `model/` directory and to its subdirectories, as needed.

Chapter 3

Model Setup

3.1 Chemical Mechanism

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

3.1.1 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. The `:` character separates the rate coefficient from the reactants and products of the chemical reaction. The rate coefficient (`k`) is a constant number or, more commonly, is calculated as a function of other variables, such as temperature (`TEMP`), air density (`M`), water vapour (`H2O`) and other environment variables (Sect. 3.4).

Comments can be inserted in the `.fac` file to document and annotate the chemical mechanism: in FACSIMILE format, comments are enclosed between the `*` and `;` characters and are ignored by the compilation scripts. A basic chemical mechanism, with comments and calculated rate coefficients, has the following format:

```
* 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 (Sect. 3.5.3). Complex mathematical expressions can be used to calculate the rate coefficients, in which case they have to be defined before the chemical reactions that use them (typically, these are 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 [1]. This feature can be used to implement simple descriptions of non-chemical processes in a box-model. For example, dilution (Sect. 3.4.7), deposition to a 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

[1] Reactions without reactants are not supported in the current version of AtChem2: see issue #413 for a workaround.

calculate the corresponding "rate coefficients" (as explained above) or by adding the appropriate Fortran function(s) to the source code.

The `.fac` file is processed by a Python script (`mech_converter.py`, see Sect. 3.1.4), which 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 used 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 the parametrized non-chemical processes) in FACSIMILE format.

For the `mech_converter.py` script 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 section can be empty (e.g., if a mechanism other than the MCM is used). 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 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 organic peroxy radicals: given that there are over 1000 RO_2 in the MCM, the number of possible self and cross reactions of these species is of the order of 10^6, which presents a significant computational challenge. The RO_2 sum is used in the MCM to reduce the number of peroxy radicals permutation reactions; more information can be found in the MCM protocol papers [Jenkin et al., 1997, Saunders et al., 2003].

AtChem2 is designed primarily to run models based upon the Master Chemical Mechanism and, therefore, the `.fac` file must contain the **Peroxy radicals** section, as explained in 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. The script `mech_converter.py` compares the list of species in the **Peroxy radicals** section of the `.fac` file with the list of RO_2 extracted from the MCM database. AtChem2 includes – in the `mcm/` directory – the complete list of organic peroxy radicals in version 3.3.1 of the MCM (`peroxy-radicals_v3.3.1`), which is the default version used by AtChem2. Complete lists of organic peroxy radicals for other versions of the MCM are also included in the `mcm/` directory. Instructions on how to set up AtChem2 to use previous versions of the MCM can be found in the `mcm/INFO.md` file.

If a species is included in the **Peroxy radicals** section of the `.fac` file, but is not present in the list of RO_2 extracted from the MCM database, the `mech_converter.py` script prints a warning message to the terminal: the user should then check manually whether that species is an organic peroxy radical or not, and amend the `.fac` file accordingly.

It is important to ensure that the RO_2 sum is accurate, because many reactions in the MCM depend on this parameter. This means that the RO_2 sum must include only organic peroxy radicals [2] and that all the organic peroxy radicals in the chemical mechanism must be included. The RO_2 sum is output, by default, to the file `environmentVariables.output` (Sect. 4.5).

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 is 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 all 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 the "Extract" button to download the `.fac` file into a directory of choice – such as the `model/` directory, as discussed in Sect. 2.5.2. The

[2] The hydroperoxyl radical (HO_2) is a peroxy radical but is not an organic molecule, and therefore it *must not* be included in the RO_2 sum.

downloaded `.fac` file is a simple text file and is ready to be used in AtChem2. If modifications are required (e.g., if some chemical reactions have to be added, deleted or modified) 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, the build process is only outlined; detailed instructions on how to build the model can be found in Sect. 4.3.

The Python script `mech_converter.py` – automatically called by the `build_atchem2.sh` script – converts the chemical mechanism from FACSIMILE format 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 showing 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 (from `mechanism.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, by default, `model/configuration/`, but its location can be changed using the second argument of the build script, as explained in Sect. 4.3 (see also Sect. 2.5.2). The shared library `mechanism.so` is created in the *Shared Library Directory*, which usually is the same as the `model/configuration/`.

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. 3.6.4), as well as for 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 run of 2 hours, with output every 5 minutes, requires 24 steps with a step size of 300 seconds (24 x 300 = 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. Two interpolation methods are currently implemented in AtChem2: piecewise constant and piecewise linear (Sect. 4.2.3). The default option is `2` (piecewise linear interpolation).

- **rates output step size**. Frequency (in seconds) of the model output for the rate of the production and destruction analysis (ROPA/RODA) of selected chemical species, i.e. those listed in `outputRates.config` (Sect. 3.6.3).

- **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:

  $$\text{stop time} = \text{start time} + (\text{number of steps} * \text{step size})$$

  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 – see Sect. 4.2.2 and Sect. 4.2.3 for details.

- **jacobian output step size**. Frequency (in seconds) of the model output of the Jacobian matrix. If this parameter is set to `0` (default option), the Jacobian matrix is not output.

  N.B.: 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 [3]. Latitude and longitude are used to calculate the solar angles, which are needed for the calculation of the photolysis rates (Sect. 3.5.3).

- **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. In previous versions of AtChem2, this output was called **instantaneous rates**. 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.

[3] The standard geographical convention is that longitude East is positive and longitude West is negative. AtChem2 uses the opposite convention for backward compatibility reasons. This may change in future versions of AtChem2.

This parameter is different from **rates output step size** (see above), which sets the frequency of a formatted output of the production and loss rates for a limited number of 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.
- **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, solar angles, boundary layer height. 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 (Sect. 4.2.1). 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 environment variables used in AtChem2 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 must be set to `NOTUSED`.

- fixed value
- constrained
- not used

Default = NOTUSED (-1)

3.4.4 H2O

Water Concentration (molecule cm^{-3}). If `H2O` is set to `CALC`, then `RH` must be set to a fixed value or to `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 usually centimetre or metre, depending on how these processes are parametrized in the chemical mechanism – go to 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. When `DILUTE` is set to a fixed value, the chemical mechanism is automatically modified during the build process: for each species in the chemical mechanism, a loss "reaction" is added to the mechanism with a "rate coefficient" equal to the dilution rate. For example, the following "reactions" are added to the "Tropospheric O3-NOx cycle" mechanism shown in Sect. 3.1.1:

```
% DILUTE : NO2 = ;
% DILUTE : NO = ;
% DILUTE : O = ;
% DILUTE : O3 = ;
```

The new set of reactions [4] effectively emulates the dilution of the chemical system. `DILUTE` cannot be constrained.

- fixed value
- not used

Default = 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 (Fig. 3.1). For more information go to Sect. 3.5.4.

- fixed value
- constrained
- calculated

Default = NOTUSED (1)

3.4.9 ROOF

Flag to switch the photolysis rates on/off. It is needed mostly for simulations of some environmental chamber experiments, where the roof of the chamber can be opened/closed or the lamps 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 (Fig. 3.1).

`ROOF` can be set only to `OPEN` (default) or `CLOSED` and cannot be constrained.

[4] AtChem2 treats molecular oxygen (O_2) and nitrogen (N_2) as model parameters, not as chemical species, and their concentrations are calculated by the model as a function of temperature and pressure. Therefore, `DILUTE` is not applied to O_2 and N_2.

3.5 Photolysis Rates

The photolysis rates are the rate coefficients of the photolysis reactions, and are identified in FACSIMILE format with the notation `J<i>` – where `i` is the ID number assigned by the MCM to each photolysis reaction (or to a group of photolysis reactions). AtChem2 implements the MCM parametrization (Sect. 3.5.3) to calculate the photolysis rates; for a comparison between calculated and measured photolysis rates, see Sommariva et al. [2020]. Alternatively, the photolysis rates can be set to a constant value or constrained to measured data. The following rules apply:

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 constrained, it assumes the values in the corresponding constraint file; the other photolysis rates are calculated.
3. If no photolysis rate is set to constant or constrained, the model calculates all photolysis rates.

Fig. 3.1 shows how AtChem2 combines constant, constrained and calculated photolysis rates, as well as the usage of the correction factor JFAC (Sect. 3.5.4). In addition, the environment variable `ROOF` can be used to turn the photolysis rates ON or OFF, as explained in Sect. 3.4.9.

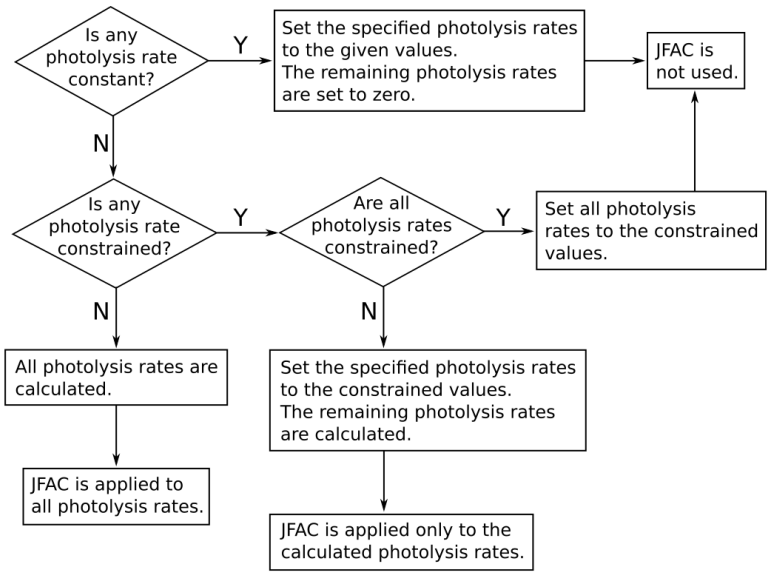

Figure 3.1: Treatment of photolysis rates in AtChem2.

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 in the chemical mechanism must be given a value in the `photolysisConstant.config` file, otherwise they are automatically 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 `photolysisConstant.config` is described in Sect. 3.6.5. If the file is empty, the photolysis rates are calculated or constrained – see below.

3.5.2 Constrained photolysis rates

All photolysis rates can be constrained to measured values. In order to do so, the name of the constrained photolysis rate (e.g., `J2`) must be listed in the `photolysisConstrained.config` file (Sect. 3.6.6); a corresponding file with the constraint data must be present in the `model/constraints/photolysis/` directory (Sect. 4.2.1).

It is not always possible to measure – and therefore to constrain – all the required photolysis rates. The photolysis rates that are not constrained (i.e. that are not listed in `photolysisConstrained.config`) are calculated using the MCM parametrization, described in the next section.

3.5.3 Calculated photolysis rates

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

$$J = l \times (cosX)^m \times e^{(-n \times secX)} \times \tau \tag{3.1}$$

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 τ is the transmission factor. The transmission factor accounts for the loss of natural or artificial light in some environmental chambers: the default value of τ is 1 (i.e. perfect trasmittance of the chamber walls).

The empirical parameters are different for each version of the MCM: by default, AtChem2 uses the empirical parameters of the MCM v3.3.1 (`mcm/photolysis-rates_v3.3.1`), but it is possible to use the empirical parameters of other versions of the MCM and to change the value of τ: see the file `mcm/INFO.md` for instructions.

The solar zenith angle (SZA) is the angle between the local vertical (zenith) and the center of the sun. The SZA is measured in radians and is calculated by the model from latitude, longitude, sun declination (`DEC`) and time of the day. The calculations of the sun declination – if not constrained or set to a constant values – and of the solar zenith angle are described in Madronich [1993].

3.5.4 JFAC calculation

Measurements of ambient photolysis rates typically show short-term variability due to changing meteorological conditions, such as clouds, rain, aerosol, etc... [Sommariva et al., 2020]. This information is retained in the constrained photolysis rates, but it is lost in the calculated ones. To account for ambient variability the calculated photolysis rates can be scaled by a constant or time-dependent correction factor, the environment variable `JFAC` (Sect. 3.4.8), defined as:

$$\text{JFAC} = \frac{j_{meas}}{j_{calc}} \tag{3.2}$$

where j_{meas} and j_{calc} are the measured and calculated (with the MCM parametrization, Sect. 3.5.3) photolysis rates of a reference species. NO_2 is often used as a reference because $j(NO_2)$ is one of the most frequently measured photolysis rates, in which case: `JFAC = j(NO2)/J4`.

`JFAC` is by default set to 1, which means that the calculated photolysis rates are not scaled; `JFAC` 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 never scaled (Fig. 3.1).

`JFAC` can be calculated by AtChem2 at runtime. To use this option, edit the `environmentVariables.config` file (Sect. 3.6.1) and set `JFAC` to the name of the photolysis rate used as reference (e.g., `J4`). In addition, the reference photolysis rate must be constrained – i.e. it must be listed in `photolysisConstrained.config` (Sect. 3.6.6) and the corresponding constraint file must be present in the `model/constraints/photolysis/` directory [5].

3.6 Config Files

The configuration files contain the settings of the environment variables, the chemical species, the photolysis rates, as well as the model constraints and

[5]The calculation of `JFAC` at runtime does not work well in the current version of AtChem2, especially in situations when the reference photolysis rate is very variable. Therefore, it is recommended to calculate `JFAC` offline and then to constrain it (see issue #16).

the model output. All the configuration files have the extension `.config` and, by default, are located in the `model/configuration/` directory. This directory also contains the files with the settings of the model (`model.parameters`) and of the solver (`solver.parameters`), which are described in Sect. 3.2 and Sect. 3.3.

Usually, the `model/configuration/` directory is also the *Shared Library Directory*, which contains the chemical mechanism files generated during the build process. The names and locations of these directories can be modified by the user, as explained in Sect. 2.5.2. The content and the format of each `.config` file are described below. Note that the names of some files have changed with the release of AtChem2 version 1.1 (November 2018).

3.6.1 environmentVariables.config

This file contains the settings of the environment variables (Sect. 3.4). The file has three columns: the first two are the ID number and the name of each environment variable. The third column, which is the only one that should be modified by the user, contains the settings of the environment variables. For example:

```
1   TEMP        293
2   PRESS       1013
3   RH          CONSTRAINED
4   H2O         CALC
5   DEC         CALC
6   BLHEIGHT    8e+4
7   DILUTE      NOTUSED
8   JFAC        CONSTRAINED
9   ROOF        OPEN
```

If an environment variable is constrained, there must be a corresponding data file in the `model/constraints/environment/` directory (Sect. 4.2.1).

3.6.2 initialConcentrations.config

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

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

Not all chemical species need to be initialized: those that are not listed in `initialConcentrations.config` are automatically set to an initial concentration of 0.0E+00 molecule cm^{-3}. It is *not necessary* to initialize the chemical species that are set to constant or constrained (i.e. those listed in `speciesConstant.config` or in `speciesConstrained.config` – see below).

3.6.3 outputRates.config

This file lists the chemical species for which detailed production and loss rates are required. In version 1.0 and earlier, these species were listed in two files called `productionRatesOutput.config` and `lossRatesOutput.config`. The file has one column, with one species per line. For example:

```
OH
HO2
CH3O2
```

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 rate of production and destruction analysis (ROPA/RODA) of selected species of interests, rather than processing all the files saved in the `model/output/reactionRates/` directory. For more information go to Sect. 4.5.

3.6.4 outputSpecies.config

This file (called `concentrationOutput.config` in version 1.0 and earlier) lists the chemical species for which the calculated concentration – in molecule cm^{-3} – is required. The file has one column, with one species per line. For example:

```
O3
NO
NO2
CH4
OH
HO2
CH3O2
```

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 `outputSpecies.config`, which can be useful for diagnostic and debugging purposes. Note that the photolysis rates, the environment variables and the RO_2 sum are always output by the model (Sect. 4.5) and therefore there is not an equivalent `.config` file for the output of these variables.

3.6.5 photolysisConstant.config

This file lists the photolysis rates that are set to constant (Sect. 3.5.1). The file has three columns: the first column is the ID number of the photolysis rate, the second column is the value of the photolysis rate (in s^{-1}), the third column is the name of the photolysis rate. The ID numbers and the names of the photolysis rates follow the MCM designation. For example:

```
1      2.5e-5      J1
4      8.7e-3      J4
7      1.8e-3      J7
```

The photolysis rates that are not listed in `photolysisConstants.config` are automatically set to zero (Fig. 3.1). If no photolysis rate is set to constant, the file should be empty.

3.6.6 photolysisConstrained.config

This file (called `constrainedPhotoRates.config` in version 1.0 and earlier) lists the photolysis rates that are constrained (Sect. 3.5.2). The file has one column, with one photolysis rate per line. The names of the photolysis rates follow the MCM designation. For example:

```
J1
J4
J7
```

If a photolysis rate is constrained, there must be a corresponding constraint 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). If no photolysis rate is constrained, the file should be empty.

3.6.7 speciesConstant.config

This file (called `constrainedFixedSpecies.config` in version 1.0 and earlier) lists the chemical species that are set to constant. The file has two columns: the first column is the list of constant species, the second column is the corresponding concentration – in molecule cm^{-3}. For example:

```
CH4      4.4e+13
CO       2.5e+12
H2       1.2e+13
```

The chemical species that are set to a constant value do not need to be initialized: the values set in `speciesConstant.config` override those set in `initialConcentrations.config`. If no chemical species is set to constant, the file should be empty.

3.6.8 speciesConstrained.config

This file (called `constrainedSpecies.config` in version 1.0 and earlier) lists the chemical species that are constrained. The file has one column, with one species per line. If a chemical species is constrained, there must be a corresponding constraint file in the `model/constraints/species/` directory (Sect. 4.2.1). For example:

```
CH4
CO
H2
```

The chemical species that are constrained do not need to be initialized: the values set in `speciesConstrained.config` override those set in `initialConcentrations.config`. If no chemical species is constrained, the file should be empty.

Chapter 4

Model Execution

4.1 Box-Model

AtChem2 is a modelling tool to build and run atmospheric chemistry box-models [Sommariva et al., 2020]. The structure and organization of AtChem2 are described in Sect. 2.5. An AtChem2 box-model requires two sets of inputs, which are provided by the user: the chemical mechanism file and the configuration files settings – both are described in the following sections. Additionally, the model can be constrained to observational data, as explained in Sect. 4.2.

4.1.1 Mechanism file

AtChem2 is designed to use the MCM as a chemical mechanism; other chemical mechanisms can be used, as long as they are in the correct format.

The chemical mechanism must be provided as a text file in FACSIMILE format, with the extension `.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 `.fac` file with a text editor, if needed.

The `.fac` file is converted into a pre-compiled shared library – called `mechanism.so` – and several mechanism files in Fortran compatible format. These files are created during the build process in the *`Shared Library Directory`* (by default, `model/configuration/`).

4.1.2 Configuration files

The configuration of the box-model is set via a number of text files, which can be modified by the user with a text editor. Detailed information on the configuration files can be found in the corresponding section:

- Model and solver parameters settings– go to Model Parameters and Solver Parameters.

- Environment variables settings – go to Environment Variables.
- Photolysis rates settings – go to Photolysis Rates.
- Model initialization, input and output – go to Config Files.

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 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 constraint data must be provided as one file for each constrained variable, with the format described below. By default, the files with the constraint data are located in `model/constraints/species/` for the chemical species, `model/constraints/environment/` for the environment variables, `model/constraints/photolysis/` for the photolysis rates [1]. The default directories can be changed, as explained Sect. 4.3 (see also Sect. 2.5.2).

4.2.1 Constrained variables

Environment variables

All environment variables, except `DILUTE` and `ROOF`, can be constrained. To do so, set the variable to `CONSTRAINED` in `environmentVariables.config` and create a file with the constraint data (Sect. 4.2.2). The name of the file must be the same as the name of the variable – e.g., `TEMP` (without extension).

Chemical species

Any chemical species in the chemical mechanism can be constrained. To do so, add the name of the species to `speciesConstrained.config` and create a file with the constraint data (Sect. 4.2.2). The name of the file must be the same as the name of the chemical species – e.g., `O3` (without extension).

[1] Although `JFAC` is an environment variable, the `JFAC` constraint file must be in the `constraints/photolysis/` directory.

Photolysis rates

Any photolysis rate in the chemical mechanism can be constrained. The photolysis rates are identified as `J<i>`, where `i` is the ID number assigned by the MCM to each photolysis reaction (Sect. 3.5). To constrain a photolysis rate add its name to `photolysisConstrained.config` and create a file with the constraint data (Sect. 4.2.2). The name of the file must be the same as the name of the photolysis rate – e.g., `J4` (without extension).

4.2.2 Constraint files

The files with the constraint data are text files with two columns and no header: the first column is the time in seconds from midnight of the start date (Sect. 3.2), the second column is the value of the variable in the appropriate unit. For the chemical species the unit is molecule cm^{-3} and for the photolysis rates the unit is s^{-1}; for the units of 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 the start date". Having data points with negative time can be useful to allow correct interpolation of the variables at the beginning of the model run. This is because the model constraints *must cover* the same amount of time, or preferably more, as the intended model runtime (Sect. 4.2.3).

For example: if the model starts at 41400 seconds (day 1 at 11:30) and stops at 225900 seconds (day 3 at 14:45), then the first and the last data points of a constraint file must have a time of 41400 seconds (or lower) and 225900 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 (O_3)

and nitrogen oxides (NO, NO_2) can be measured once every minute, but most hydrocarbons can be measured only once every 30-60 minutes. The user can average the constraints so that they are all with the same timescale, or can use the constraint data with the original timescales. 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: for more information, see the discussion in Sommariva et al. [2020]. It is up to the user to decide which approach is more suitable, based on the objectives of the modelling work and on the available computing resources.

Whether all the constraints have the same timescale or not, AtChem2 interpolates between data points using the interpolation method selected in `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 in Sect. 4.2.2, the model start and stop time *must be* within the time interval of the constrained data to avoid interpolation errors or model crashes. If data is not supplied for the entire runtime interval, the final value of the constrained variable will be used for all times *before the first* data point and *after the last* data point. If this situation occurs, a warning is printed to the terminal for all data evaluations outside of the supplied time interval [2]. In any case, and to avoid errors, it is good practice to always provide constraint data that include a short period before the start time and a short period after the stop time.

4.3 Build

The script `build_atchem2.sh` in the `build/` directory is used to process the chemical mechanism file (`.fac`) and compile the box-model. The script generates one Fortran file (`mechanism.f90`), one pre-compiled shared library (`mechanism.so`) and four mechanism files in Fortran compatible format (`mechanism.species`, `mechanism.reac`, `mechanism.prod`, `mechanism.ro2`). The content and the format of these files are described in Sect. 3.1.4.

The `build_atchem2.sh` script must be run from the *`Main Directory`* and takes three arguments which must be provided to the script in the ex-

[2] This behaviour is likely to change in future versions of AtChem2, at least to avoid the situation where the last value is used for all times before the first data point (see issue #294).

act order indicated below. This means that if – for example – the second argument needs to be specified, it is also necessary to specify the first argument, even if it has the default value. To avoid mistakes, the user can choose to always specify all the arguments. The three arguments, and their default values, are:

1. the path to the chemical mechanism file – there is no default, but it is suggested to keep the `.fac` file in `model/` or in `model/configuration/`.
2. the path to the the directory for the Fortran and mechanism files and to the *Shared Library Directory* – default: `model/configuration/`.
3. the path to the directory with 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/ mcm/
```

An installation of AtChem2 can have multiple `model/` directories, corresponding to different models or different projects; this allows the user to work with more than one model at the same time and makes it easy to run batch simulations for sensitivity studies. As mentioned in Sect. 2.5.2, the `model/` directory can also be located outside the *Main Directory*, which gives the users the flexibility to organize the modelling work as they prefer.

Fig. 4.1 shows a possible setup: a user directory (e.g., `$HOME`) contains an AtChem2 installation – with the default `model/` directory – plus a project directory (`Project_A`), which corresponds, for example, to a field campaign or to a set of related experiments. The project directory contains two different `model/` directories (`model_1/` and `model_2/`), each with their own chemical mechanism, configuration, constraints and output. In this case, the user can build each model with the following commands [3]:

```
./build/build_atchem2.sh ~/Project_A/model_1/mechanism.fac
                         ~/Project_A/model_1/configuration/

./build/build_atchem2.sh ~/Project_A/model_2/mechanism.fac
                         ~/Project_A/model_2/configuration/
```

There are many different ways in which the `model/` directory can be customized: for example, it is possible to keep all the constraint files related to a project in one directory, thus avoiding the need to have identical

[3] The third argument is not specified in this example, so the default value (`~/AtChem2/mcm/`) will be used.

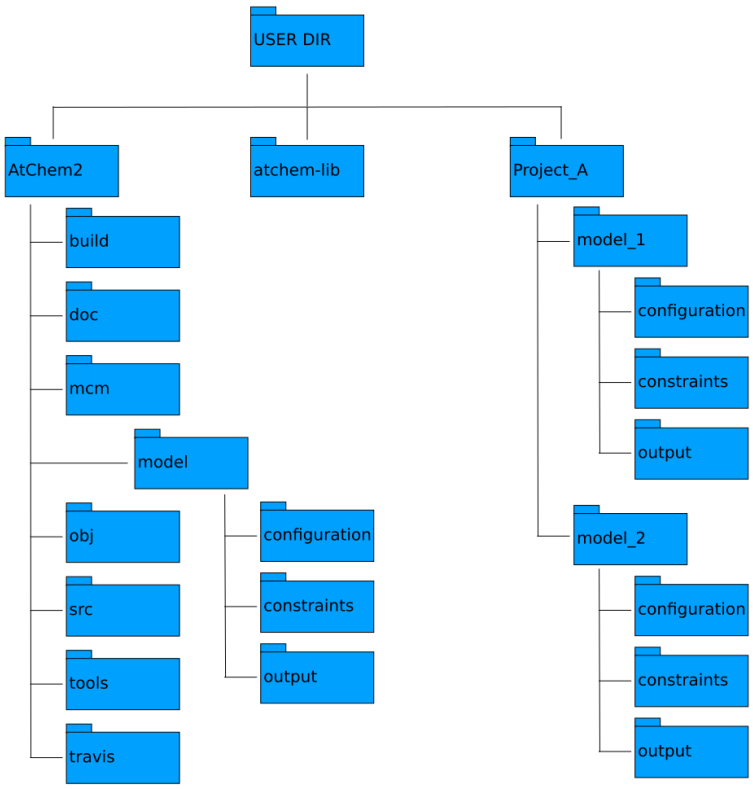

Figure 4.1: Example of a modelling setup, with an AtChem2 installation and a project directory containing two box-models.

`constraints/` directories in each `model/` directory. Each user has specific needs and personal preferences; as long as the correct paths are passed to the `build_atchem2.sh` script (and to the executable, see Sect. 4.4), the model will compile and run.

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 changes the `.fac` file, then the shared library (`mechanism.so`) needs to be recompiled by running the `build_atchem2.sh` script again. 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, the `build_atchem2.sh` script must be used; otherwise – if only the Fortran code has been changed – executing the `make` command from the *`Main Directory`* is enough to recompile the model.

4.4 Execute

The build process (Sect. 3.1.4 and Sect. 4.3) creates an executable file called `atchem2` in the *Main Directory*. The executable file takes up to nine arguments, corresponding to the *relative paths* (with respect to the *Main Directory*) of the model configuration, the shared library, the constraint files, and the model output:

1. the path to the model directory – default:
   `model/`
2. the path to the directory for the model output – default:
   `model/output`
3. the path to the directory with the configuration files – default:
   `model/configuration/`.
4. the path to the directory with the model constraints – default:
   `model/constraints/`
5. the path to the directory with the data files of constrained environment variables – default:
   `model/constraints/environment/`
6. the path to the directory with the data files of constrained photolysis rates – default:
   `model/constraints/photolysis/`
7. the path to the directory with the data files of constrained chemical species – default:
   `model/constraints/species/`
8. the path to the directory with the MCM data files – default:
   `mcm/`.
9. the path to the shared library – default:
   `model/configuration/mechanism.so`.

AtChem2 uses a series of input flags to pass the arguments to the executable: `--model`, `--output`, `--configuration`, `--constraints`, `--env_constraints`, `--photo_constraints`, `--spec_constraints`, `--mcm`, `--shared_lib`. In addition, the input flag `--help` displays an help message which shows the usage of the command line arguments and of the input flags.

AtChem2 can be run simply by executing the command `./atchem2` from the *Main Directory*, in which case the executable will assume that all arguments have the default values. The equivalent command using the input flags is:

```
./atchem2 --model=model/
          --output=model/output/
          --configuration=model/configuration/
          --constraints=model/constraints/
          --spec_constraints=model/constraints/species/
          --env_constraints=model/constraints/environment/
          --photo_constraints=model/constraints/photolysis/
          --shared_lib=model/configuration/mechanism.so
          --mcm=mcm/
```

Not all flags have to be used, and the order in which they are used does not matter. If a flag is not used, the executable assumes the default value following a hierarchical directory structure. For example, the following command assumes that a directory called `output/` is present inside the `model/` directory, and that three directories called `species/`, `environment/` and `photolysis/` are present inside the `model/constraints/` directory:

```
./atchem2 --model=model/
          --configuration=model/configuration/
          --constraints=model/constraints/
          --shared_lib=model/configuration/mechanism.so
```

This approach gives the the user a lot of flexibility in the organization of the modelling work and makes it possible to run several models using the same executable with different configurations, constraint sand chemical mechanisms. For example, the following commands run the same model with two different configurations and save the corresponding outputs in separate directories:

```
./atchem2 --configuration=model/configuration_1/ --output=model/output_1/
./atchem2 --configuration=model/configuration_2/ --output=model/output_1/
```

It is also possible to use configurations or constraints from different models: for example, based on the setup shown in Fig. 4.1, the following command runs a model constrained to the chemical species and environment variables of `model_1/`, but constrained to the photolysis rates of `model_2/`:

```
./atchem2 --configuration=model/configuration/
          --output=model/output/
          --spec_constraints=~/Project_A/model_1/constraints/species/
          --env_constraints=~/Project_A/model_1/constraints/environment/
          --photo_constraints=~/Project_A/model_2/constraints/photolysis/
```

AtChem2 can be installed and run on High Performance Computing (HPC) systems. This is recommended, especially for models with long runtimes and/or many constraints. Each HPC system has its own rules and

setup, so it is not possible to give specific advice. Users should check the local documentation or ask the system administrator; some information about running AtChem2 on HPC systems can be found on the wiki.

4.5 Output

The model output is saved by default in the `model/output/` directory. The location can be modified by changing the arguments of the `atchem2` executable, as explained in Sect. 4.4 (see also Sect. 2.5.2). The frequency of the model output is determined by the following model parameters, which are set in the `model.parameters` file (Sect. 3.2):

- **step size** for the chemical species, the environment variables, the photolysis rates, the diagnostic variables.
- **rates output step size** for the rate of production and destruction analysis (ROPA/RODA) of selected species.
- **reaction rates output step size** (previously called **instantaneous rates**) for the reaction rates of all chemical reactions.

All AtChem2 output files are space-delimited text files with a one line header containing the names of the variables. The first column of each file is the model time (`t`) in seconds since midnight of the start date [4]:

- Environment variables and RO_2 sum:
  `environmentVariables.output`
- Concentrations of the chemical species:
  `speciesConcentrations.output`
- Photolysis rates:
  `photolysisRates.output`
- Latitude, longitude, solar angles and related parameters:
  `photolysisRatesParameters.output`
- Loss and production rates of selected species:
  `lossRates.output`
  `productionRates.output`
- Jacobian matrix (optional, see Sect. 3.2):
  `jacobian.output`

[4]Note that the **start date** is different than the **model start time**. The model start time indicates when the model begins its run and is in seconds since the midnight of the start date (Sect. 3.2).

- Error messages and diagnostic variables:
  `errors.output`
  `finalModelState.output`
  `mainSolverParameters.output`

In addition to the `.output` files, the reaction rates of every reaction in the chemical mechanism are saved in the `reactionRates/` directory – called `instantaneousRates/` in previous versions of AtChem (Tab. 2.1) – as one file for each model step, with the name of the file corresponding to the output time in seconds.

The reaction rates files are useful for diagnostic purposes, but can be cumbersome to process and analyze. In order to make it easier to perform the rate of production and destruction analysis of chemical species of particular interest – i.e. those listed in `outputRates.config` (Sect. 3.6.3) – the model produces the output files `productionRates.output` and `lossRates.output`. These files contain the reaction rates of production and destruction of the selected species in a human readable format, illustrated in Fig. 4.2.

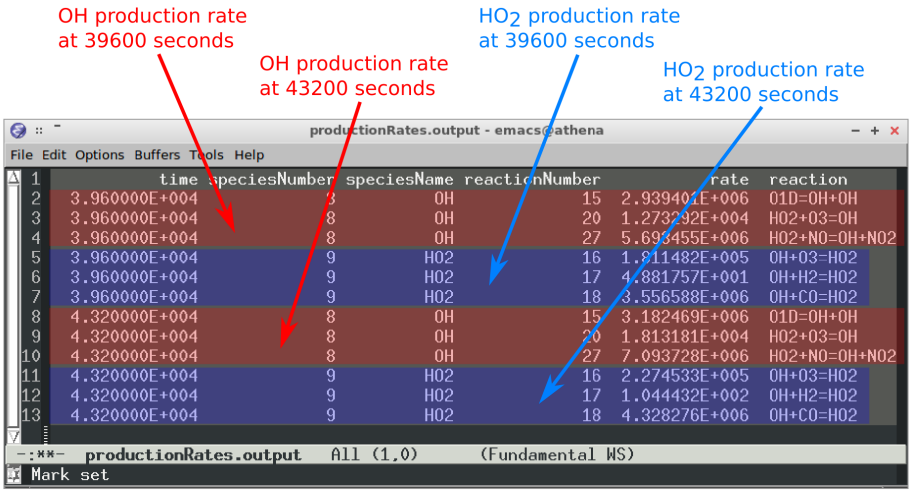

Figure 4.2: Format of the `productionRates.output` file. The `lossRates.output` file has a similar format.

While the model is running, diagnostic information is printed to the terminal: this can be redirected to a log file using standard unix commands. On HPC systems the submission script can usually take care of redirecting the terminal printout to a log file (see the related wiki page). A successful model run completes with a message similar to the one shown in Sect. 2.4.

Chapter 5

Model Development

5.1 General Information

There are two versions of AtChem2 in the main github repository:

1. Stable version: is indicated by a version number (e.g., **v1.0**), and can be downloaded from the Releases page.

2. Development version – i.e. the `master branch`: is indicated by a version number with the suffix `-dev` (e.g., **v1.1-dev**), and can be downloaded as an archive file or obtained via **git** (Sect. 2.2).

AtChem2 is under active development, which means that the `master branch` is sometimes a few steps ahead of the latest stable release. Any modification to the code is automatically run through the Test Suite before it is merged into the `master branch`. 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 normally safe to use. However, it is recommended to use the stable version for "production runs" and publications since it can be more easily referenced – see the discussion about traceability and reproducibility of computational models in Sommariva et al. [2020].

Feedback, bug reports, comments and suggestions are welcome: the list of open issues and known bugs can be found on the related github page. The preferred way to contribute to the development of AtChem2 is to use **git**: instructions on how to set up git and submit contributions can be found on the wiki. The coding guidelines for Fortran are detailed in Sect.5.3.

5.2 Test Suite

AtChem2 uses Travis CI for **continuous integration** testing. This programming approach ensures that changes to the code do not modify the

behaviour and the results of the software in an unintended fashion. The Test Suite is in the `travis/` directory and consists of a series of tests and short model runs that check the model functionality and calculations against known outputs. The Codecov service is used to verify that the Test Suite covers a significant fraction of the codebase (>90%) and a wide range of common configurations.

There are four types of tests, which can be executed from the *`Main Directory`* using the `make` command (note that the optional dependencies need to be installed):

- **Indent** and **Style** tests: check that the indentation and coding style of the Fortran code are consistent with the guidelines – `make indenttest` and `make styletest`.
- **Unit** tests: check that individual functions generate the expected outputs – `make unittests`.
- **Behaviour** tests: build and run a number of models with different configurations and check that they generate the expected outputs – `make tests`.

The command `make alltests` runs all the tests in the Test Suite in succession. Each test outputs the results to the terminal and, in case of failure, a log file (`tests/testsuite.log`) is generated for the indent, style and behaviour tests. If all tests are successfully passed the following message is printed to the terminal:

```
Style test       PASSED
Indentation test PASSED
Tests            PASSED
  (20/20 tests PASSED)
Testsuite PASSED
```

The Test Suite is automatically run every time a Pull Request (PR) is created or updated on the main github repository (`AtChem/AtChem2`). The Pull Request triggers a build on Travis CI which runs the entire Test Suite on two architectures (Linux and macOS) with one compiler (GNU `gfortran`). The CI tester performs the following tasks 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 [1].

[1] `apt-get` could also be used to install SUNDIALS, but the repository does not currently hold CVODE v2.9.

  - macOS: use `Homebrew` for `gfortran` and `numdiff`. Install `cvode` from source.

- Build and run the example AtChem2 model using the default configuration. PASS if it exits with 0.
- Build and run the indent and style tests. PASS if all tests pass.
- Build and run the unit tests. PASS if all unit tests pass.
- Build and run the behaviour tests. PASS if no differences from the reference output files are found, otherwise FAIL.

Every test must pass to allow the full CI to pass. This is indicated by the message "All checks have passed" on the github PR page. Pull Requests should only be merged into the `master branch` once Travis CI has completed with passes on both architectures.

5.2.1 Adding new unit tests

The unit tests are in the `travis/unit_tests` directory and require the FRUIT optional dependency (which requires Ruby, Sect. 2.3.2). To add new unit tests, follow the procedure outlined below:

- The unit test files are called `*_test.f90`. If the new unit test to be added fits into an existing test file, edit that file – otherwise, create a new test file following the same naming pattern. It is suggested that unit tests covering functions from the source file `xFunctions.f90` should be named `x_test.f90`.
- The unit test file must contain a module with the same name as the file – i.e. `x_test` – and it must include the statement `use fruit`, plus any other required module.
- The module should contain a number of subroutines with the naming pattern `test_*`. These subroutines must take no arguments and, importantly, must not have any brackets for arguments – subroutine `test_calc` is correct, but subroutine `test_calc()` is wrong.
- Each subroutine should call one or more assert functions: usually, these are `assert_equals()`, `assert_not_equals()`, `assert_true()`, `assert_false()`. The assert functions act as the arbiters of pass or failure of the unit test – 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 of the assert function as a *unique* and *descriptive* message. When a unit test fails, it is highlighted in the FRUIT output summary, and the message of the assert function is printed. Unique and descriptive messages thus 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 the new ones. To verify that the new tests have indeed been run and passed, check the output summary – there should be a line associated to each of the `test_*` subroutines in each test file.

5.2.2 Adding new behaviour tests

Each behaviour test (`$TESTNAME`) is contained in its own subdirectory inside the `travis/tests/` directory. A behaviour test requires the following files and directory structure:

```
|- 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
|  |- constraints
|     |- environment/ (**)
|     |- photolysis/ (**)
|     |- species/ (**)
|- output
```

```
|  |- reactionRates/
|  |- 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
```

The files marked with `(*)` and the directories marked with `(**)` are optional – depending on the configuration used in the test. If present, the directories marked with `(**)` should contain the relevant constraint files, according to the corresponding configuration files in `model/configuration/`. In addition, these directories should contain a `.gitignore` file with the following content:

```
# Ignore nothing in this directory

# Except this file
!.gitignore
```

The file `$TESTNAME.out.cmp` should contain the exact copy of the expected terminal printout. Each behaviour test is briefly described in the `travis/tests/INFO.md` file, which should be updated after a new test is added. New tests added to the `travis/tests/` directory are automatically picked up by the `Makefile` when running `make test` or `make alltests` from the *Main Directory*.

5.3 Style Guide

In order to make the AtChem2 code more readable and easier to maintain, the source code should follow a consistent style (Sect. 5.3.1). Two Python scripts are used to check and correct the Fortran code:

- `fix_style.py` edits a Fortran file in-place to make the code consistent with the style recommendations.
- `fix_indent.py` works in a similar way, but only looks at the indentation level of each line of code.

These scripts are in the `build/` directory and can be invoked from the *`Main Directory`* with the following commands:

```
python build/fix_style.py src/filename.f90
python build/fix_indent.py src/filename.f90
```

It is important to keep in mind that these scripts are *not infallible* and, therefore, it is strongly recommended to always have a backup of the Fortran file to revert to, in case of wrong edits. This can be done by passing two arguments to the script instead of one: the second argument sends the script output to another file, leaving the original file untouched.

Both scripts are also used in the Test Suite to run the style and indent tests (Sec. 5.2): each script is run over each source file and the output is sent to a `.cmp` file. If the `.cmp` file matches the original file, the test passes.

5.3.1 Style recommendations

General principles

- All code should be organized in a module structure, except the main program. There is only one exception: due to a complicating factor with linking to CVODE, the functions `FCVFUN()` and `FCVJTIMES()` are placed within the main file `atchem.f90`.
- All code should be written in free-form Fortran, and the source files should have the extension `.f90`.
- Always use two spaces to indent blocks.
- At the top of each file there should be a header indicating the author(s), date, and purpose of the code; if necessary, acknowledgements to other contributors should be added.
- Always comment a procedure with a high-level explanation of what that procedure does.
- There are no specific guidelines for comments, although common sense applies and any code within the comments should broadly follow the rules below.

Specific recommendations

- All **keywords** should be lowercase, e.g., `if then`, `call`, `module`, `integer`, `real`, `only`, `intrinsic`. This includes the `(kind=XX)` and `(len=XX)` statements.
- All **intrinsic** function names should be lowercase, e.g., `trim`, `adjustl`, `adjustr`.

- The **relational operators** should use $\geq$ and `==` rather than `.GE.`, `.EQ.`, and should be surrounded by a single space.
- The `=` operator should be surrounded by one space when used as assignment – except in the cases of the `(kind=XX)` and `(len=XX)` statements, where no spaces should be used.
- The **mathematical operators** (`*`, `-`, `+`, `**`) should be surrounded by one space.
- Numbers in scientific notation should have no spaces around the `+` or `-`, e.g., `1.5e-9`.
- The names of **variables** should begin with lowercase, while those of **procedures** (that is, subroutines and functions) should 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.
- The **dummy arguments** of a procedure should include an `intent` statement in their declaration.
- The following rules apply to **brackets**:
  - Opening brackets should not have a space before them, except for `read`, `write`, `open`, `close` statements.
  - All `call` statements and the definitions of all procedures should contain one space before the first argument and one after the last argument inside the brackets:

    ```
    call function_name( arg1, arg2 )
    subroutine subroutine_name( arg1 )
    ```

  - Functions calls and array indices should not have spaces before the first argument or after the last argument inside the brackets.

Chapter 6

Credits and Acknowledgements

6.1 Credits

AtChem2 has been developed at the **University of Leicester** by:

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

Additional code has been contributed by:

- Beth Nelson
- Mike Newland
- Marios Panagi
- Maarten Fabré

AtChem2 is a development of AtChem-online, created at the **University of Leeds** by:

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

Model evaluation and testing of AtChem-online was performed by Andrew Rickard (NCAS/York) and Monica Vázquez Moreno (CEAM/EUPHORE), and technical support was provided by David Waller (University of Leeds).

6.2 Acknowledgements

Thanks for their support, feedback, and contributions to:

- Bill Bloss
- Peter Bräuer
- Nahid Chowdhury
- Vasilis Matthaios
- Paul Monks
- Jon Wakelin
- Robert Woodward-Massey

Special thanks to Harald Stark (University of Colorado-Boulder, USA) for providing the observational data used to test the photolysis rates subroutines.

6.3 Funding

Funding provided, at different stages, by:

- EUROCHAMP project.
- National Centre for Atmospheric Science (NCAS).
- Natural Environment Research Council (NERC).
- University of Leicester ReSET programme.

References

C. Bloss, V. Wagner, M. E. Jenkin, R. Volkamer, W. J. Bloss, J. D. Lee, D. E. Heard, K. Wirtz, M. Martin-Reviejo, G. Rea, J. C. Wenger, and M. J. Pilling. Development of a detailed chemical mechanism (MCMv3.1) for the atmospheric oxidation of aromatic hydrocarbons. *Atmospheric Chemistry and Physics*, 5(3):641–664, 2005. doi: 10.5194/acp-5-641-2005.

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.

M. E. Jenkin, S. M. Saunders, V. Wagner, and M. J. Pilling. Protocol for the development of the Master Chemical Mechanism, MCM v3 (Part B): tropospheric degradation of aromatic volatile organic compounds. *Atmospheric Chemistry and Physics*, 3(1):181–193, 2003. doi: 10.5194/acp-3-181-2003.

M. E. Jenkin, J. C. Young, and A. R. Rickard. The MCM v3.3.1 degradation scheme for isoprene. *Atmospheric Chemistry and Physics*, 15(20):11433–11459, 2015. doi: 10.5194/acp-15-11433-2015.

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.

C. J. Martin. *Chemical models for, and the role of data and provenance in, an atmospheric chemistry community.* PhD thesis, School of Chemistry & School of Computing, University of Leeds, 2009. URL `https://etheses.whiterose.ac.uk/1596/`.

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.

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. K. Jimack, M. J. Pilling, V. N. Matthaios, B. S. Nelson, M. J. Newland, M. Panagi, W. J. Bloss, P. S. Monks, and A. R. Rickard. AtChem (version 1), an open-source box model for the Master Chemical Mechanism. *Geoscientific Model Development*, 13(1):169–183, 2020. doi: 10.5194/gmd-13-169-2020.