AtChem2 v1.2.3

User Manual

R. Sommariva
S. Cox

May 21, 2025

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., 2012, 2015]. AtChem2 is designed for use with the MCM, but it is able to 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 at the University of Leeds 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 York: in order to use AtChem-online, the user needs only a text editor, a file compression tool, 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; the aim is to provide an *offline* modelling tool capable of running long simulations of computationally intensive models, as well as batch simulations (e.g., for sensitivity studies). In addition, AtChem2 implements a continuous integration workflow, along with a comprehensive Testsuite and a version control workflow (with git), which makes it robust, reliable and traceable.

The codebase of AtChem2 is structured into four components (or layers, see Fig. 1.1) and is written 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. Model configuration and inputs consist of simple text files that can be easily modified with any text editor.

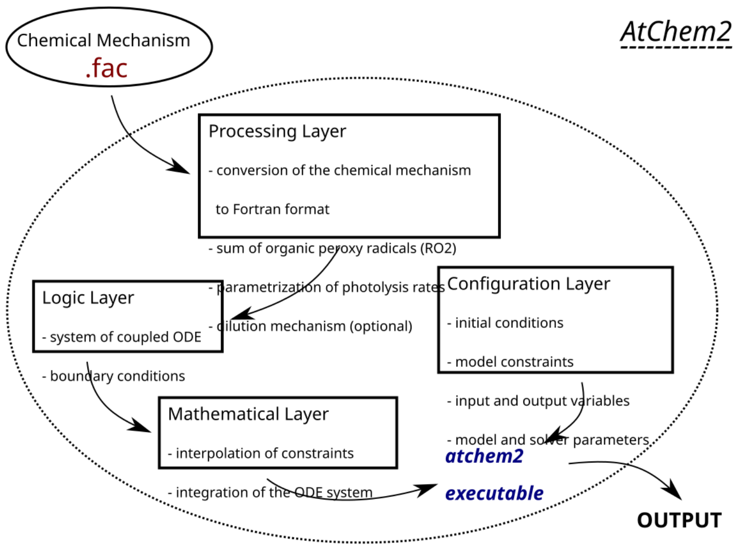

Chemical Mechanism
.fac
AtChem2
Processing Layer
- conversion of the chemical mechanism
to Fortran format
- sum of organic peroxy radicals (RO2)
- parametrization of photolysis rates
- dilution mechanism (optional)
Logic Layer
- system of coupled ODE
- boundary conditions
Configuration Layer
- initial conditions
- model constraints
- input and output variables
- model and solver parameters
Mathematical Layer
- interpolation of constraints
- integration of the ODE system
atchem2
executable
OUTPUT

Figure 1.1: Architecture of AtChem2.

This document (`AtChem2-Manual.pdf`) is the AtChem2 user manual and contains all the information required to install, configure, and use the current version of the model. The AtChem2 wiki contains summaries of these instructions for quick user reference; additional information, such as example submission scripts for HPC systems (Sect. 4.4.1), and guidance for resolving some known issues are also available on the wiki.

AtChem-tools is a set of Python tools designed to be used with AtChem2, available in the related repository. The tools are written in Python and include scripts to help with the model setup and configuration, to run the model, and to analyze the model outputs.

1.1 Conventions

The following conventions are adopted in the AtChem2 user manual and documentation:

`Main Directory` is the base directory of the AtChem2 model.
For example: `$HOME/AtChem2/`.

`Dependencies Directory` is the directory where the AtChem2 dependencies are installed.
For example: `$HOME/AtChem-lib/`.

Shared Library Directory is the directory for the pre-compiled chemical mechanism shared library (`mechanism.so`).
By default, it is `model/configuration/` inside the *`Main Directory`*.

The user can change the locations and names of these directories according to their setup, workflow, personal preferences, etc... As long as the correct paths to the libraries and directories needed by AtChem2 are specified, the model will compile and run.

Important Note: AtChem2 runs in **Coordinated Universal Time (UTC)**, with no daylight saving. This is to simplify the calculation of the solar angles and, therefore, of the photolysis rates (Sect. 3.5). The user must ensure that the input data and the model constraints are provided in UTC (i.e. GMT timezone) and convert the output to Local Time, if needed.

1.2 License and citation

AtChem2 is open source and free to use, under the terms of the **MIT license**. A copy of the license can be found in the `LICENSE` file. If used in a publication, please cite the GMD paper [Sommariva et al., 2020]:

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 and used on either Linux/Unix or 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 software tools:

- ◇ a terminal with bash or a bash-compatible shell (e.g., zsh is the default shell in macOS), and common utilities: wget, tar, zip, make, etc. . .
- ◇ a text editor: Emacs, vim, Gedit, Kate, TextEdit, Atom, etc. . .
- ◇ a Fortran compiler – the model compiles with GNU `gfortran` (version 4.8 and later) and with Intel `ifort` (version 17.0 and later).
  The default Fortran compiler is `gfortran`.
- ◇ Python [1].
- ◇ Ruby version 3.0 or earlier (optional).

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

- ◇ CVODE – requires gcc, cmake
- ◇ openlibm
- ◇ numdiff (optional)

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

◇ FRUIT (optional)

For detailed instructions on the installation and configuration of the dependencies, go to Sect. 2.3. As an alternative, AtChem2 can be run as a Docker container. For details, go to Sect. 2.5.

2.2 Download

The primary repository for the AtChem2 code and documentation is: https://github.com/AtChem/AtChem2.

Two versions of the model are available: the *stable version* and the *development version* – i.e. the `master branch` of the git repository, see Sect. 5.1. Although the two versions are usually equivalent, it is recommended to use the stable version which can be easily referenced in publications using the version and doi numbers. The AtChem2 source code can be obtained in two ways:

1. with **git** (this is the best method for users who want to contribute to the model development, Sect. 5):
   - ◇ Open the terminal. Move to the directory where AtChem2 will be installed.
   - ◇ Execute one of the following commands to copy the `master branch` to the local machine:
     - ∘ if using HTTPS:
       `git clone https://github.com/AtChem/AtChem2.git`
     - ∘ if using SSH:
       `git clone git@github.com:AtChem/AtChem2.git`
2. with the **archive file**:
   - ◇ Download the archive file (`.tar.gz` or `.zip`) of the stable or the development version to the directory where AtChem2 will be installed.
   - ◇ Open the terminal. Move to the directory where the archive file was downloaded.
   - ◇ Execute `tar -zxfv *.tar.gz` or `unzip -v *.zip` (depending on which 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 archive file of the stable version was downloaded) or `AtChem2-master` (if the archive file of the development version was downloaded). This directory is the AtChem2 ***`Main Directory`***. In this manual, the ***`Main Directory`*** is assumed to be `$HOME/AtChem2/`, but it can be given any name and path – depending on the user's preferences.

2.3 Dependencies

AtChem2 has a number of dependencies (additional software tools and libraries): some are *required*, and without them the model cannot be installed or used, others are *optional*, and only needed to run the Testsuite (Sect. 5.2).

It is recommended to install all the dependencies into the same directory, called *`Dependencies Directory`*. In this manual, the *`Dependencies Directory`* is assumed to be `$HOME/AtChem-lib/`, but it can be given any name and path – depending on the user's preferences.

Before installing the dependencies, make sure that the requirements listed in Sect. 2.1 are met, and that the unix shell is bash or bash-compatible (type `echo $SHELL` at the terminal to check the shell type).

2.3.1 Required dependencies

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 script `install_cvode.sh` 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 the text editor:
   - ◇ If using a compiler other than `gcc`, change the line:
     `-DCMAKE_C_COMPILER:FILEPATH=gcc \`
3. Run the installation script (change the path to the *`Dependencies Directory`* as needed):

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

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

openlibm

openlibm is a portable implementation of the open source mathematical library libm. 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 0.8.1 and it can be installed using the script `install_openlibm.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 ***Dependencies Directory*** as needed):

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

Warning: the compilation of openlibm on **macOS** may fail with a message concerning the `dyld` library. This is analogous to the error that may occur the first time that AtChem2 is run on macOS (see below, in Sect. 2.4) and can be resolved in a similar way.

If the installation is successful, there is now a working openlibm installation at `~/AtChem-lib/openlibm-0.8.1/`. Take note of the openlibm library path, 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 to compare text files containing numerical fields. It is needed only to run the Testsuite, a battery of tests used during the development of AtChem2 to ensure that the model works properly and that changes to the source code do not result in unintended behaviour. Installation of numdiff is recommended for users who want to contribute to the development of AtChem2, but otherwise optional.

Use `which numdiff` to check if the program is already installed on the operating system. If not, ask the system administrator. Alternatively, numdiff can be installed locally (e.g., in the ***Dependencies 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 ***Dependencies 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 the system configuration) with a text editor. Add the following line to the bottom of the file (change the path to the *`Dependencies 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 an open source testing library for Fortran. It requires Ruby [2] and it is needed only to run the unit tests (Sect. 5.2). Installation of FRUIT is recommended for users who want to contribute to the development of AtChem2.

The current version of FRUIT is 3.4.3 and it can be installed using the script `install_fruit.sh` 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 the system 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 *`Dependencies Directory`* as needed):

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

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

[2] The current version of FRUIT does not compile with Ruby v3.2 due to some deprecated syntax. Version 3.0 or earlier must be used until the FRUIT developers release an update.

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 (if installed) 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 `$CVODELIBDIR`, `$OPENLIBMDIR`, `$FRUITDIR` to the paths of CVODE, openlibm and FRUIT, as indicated in Sect. 2.3.1. Use *full paths*, because relative paths may cause compilation errors [3]. For example (change the path to the ***Dependencies Directory*** as needed):

   ```
   CVODELIBDIR = $(HOME)/AtChem-lib/cvode/lib
   OPENLIBMDIR = $(HOME)/AtChem-lib/openlibm-0.8.1
   FRUITDIR = $(HOME)/AtChem-lib/fruit_3.4.3
   ```

   If FRUIT has not been installed (it is optional and needed only to run certain tests), leave the default value for `$FRUITDIR`.

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

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

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

5. Execute `./atchem2` from the ***Main Directory***. The command runs the model executable using the default chemical mechanism and settings, which are in the `model/` directory. The executable prints information to the terminal while it is running, such as the model settings, warning messages (if any), and timesteps. If the model run completes successfully, the terminal will display a message like this:

[3]See issue #364.

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

Warning: when AtChem2 is run for the first time on a **macOS** system, it may abort with the following message:

```
dyld: Library not loaded: @rpath/libsundials_cvode.2.dylib
```

This error is related to the path of dynamic libraries on macOS: to resolve the issue, follow the suggestions on the wiki.

AtChem2 is now ready to be used. Optionally, the Testsuite can be run to check the model installation: go to Sect. 2.4.1 for more information. The directory structure of AtChem2 is described in Sect. 2.6 (see also Fig. 1.1). For detailed instructions on how to set up, configure, build and run an AtChem2 box-model, go to Chapt. 3 and Chapt. 4.

2.4.1 Tests (optional)

The Testsuite can be used to verify that AtChem2 has been installed correctly and works as intended. This step is not necessary, but it is recommended especially for users who want to contribute to the development of the AtChem2 model. Note that the optional dependencies have to be installed to run the Testsuite.

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 modeltests`: runs only the Model 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. The logfiles of the test run can be found in the `tests/` directory.

2.5 Docker

As an alternative to the installation of AtChem2 described above, a containerised version of the model is available. Currently this is built only for version 1.2.2. The container is built on Rocky Linux 8.9 and pre-installs the CVODE and openlibm dependencies via the corresponding install scripts (Sect. 2.3.1). The image can be downloaded via the command:

```
docker pull ghcr.io/wacl-york/atchem2:1.2.2
```

When running the container, changes to the model (e.g., those made to configurations, constraints and mechanisms) should be in a folder that matches the AtChem2 directory structure. This folder is then mounted as a volume to the container with the name `/data_transfer/`. The chemical mechanism to use is provided as a positional argument to the image (e.g., `./model/my_mech.fac`).

Example host file structure:

```
my_model_run
 |__ model
     |__ configuration
     |__ constraints
     |__ my_mech.fac
```

Example Docker run command:

```
docker run -it --rm -v /path/to/my_model_run:/data_transfer
       ghcr.io/wacl-york/atchem2:1.2.2 ./model/my_mech.fac
```

Outputs will be copied to `my_model_run/model/output` on completion.

Running on Singularity / Apptainer

Some HPC systems use Singularity / Apptainer instead of Docker as their container engine. This image is compatible with those as well. The image can be converted to a `.sif` via the command:

```
apptainer pull path/to/image/atchem2.sif
          docker://ghcr.io/wacl-york/atchem2:1.2.2
```

Example Apptainer run command:

```
apptainer run --bind /path/to/my_model_run/:/data_transfer/
          path/to/image/atchem2.sif ./model/my_mech.fac
```

Note that the leading `/` is important when mounting the volume for apptainer, as the container opens in the users directory whereas Docker opens at `/`.

2.6 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 Testsuite.

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.

Table 2.1: Directory structure of AtChem2.
(†) Not present in AtChem-online (‡) Since version 1.2.2

Version 1.0 and earlier	Version 1.1 and later	Description
–	`build/`	scripts to build the model
–	`doc/`	user manual and other documents
–	`docker/` (‡)	Docker-related files
–	`mcm/`	MCM data files and example mechanism files
`modelConfiguration/`	`model/configuration/`	chemical mechanism, shared library, model and solver configuration files
`speciesConstraints/`	`model/constraints/species/`	model constraint files: chemical species
`environmentConstraints/`	`model/constraints/environment/`	model constraint files: environment variables
`environmentConstraints/`	`model/constraints/photolysis/`	model constraint files: photolysis rates, JFAC
`modelOutput/`	`model/output/`	model results: chemical species, environment variables, photolysis rates, production/loss rates of selected species, diagnostic variables
`instantaneousRates/`	`model/output/reactionRates/`	model results: reaction rates of every reaction in the chemical mechanism
`obj/`	`obj/`	module and object files generated by the Fortran compiler
`src/`	`src/`	Fortran source files
`travis/` (†)	`tests/` (‡)	Testsuite scripts, logs and files
`tools/`	`tools/`	installation scripts, plotting tools and other utilities

In AtChem2 v1.1 (and later versions) the directories `build/`, `mcm/`, `obj/` and `src/` contain the build scripts, the MCM data files, the files generated by the Fortran compiler and the Fortran source code. The directory `tests/` contains the files and scripts necessary to run the Testsuite; the directory `docker/` contains the scripts necessary to run AtChem2 as a Docker container.

For the majority of the users, the most important directories are `doc/` which contains the user manual, `tools/` which contains the installation and plotting scripts, and `model/` which contains the model configuration, input, output and (usually) the chemical mechanism. Detailed information on these three directories are in the following sections.

2.6.1 The `doc/` and `tools/` directories

The `doc/` directory contains 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 the script used to update the version number of AtChem2 in each development cycle (`update_version_number.sh`), two scripts to check the consistency of the Fortran code (`fix_style.py` and `fix_indent.py`, see Sect. 5.3), 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 with the release of 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., R, Python, Igor, MATLAB, Origin, IDL, etc. . .) to process and analyze the model results.

The plotting scripts are written in different programming languages: gnuplot, Octave [4], Python, R. One or more of these environments is probably already installed on the operating system: check the local documentation or ask the system administrator. All plotting scripts require one argument – the path of the directory containing the model output – and produce the same output: one file called `atchem2_output.pdf` in the model output 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/`

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

- ◇ `python ./tools/plot/plot-atchem2-pandas.py ./model/output/`
- ◇ `Rscript --vanilla ./tools/plot/plot-atchem2.r ./model/output/`

2.6.2 The `model/` directory

The `model/` directory is the most important from the point of view of the user. All the information required to set up and run a box-model with AtChem2, together with the results of the model run, is contained in this directory.

The `model/` directory includes the model configuration files, the model constraints, and the model output. In principle, the chemical mechanism file (`.fac`) can be located in any directory, although it is recommended to keep it in the `model/` directory with the rest of the configuration files.

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 chemical mechanism file are given as arguments to the build script (`build/build_atchem2.sh`) and to the `atchem2` executable, as explained in Sect. 4.3 and 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 to recompile the source code and create a different executable every time. Batch model runs are therefore easy to do, since all the components of the model that have to be modified are contained in the same directory. Further information can be found in Sect. 3.1.5.

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. Historically, this software and the format have been closely associated with the Master Chemical Mechanism. The extraction tool on the MCM website generates `.fac` files that can be directly used in AtChem2 (Sect. 3.1.3).

Since version 1.2.3, AtChem2 also supports the **KPP format** for chemical mechanisms. The `.kpp` mechanism file, which should have a similar format as the KPP files generated by the MCM extraction tool, is automatically converted to a `.fac` file (Sect. 4.1.1).

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. The definition of a chemical reaction in FACSIMILE starts with the `%` character and ends with the `;` character. The `:` character separates the rate coefficient from the reactants and products.

AtChem2 accepts stoichiometric coefficients in reaction definitions since version 1.2.3. The coefficients must be written before a species name and can be integer or decimal values, with or without a following space. For example, the following reaction definitions are all equivalent:

```
% k : A + A = B + B + C ;
% k : 2A = 2B + C ;
% k : 2.0A = 2.0B + C ;
% k : 2 A = 2 B + C ;
% k : 2.0 A = 2.0 B + C ;
% k : A + A = 2B + 1.0 C ;
```

Important Note: the names of the rate coefficients and of the chemical species (products and reactants) must begin with a letter and not with a numerical character, although numbers may occur at other points of the variable name. This is to avoid confusion with the stoichiometric coefficients.

The rate coefficient of a chemical reaction (`k`) can be a constant number or, more commonly, is calculated as a function of other variables: e.g., 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 build 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 ;
```

In the example above, the photolysis rate of NO_2 (`J_NO2`) can be calculated by AtChem2 as function of latitude, longitude and solar zenith angle, or can be provided by the user (Sect. 3.5).

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 or dissociation reactions). For example:

```
* Formation of nitric acid (HNO3) in the gas-phase ;
*;
* Rate coefficient (Atkinson et al., ACP, 2004) ;
K8O = 3.3D-30*M*(TEMP/300)@-3.0 ;
K8I = 4.1D-11 ;
KR8 = K8O/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 also be written in FACSIMILE format without reactants or products. This feature can be used to implement simple descriptions of non-chemical processes in a box-model. For example, dilution (Sect. 3.4.8), surface deposition and direct emission of a chemical species can be parametrized as:

```
* Deposition velocity of O3 = 1.4 cm s-1 ;
* Boundary layer height (BLHEIGHT) in cm ;
% 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 by defining complex mathematical expressions to calculate the corresponding “rate coefficients”, as explained above. Alternatively, since version 1.2.3, AtChem2 allows the user to create custom Fortran functions: for detailed instructions, go to Sect. 3.1.4.

During the build process, the chemical mechanism file has to be parsed by the `mech_converter.py` script, which expects the chemical mechanism file to have four sections:

Generic rate coefficients : contains the equations of the generic rate coefficients, typically used when experimental or theoretical kinetic data are not available.

Complex reactions : contains the equations of the complex rate coefficients (e.g., for combination/dissociation reactions or parametrized non-chemical processes).

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

Reaction definitions : contains the definitions of the chemical reactions and of the parametrized non-chemical processes (e.g., aerosol uptake, dry deposition, etc…).

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). Minimal chemical mechanism files (`mechanism_skel.*`) can be found in the `mcm/` directory for reference. In addition, an example chemical mechanism downloaded from the MCM website (`mechanism.fac`) is included in the `model/` directory.

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 primarily designed to run box 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 `.fac` file 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, which is included in the `mcm/` directory. By default, AtChem2 uses the MCM v3.3.1; the corresponding list of organic peroxy radicals is in the `peroxy-radicals_v3.3.1` file. Lists of organic peroxy radicals for other versions of the MCM are included in the `mcm/` directory; instructions on how to change the MCM version can be found in the `mcm/INFO.md` file.

If one or more chemical species are 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:

```
      !!! WARNING !!!
  The following species are not present in the RO2 reference list:
    speciesX speciesY
  Should they be included in the RO2 sum?
```

In this case, the user must check manually whether the species in question are organic peroxy radicals, amend the **Peroxy radicals** section of the `.fac` file accordingly, and rerun the `build_atchem2.sh` script.

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 all the organic peroxy radicals in the chemical mechanism must be included in the RO_2 sum, and that the RO_2 sum must include only organic peroxy radicals [1]. The model calculates the RO_2 sum at runtime and, by default, outputs the value of this variable to the `environmentVariables.output` file (Sect. 4.5).

3.1.3 MCM extraction

The MCM website provides a convenient tool to download the whole Master Chemical Mechanism, or subsets of it, in FACSIMILE format. A brief overview of the process is given here; more information can be found on the MCM website.

First, select the species of interest using the MCM browser and add them to the "Marklist". Then proceed to the MCM export tool and select the option "FACSIMILE" as output format. Make sure to also select the following options, so that all the required headers (Sect. 3.1.1) are included in the generated `.fac` file: :

```
[x] Include inorganic reactions?
[x] Include generic rate coefficients?
```

Click on the "Download" button and save the `.fac` file into a directory of choice – such as the `model/` directory (Sect. 2.6.2). The procedure to generate the chemical mechanism file is identical if a `.kpp` file is desired (just select the option "KPP" instead of "FACSIMILE" as output format). The chemical mechanism file downloaded from the MCM website is a simple ASCII text file and is ready to be used in AtChem2. If modifications are required (e.g., chemical reactions to be added, deleted or modified), the user can open the file with a text editor and modify the chemical mechanism as needed.

[1]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.

3.1.4 Custom Functions

The Fortran file `customRateFuncs.f90` allows the user to define custom functions that can be called from within the chemical mechanism to enhance the capability of the model [2]. These functions can be used to parametrize heterogeneous reactions, aerosol formation, surface deposition, emissions and other non-chemical processes, as discussed in Sect. 3.1.1.

By default, `customRateFuncs.f90` is located in `model/configuration/` and contains an empty Fortran module which can be modified by the user to add any number of custom functions. During the build process, the `mech_converter.py` script identifies the names of all the custom functions added to the module and includes them into the `mechanism.f90` file (Sect. 3.1.5). If no custom functions have been defined, the content of the module is ignored.

Note that the custom functions must be defined as Fortran functions and not as subroutines, i.e. the Fortran keyword **function** must be used in the declaration. Environment variables (Sect. 3.4), such as temperature and pressure, can be used as arguments in a custom function. Below is an example of a modified `customRateFuncs.f90` file which defines a basic function to add two numbers:

```
module custom_functions_mod
  implicit none

contains

  ! Function to multiply two numbers
  pure function multiplyNumbers( inNum1, inNum2 ) result ( outNum )
    real, intent(in) :: inNum2, inNum1
    real :: outNum

    outNum = inNum1 * inNum2

    return
  end function multiplyNumbers

end module custom_functions_mod
```

The “multiplyNumbers()” function can then be called in the `.fac` file, together with the regular FACSIMILE notation (Sect. 3.1.1). For example:

```
% multiplyNumbers(2.0, 0.5) : A = B ;
```

[2] *N.B.*: this feature is available only in AtChem2 v1.2.3 and later.

It is the user's responsibility to ensure that the custom Fortran code is correct. Optionally, but recommended, the Python scripts `fix_style.py` and `fix_indent.py` in the `tools/` directory can be used to check that the Fortran code follows the AtChem2 coding guidelines, which are described in Sect. 5.3.

3.1.5 Build process

AtChem2 is built using the scripts in the `build/` directory. Here, the build process is only outlined; more detailed instructions can be found in Sect. 4.3.

The Python script `mech_converter.py` – called by the `build_atchem2.sh` script – converts the chemical mechanism into a Fortran-compatible format. In doing so, the build script generates a number of files:

- `mechanism.f90` contains the Fortran equations to calculate the rate coefficients of each reaction in the chemical mechanism.
- `mechanism.o` is the compiled Fortran code and `mechanism.so` is the shared library, i.e. the pre-compiled chemical mechanism.
- `mechanism.species` contains the 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) of each reaction in the chemical mechanism. The files have a one line header with the total number of species and reactions in the mechanism, and the total number of equations in the **Generic rate coefficients** and **Complex reactions** sections of the `.fac` file (Sect. 3.1.1). The first column is the ID number of the reaction, the second column is the ID number of the species (from `mechanism.species`) which is a reactant/product in that reaction, the third column is the stoichiometric coefficient of the reaction:

  ```
  29 71 139 numberOfSpecies numberOfReactions numberOfGenericComplex
  1 1 1.0
  2 1 1.0
  3 1 1.0
  3 2 1.0
  ```

- ◇ `mechanism.ro2` contains the organic peroxy radicals (RO_2) in the chemical mechanism. The file has a one line header 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 automatically generated by
  build/mech_converter.py -- 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 `build_atchem2.sh`, as explained in Sect. 4.3 (see also Sect. 2.6.2). The shared library (`mechanism.so`) is created in the ***`Shared Library Directory`***, which is the same as the model configuration directory unless otherwise specified.

3.2 Model Parameters

The model parameters control the general setup of the model, such as the runtime, the interpolation methods, the output frequencies, etc... The model parameters, which are described below, are set in the `model.parameters` file (by default, located in the `model/configuration/` directory).

- ◇ **number of steps** and **step size**. The model runtime 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. Note that the step size is *distinct* from the integration step, which is independently 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: $24 \times 300 = 7200$ seconds = 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 production and loss rates of selected species – listed in `outputRates.config` (Sect. 3.6.3). This is used for the rate of production and destruction analysis (ROPA/RODA, Sect. 4.5.1).

- ◇ **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** does not need to be specified: it is automatically calculated by the model as:

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

  Note that when one or more variables are constrained, the time interval between the model start time and stop time *must be* equal to, or less than, the time interval of the constrained data – go to Sect. 4.2.3 for more information.

- ◇ **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 is not saved. 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. Therefore it is recommended to output the Jacobian matrix only if needed.

- ◇ **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 (i.e. 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, they are saved in the `reactionRates/` subdirectory of `model/output` (Sect. 4.5). Note that this parameter is

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

different from **rates output step size** (see above), which sets the output frequency for the production and loss rates of a limited number of species of interest.

3.3 Solver Parameters

The solver parameters control the behaviour of the ordinary differential equations (ODE) solver, and are set in the `solver.parameters` file (by default, located 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, 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 value, i.e. a numerical or logical constant, or can be constrained to a timeseries of measured values

(`CONSTRAINED`), depending on their type. If a variable is constrained, 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 either 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
  ↪ default: 288.15
- ◇ constrained

3.4.2 PRESS

Ambient Pressure (mbar).

- ◇ fixed value
  ↪ default: 1013.25
- ◇ constrained

3.4.3 RH

Relative Humidity (%). It is required only if `H2O` (see below) 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` (see above) must be set to a fixed value or `CONSTRAINED`.

- ◇ fixed value
  ↪ default: 4.45e+17
- ◇ constrained
- ◇ calculated

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: `CALC`

3.4.6 BLHEIGHT

Boundary Layer Height. It is required only if the model includes non-chemical processes such as the emission or deposition of chemical species. The unit is centimetre (cm) or metre (m), 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 ASA

Aerosol Surface Area. It is required only if the model includes heterogeneous chemical reactions. The unit is area (e.g., cm^2, μm^2, nm^2) per volume (e.g., m^3, cm^3), depending on how these reactions are implemented in the chemical mechanism.

- ◇ fixed value
- ◇ constrained
- ◇ not used
  ↪ default: `NOTUSED` (-1)

3.4.8 DILUTE

Dilution Rate (s^{-1}). It is required only if the model includes the dilution of chemical species.

- ◇ fixed value
- ◇ constrained

◇ not used
↪ default: `NOTUSED` (-1)

When `DILUTE` is set to a fixed value or constrained, 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 [4]. 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 = ;
```

Note that AtChem2 treats molecular oxygen (O_2) and nitrogen (N_2) as model parameters rather than chemical species. Their concentrations are calculated by the model as a function of temperature and pressure and, therefore, `DILUTE` is not applied to O_2 and N_2.

3.4.9 JFAC

Correction factor used to adjust the photolysis rates – e.g., to account for the effect 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 detailed information on this parameter, go to Sect. 3.5.4.

◇ fixed value
↪ default: 1

◇ constrained

◇ calculated

3.4.10 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

[4]The behaviour of `DILUTE` was changed with the release of version 1.2. Previously, the user had to manually add to the chemical mechanism a "reaction" using `DILUTE` for each species for which dilution was required. The new approach makes the use of `DILUTE` both more intuitive and more accurate.

constant or constrained: this is different than setting `JFAC` to `0`, which only applies to the calculated photolysis rates (Fig. 3.1).

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

3.5 Photolysis Rates

The photolysis rates are the rate coefficients of the photolysis reactions, and are indicated 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, as per MCM protocol). 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 to a constant value, 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 photolysis 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. 3.4.10.

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 fixed value (in s^{-1}) in the `photolysisConstant.config` file, otherwise they are automatically set to zero (Fig. 3.1). This approach allows the user to model individual photolytic 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 constrained or calculated by the model as explained below.

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

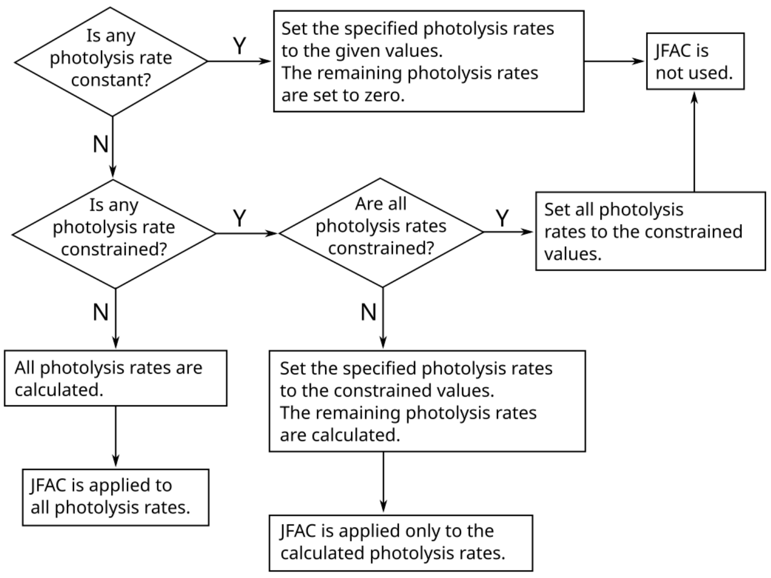

Is any photolysis rate constant?
Y
Set the specified photolysis rates to the given values. The remaining photolysis rates are set to zero.
JFAC is not used.
N
Is any photolysis rate constrained?
Y
Are all photolysis rates constrained?
Y
Set all photolysis rates to the constrained values.
N
All photolysis rates are calculated.
N
Set the specified photolysis rates to the constrained values. The remaining photolysis rates are calculated.
JFAC is applied to all photolysis rates.
JFAC is applied only to the calculated photolysis rates.

Figure 3.1: Photolysis rates and `JFAC` in AtChem2.

`photolysisConstrained.config` file (Sect. 3.6.6); a corresponding file with the constraint data must be present in the `model/constraints/photolysis/` directory (see Sect. 4.2.1 for details).

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. not listed in `photolysisConstrained.config`) are calculated by the model 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`) is calculated as a function of the solar zenith angle according to the following 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, which means there is perfect transmittance of light through the chamber walls or windows. For more information, see the photolysis page on the MCM website.

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 MCM versions 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 and time of the day. These variables are set by the user in the `model.parameters` file (Sect. 3.2). The sun declination (`DEC`, Sect. 3.4.5) – unless it is constrained or set to a constant value – is calculated by the model. The calculations of the solar zenith angle and of the sun declination 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, aerosol, etc... [Sommariva et al., 2020]. This information is retained in the constrained photolysis rates, but it is not included in the calculated ones. To take into account the ambient variability, the calculated photolysis rates can be scaled by a constant or time-dependent correction factor – the environment variable `JFAC` (Sect. 3.4.9) – defined as:

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

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 reference because $j(NO_2)$ is one of the most frequently measured photolysis rates, in which case: `JFAC = j(NO2)/J4`.

The parameter `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. 3.4.9), 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).

AtChem2 can calculate `JFAC` automatically 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`). The reference photolysis rate must also be constrained – i.e. it must be listed in the `photolysisConstrained.config` file (Sect. 3.6.6) and the corresponding constraint file must be present in the `model/constraints/photolysis/` directory [5].

[5]The calculation of `JFAC` at runtime does not work very well in the current version

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 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, respectively.

Usually, the `model/configuration/` directory is also the same as the *`Shared Library Directory`*, which contains the pre-compiled chemical mechanism generated during the build process. The names and paths of these directories can be modified by the user, as explained in Sect. 2.6.2. The content and the format of each `.config` file are described below.

3.6.1 environmentVariables.config

This file specifies the settings of the environment variables, i.e. the physical parameters of the model. The file has three columns: the first two are the ID number and the name of the environment variable. The third column – the only one that should be modified by the user – contains the corresponding value. The possible values of each environment variable are detailed in Sect. 3.4. 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
10  ASA          2e-5
```

If an environment variable is constrained, there must be a corresponding data file in the `model/constraints/environment/` directory, except for `JFAC` whose data file should go in the `model/constraints/photolysis/` directory. For more information, go to Sect. 4.2.1.

of AtChem2, especially in situations when the reference photolysis rate is highly variable and at sunrise/sunset. Therefore, it is recommended to calculate `JFAC` offline and then to constrain it (see issue #16).

3.6.2 initialConcentrations.config

This file specifies the initial concentrations (in molecule cm^{-3}) of the chemical species. The file has two columns: the first column is the name of the initialized species, the second column is the corresponding concentration at the **model start time**, which is defined in the `model.parameters` file (Sect. 3.2). 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 assumed to have an initial concentration of zero molecule cm^{-3}. It is also not necessary to initialize the chemical species that are set to constant values or constrained to measured concentrations, i.e. those listed in `speciesConstant.config` and/or `speciesConstrained.config` (see below).

3.6.3 outputRates.config

This file lists the chemical species for which detailed production/loss rates are required [6]. The file has one column, with one species per line. For example:

```
OH
HO2
NO3
```

The frequency of this output is controlled by **rates output step size** in the `model.parameters` file (Sect. 3.2). The corresponding output files – called `productionRates.output` and `lossRates.output` – are formatted to facilitate the rate of production and destruction analysis of selected species. For more information, go to Sect. 4.5.1.

3.6.4 outputSpecies.config

This file [7] 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:

[6]In version 1.0 and earlier, these species were listed in two separate files: `productionRatesOutput.config`, `lossRatesOutput.config`.

[7]Called `concentrationOutput.config` in version 1.0 and earlier.

```
O3
NO2
CH4
OH
HO2
```

The frequency of this output is controlled by the **step size** parameter in the `model.parameters` file (Sect. 3.2). Constrained chemical species may be listed in `outputSpecies.config`, which can be useful for diagnostic and debugging purposes. Note that the photolysis rates, the environment variables, H_2O, and the RO_2 sum are always output by the model (Sect. 4.5). Therefore, there is not a `.config` file to set the output of these variables and they *should not* be listed in `outputSpecies.config`.

3.6.5 photolysisConstant.config

This file lists the photolysis rates that are set to constant values (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 [8] 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.

[8]Called `constrainedPhotoRates.config` in version 1.0 and earlier.

3.6.7 speciesConstant.config

This file [9] lists the chemical species that are set to constant concentrations. 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 [10] 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.

[9]Called `constrainedFixedSpecies.config` in version 1.0 and earlier.
[10]Called `constrainedSpecies.config` in version 1.0 and earlier.

Chapter 4

Model Execution

4.1 Box-Model

AtChem2 is a modelling tool to build and run chemical box-models for atmospheric chemistry and air quality studies [Sommariva et al., 2020]. An AtChem2 box-model requires two sets of inputs, which are provided by the user: the chemical mechanism file and the configuration files – both are described in the following sections. Additionally, AtChem2 gives the option to constrain a box-model to observations (see Sect. 4.2), which requires the user to provide constraint data files.

4.1.1 Mechanism file

AtChem2 is designed to use the Master Chemical Mechanism (MCM). Other chemical mechanisms can be used, as long as they are in the right format. By default, AtChem2 uses the FACSIMILE format, which is described in Sect. 3.1.1: it must be provided by the user as a text file with the extension `.fac`.

AtChem2 also accepts chemical mechanisms in KPP format – since version 1.2.3: in this case, the `.kpp` file is automatically converted by the build scripts into a `.fac` file, which is then processed as any other FACSIMILE formatted file. The mechanism file (`.fac` or `.kpp`) can be downloaded from the MCM website, as explained in Sect. 3.1.3, or it can be assembled manually. The user can modify the `.fac` file with a text editor, if needed.

During the build process, the `.fac` file is converted into a pre-compiled shared library (`mechanism.so`, in the *`Shared Library Directory`*), plus several files in Fortran-compatible format (in the `model/configuration/` directory). Note that, by default, the *`Shared Library Directory`* is the same as the model configuration directory, although this can be changed by the user.

4.1.2 Configuration files

The configuration of the box-model is set via a number of text files, which can be modified 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

An AtChem2 box-model can be set up 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 from the initial conditions and from the constrained variables.

The constraint data must be provided as one text file for each constrained variable, with the format described below. By default, the constraint data files are located in `model/constraints/species/` for the chemical species, `model/constraints/environment/` for the environment variables, `model/constraints/photolysis/` for the photolysis rates. Note that, although `JFAC` is an environment variable (Sect. 3.4), its constraint file must be located in the same directory as the photolysis rates.

The default directories can be changed by the user, as explained in Sect. 4.3 – see also Sect. 2.6.2.

4.2.1 Constrained variables

Environment variables

All environment variables, except `ROOF`, can be constrained. To do so, set the variable to `CONSTRAINED` in `environmentVariables.config` (Sect. 3.6.1) 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`.

Chemical species

Any chemical species in the chemical mechanism can be constrained. To do so, add the name of the species to `speciesConstrained.config` (Sect. 3.6.8) 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`.

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` (Sect. 3.6.6) 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`.

Important Note: the filenames of all constraint data files must match the variable name (including capitalization), without any extension.

4.2.2 Constraint files

All constraint data files are text files with two columns and no header: the first column is the time in seconds from midnight of the start date (in UTC, Sect. 3.2), while the second column is the value of the variable in the appropriate unit. For chemical species the unit is molecule cm^{-3}, for photolysis rates the unit is s^{-1}; for the units of environment variables, see Sect. 3.4. For example:

```
-1800  71.48
-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
```

The time in the first column of a constraint file does not have to be at regular intervals and can be negative. AtChem2 interprets negative times as “seconds before midnight of the start date”. Data points with negative time can be useful to ensure the correct interpolation of the variables at the beginning and at the end of the model run. This is because the constrained data must cover the same amount of time, and preferably more, as the intended model runtime to avoid interpolation errors. For details, go to 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 constraints can be used with their original timescales, or can be averaged so that they are all on the same timescale. Both approaches have advantages and disadvantages in terms of how much pre-processing work is required to prepare the data files, and in terms of model accuracy and integration speed: for a discussion of these issues, see Sommariva et al. [2020]. The choice of approach depends on the specific modelling objectives, as well as the available computing resources.

Whether all the constraints are on the same timescale or not, AtChem2 interpolates between data points using the interpolation methods 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, and only when constrained. This happens every time the `mechanism_rates()` function is called from the `FCVFUN()` function, 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.

The model start and stop times must fall within the time range of all constrained data to prevent interpolation errors or model crashes, as mentioned in Sect. 4.2.2 and illustrated below:

```
                  start            stop
model run           |----------------|
constraint 1    |---------------------------|
constraint 2       |------------------|
constraint 3     |-----------------------|
```

If constraint data are 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. When this situation occurs, a warning is printed to the terminal for all data evaluations outside of the supplied time interval. For example:

```
error in piecewise linear interpolation
 4.3205895E+05 720  3.0000000E+04
error in piecewise linear interpolation
 4.3205895E+05 720  0.0000000E+00
```

Adjusting the **model start time** and/or the **number of steps** in the `model.parameters` file (Sect. 3.2) usually resolve this issue [1]. 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 `build_atchem2.sh` script in the `build/` directory is used to process the chemical mechanism file(Sect. 4.1.1) and compile the model. The script generates five files which describe the chemical mechanism in Fortran-compatible format (`mechanism.*`), and one pre-compiled shared library (`mechanism.so`). The format and content of these files are described in Sect. 3.1.5.

The `build_atchem2.sh` script must be run from the ***`Main Directory`*** and takes three arguments which must be provided in the exact 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 potential mistakes, the user can choose to always specify all arguments. The three arguments, and their default values, are:

1. path to the chemical mechanism file – there is no default, but it is suggested to keep the `.fac` (or `.kpp`) file either in the `model/` or in the `model/configuration/` directories.

2. path to the the directory for the Fortran mechanism files and to the *`Shared Library Directory`* – default: `model/configuration/`.

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

Both full and relative paths are acceptable for all three arguments, according to the user's preference. For example, if the chemical mechanism file is in the `model/` directory, the model is build using the command:

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

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

An installation of AtChem2 can have multiple `model/` directories, each corresponding to a different model or project. This approach allows the user to work with multiple models at the same time and makes it easy to run batch simulations for sensitivity studies. As mentioned in Sect. 2.6.2, the `model/` directory can also be located outside the ***`Main Directory`***, which gives the users great flexibility in the organization of their workflow.

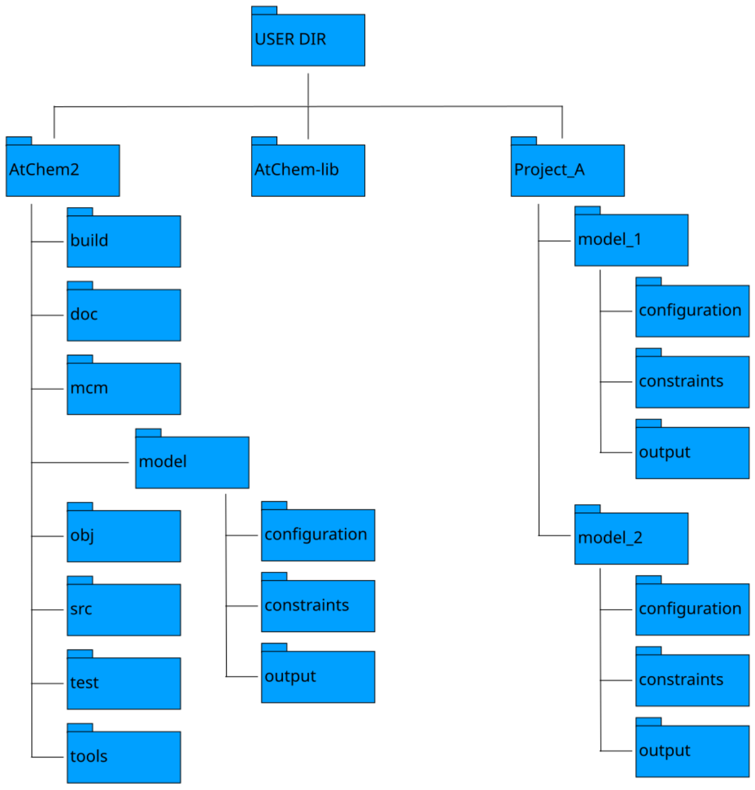


Figure 4.1: Example of a modelling setup, with an AtChem2 installation and a project directory containing two different box-models.

Fig. 4.1 shows an example of a possible setup: a user directory contains an AtChem2 installation, consisting of the ***`Main Directory`*** (`AtChem2/`, which includes the default `model/` directory) and the ***`Dependencies Directory`*** (`AtChem-lib/`). In addition, a project directory (`Project_A/`) contains two separate model directories (`model_1/` and `model_2/`), each with their own chemical mechanism, configuration, constraints and output. The project directory may correspond, for example, to a field campaign or to a set of related experiments. In this case, each model can be built from the ***`Main`***

Directory with the following commands [2]:

```
./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 organized and 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 `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.

The `build_atchem2.sh` script also handles model compilation, converting the Fortran code into the `atchem2` executable, which is dynamically linked to the shared chemical mechanism library (`mechanism.so`). If the Fortran code is not changed, compilation is required only once for a given `.fac` (or `.kpp`) file.

Changes to the configuration files do not require recompiling the model. Likewise, if constraints files are added, removed or modified there is no need to recompile the model. This is because the model configuration and constraints are read by the executable at runtime. However, if the chemical mechanism is changed, then the shared library (`mechanism.so`) needs to be recompiled. However, if the user wants, or needs, to change the Fortran code (`src/*.f90`), then the executable needs to be recompiled.

To recompile the model, the `build_atchem2.sh` script needs to be executed again [3]. The build script is able to detect whether the Fortran code, the chemical mechanism, or both have been modified, and runs the appropriate commands. This design approach reduces both the frequency and the duration of model compilations by requesting it only when strictly necessary, therefore facilitating running the model in batch mode – e.g., for sensitivity studies.

4.4 Execute

The build process – described in Sect. 3.1.5 and Sect. 4.3 – creates an executable file called `atchem2` in the *Main Directory*. The executable takes up to nine arguments, corresponding to the *relative paths* (with respect to

[2]The third argument is not specified in this example, so the default value (`./mcm/`) is implicitly used.

[3]If the recompilation fails, it is sometimes useful to run the command `make clean` from the *Main Directory* before rerunning the build script.

the ***Main Directory***) of the model configuration, the chemical mechanism shared library, the constraint data files, and the model output. The arguments are passed to the executable via a series of input flags [4], each with a default value as described below:

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

In addition, the input flag `--help` displays an help message explaining the usage of the command line arguments and of the input flags.

[4]Prior to version 1.2, AtChem2 used a system of positional arguments instead of the input flags – see the wiki for details.

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 explicitily 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 omitted, the executable assumes the default value, based on a *hierarchical directory structure*. For example, the following command implies 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, constraints and chemical mechanisms. For example, the following commands run two models with the same chemical mechanism and two different configurations, and save the corresponding outputs in separate directories:

```
./atchem2 --configuration=model/configuration_1/
          --output=model/output_1/
          --shared_lib=model/mechanism.so

./atchem2 --configuration=model/configuration_2/
          --output=model/output_2/
          --shared_lib=model/mechanism.so
```

Only three input flags are specified in the above commands, and therefore the executable assumes that the default values for the remaining input flags apply – i.e. the model directory is `model/` and the constraints directory is the same for both model runs (`model/constraints/`.

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

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

While the model is running, diagnostic information is printed to the terminal. A successful model run completes with a message similar to the one shown in Sect. 2.4. Users have the option to redirect the terminal printout to a log file, using standard unix commands.

4.4.1 HPC

AtChem2 can be installed and run on High Performance Computing (HPC) clusters, or supercomputers, which typically run some version of Linux/Unix. This is recommended, especially for models with long runtimes and/or many constraints.

Installation and setup on an HPC system are exactly the same as on a regular workstation, but each system has its own rules and guidelines. Therefore, it is not possible to give specific advice, and users should check the local documentation or ask the system administrator.

On most HPC systems, software execution is managed by a job scheduler, which requires a submission script. The script will allocate the requested resources [5], alert the user that the model run is finished, and save the terminal printout and any other message to log files. The syntax and format of the submission script depend on the type of job scheduler installed on the HPC system. Examples of submission scripts 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. 2.6.2). The frequency of the

[5]AtChem2 does not support parallelization at present; this may change in future versions.

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, environment variables, photolysis rates and diagnostic variables.
- ◇ **rates output step size** for the rate of production and destruction analysis of selected species (Sect. 4.5.1).
- ◇ **reaction rates output step size** 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** [6]:

- ◇ Environment variables and RO_2 sum:
  `environmentVariables.output`
- ◇ Concentrations of selected chemical species – see Sect. 3.6.4:
  `speciesConcentrations.output`
- ◇ Photolysis rates:
  `photolysisRates.output`
- ◇ Latitude, longitude, solar angles and related parameters:
  `photolysisRatesParameters.output`
- ◇ Loss and production rates (ROPA/RODA) of selected chemical species – see Sect. 3.6.3:
  `lossRates.output`, `productionRates.output`
- ◇ Concentrations of all chemical species, except the RO_2 sum, at the end of the model run (**model stop time**):
  `finalModelState.output`
- ◇ Jacobian matrix (optional, see Sect. 3.2):
  `jacobian.output`
- ◇ Error messages and diagnostic variables:
  `errors.output`, `mainSolverParameters.output`

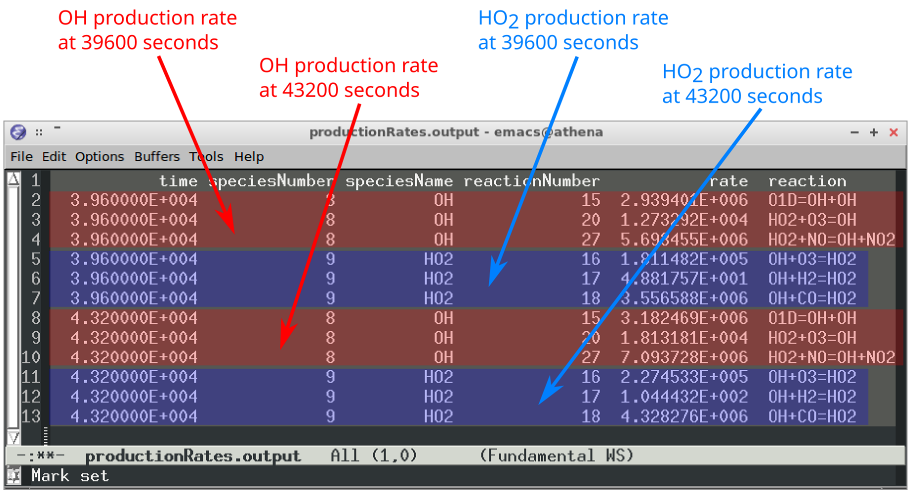
In addition to the `.output` files, the reaction rates of every reaction in the chemical mechanism are saved, by default, in the `output/reactionRates/`

[6]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).

directory [7]: one file for each model step, with the name of the file corresponding to the output time in seconds. The files have no header, and the order of reactionn is the same as in the chemical mechanism (under the "Reaction definitions" section – see Sect. 3.1.1).

4.5.1 ROPA/RODA

The reaction rates files in the `output/reactionRates/` directory are useful to analyze the model results and understand the chemical system, as well as for diagnostic purposes. The format, however, may be cumbersome to process for some users. To simplify the rate of production and destruction analysis for specific chemical species (i.e. those listed in `outputRates.config`, Sect. 3.6.3) AtChem2 creates the two 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, shown in Fig. 4.2.


OH production rate at 39600 seconds
OH production rate at 43200 seconds
HO2 production rate at 39600 seconds
HO2 production rate at 43200 seconds
productionRates.output - emacs@athena
File Edit Options Buffers Tools Help

```
          time speciesNumber speciesName reactionNumber           rate   reaction
  3.960000E+004             8          OH             15  2.939401E+006  O1D=OH+OH
  3.960000E+004             8          OH             20  1.273292E+004  HO2+O3=OH
  3.960000E+004             8          OH             27  5.693455E+006  HO2+NO=OH+NO2
  3.960000E+004             9         HO2             16  1.811482E+005  OH+O3=HO2
  3.960000E+004             9         HO2             17  4.881757E+001  OH+H2=HO2
  3.960000E+004             9         HO2             18  3.556588E+006  OH+CO=HO2
  4.320000E+004             8          OH             15  3.182469E+006  O1D=OH+OH
  4.320000E+004             8          OH             20  1.813181E+004  HO2+O3=OH
  4.320000E+004             8          OH             27  7.093728E+006  HO2+NO=OH+NO2
  4.320000E+004             9         HO2             16  2.274533E+005  OH+O3=HO2
  4.320000E+004             9         HO2             17  1.044432E+002  OH+H2=HO2
  4.320000E+004             9         HO2             18  4.328276E+006  OH+CO=HO2

-:**-  productionRates.output    All (1,0)        (Fundamental WS)
Mark set
```

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

The frequency of the ROPA/RODA output is controlled by the parameter **rates output step size**, as explained in Sect. 3.2 and 3.6.3. The `productionRates.output` and `lossRates.output` files can be very large and complex, especially for species that are involved in many reactions and/or for long model runtimes. Therefore, it is recommended to limit the list of chemical species in `outputRates.config` to only those of special interest, and to keep the frequency of the output low (the default is 1 hour).

[7]Called `instantaneousRates/` prior to version 1.1.

Important Note: although, in principle, a rate of production and destruction analysis for constrained chemical species can be done, the results should be interpreted with extreme caution. The user must be aware that the concentrations of these species are determined by the constraining data rather than by the chemical reactions.

Chapter 5

Model Development

5.1 General Information

AtChem2 is hosted on the GitHub developer platform. Two versions of the model are available in the repository at any given time:

Stable version : is indicated by a version number (e.g., `v1.0`) and can be downloaded from the Releases page. A **doi number** is assigned to each stable version via the Zenodo service.

Development version : is the `master branch` of the git repository and is indicated by a version number with the suffix "-dev" (e.g., `v1.1-dev`). It can be downloaded as an archive file or obtained via **git** – go to Sect. 2.2 for more information.

AtChem2 is under active development, which means that the `master branch` is sometimes a few commits ahead of the latest stable release. However, all code modifications are automatically run through the Testsuite before being merged into the `master branch`. Since the Testsuite is designed to ensure that changes to the code do not cause unintended behaviour or unexplained differences in the model results, this ensures that even the development version of AtChem2 is reliable for most use cases. Nevertheless, it is strongly suggested to use the **stable version** for "production runs" and publications, as it can be easily referenced via the associated doi number – see the discussion about traceability and reproducibility of computational models in Sommariva et al. [2020].

Feedback, bug reports, comments, suggestions, and feature requests are welcome: the list of open issues, known bugs, etc... can be found on the related GitHub page. Please check this page before reporting new issues/bugs. Also note that the AtChem2 wiki contains suggestions to solve some common issues.

Contributions and corrections to the user manual are also welcome. The manual is written in LaTeX format, with figures in `.svg` format. The

`.tex` files, located in the `doc/latex/` directory, can be opened and modified using any text editor. To compile the `.pdf` file of the manual, use the script `tools/make_manual_pdf.sh`, then commit it together with the other changes [1]; alternatively, the `.pdf` file is automatically compiled via GitHub Actions when a Pull Request (PR) is opened.

The preferred way to contribute to the development of AtChem2 is to use the open source git version control tool: instructions on how to set up git, prepare and submit contributions to the AtChem2 repository can be found on the wiki. Contributors are strongly encouraged to follow the Fortran coding guidelines, as detailed in Sect.5.3.

5.2 Testsuite

AtChem2 uses GitHub Actions for continuous integration testing. This programming approach ensures that changes to the code do not unintentionally modify the behaviour and the results of a software. The Testsuite is located in the `tests/` directory [2] and consists of a series of tests and short model runs that check the model functionality and calculations against known outputs. In addition, the Codecov service is used to ensure that the Testsuite covers a significant fraction of the codebase (>90%) and a wide range of common configurations.

There are four types of tests, executed from the ***`Main Directory`*** with the `make` command. Note that the *optional dependencies* must be installed in order to run the Testsuite – see Sect. 2.3.2 and 2.4.1 for details.

- ◇ the **Indent** and **Style** tests check that the indentation and coding style of the Fortran code are consistent with the guidelines:
  $\hookrightarrow$ `make indenttest` and `make styletest`
- ◇ the **Unit** tests check that individual Fortran functions generate the expected outputs:
  $\hookrightarrow$ `make unittests`
- ◇ the **Model** tests run a number of models with different configurations to check that they generate the expected outputs:
  $\hookrightarrow$ `make modeltests` and/or `make oldtests` [3]
- ◇ The command `make alltests` runs all the tests described above in succession.

[1] The script requires a LaTeX installation and ImageMagick (v7.x).

[2] Called `travis/` in versions 1.0-1.2.2, when the TravisCI service was used to run the Testsuite.

[3] At present these tests are split into two groups, `modeltests` and `oldtests`, which will eventually be merged. Both groups of tests pass on Linux, but `modeltests` fail on macOS. See issue #522.

The Indent, Style and Model tests create `.log` files in the `tests/` directory and print a summary message to the terminal, while the Unit tests output the results directly to the terminal. If one or more tests fail, search the log files for detailed information.

The Testsuite is automatically run every time a Pull Request (PR) is created or updated on the GitHub repository. The PR triggers a GitHub Actions workflow which runs the entire Testsuite on two architectures (Linux and macOS) using different versions of the GNU `gfortran` compiler. The workflow – described in the `.github/workflows/ci.yml` file – performs the following tasks on each architecture:

- ◇ Install `gfortran` and the AtChem2 dependencies using `apt-get` (Linux) or `Homebrew` (macOS).
- ◇ Build and run a model using the example chemical mechanism and 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 Model tests: PASS if there are no differences from the reference output files (within tolerance), otherwise FAIL.
- ◇ Build and run the Unit and Model tests, and upload the overage reports to the Codecov service.

The Testsuite is successful only if all tests pass successfully. 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 GitHub Actions has completed with passes on both architectures.

5.2.1 Adding new Unit tests

The Unit tests are in the `tests/unit_tests/` directory and require the FRUIT optional dependency (which needs Ruby v3.0, 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 Model tests

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

```
|- $TESTNAME
|  |- configuration
|  |  |- customRateFuncs.f90          [*]
|  |  |- environmentVariables.config
|  |  |- initialConcentrations.config
|  |  |- mechanism.prod.cmp
|  |  |- mechanism.reac.cmp
|  |  |- mechanism.ro2.cmp
|  |  |- mechanism.species.cmp
|  |  |- model.parameters
|   |   |- outputRates.config
|   |   |- outputSpecies.config
|   |   |- photolysisConstant.config     [*]
|   |   |- photolysisConstrained.config  [*]
|   |   |- solver.parameters
|   |   |- speciesConstant.config        [*]
|   |   |- speciesConstrained.config     [*]
|   |- constraints      [**]
|       |- environment/  [**]
|       |- photolysis/   [**]
|       |- species/      [**]
|- output
|   |- reactionRates/
|   |- environmentVariables.output.cmp
|   |- errors.output.cmp
|   |- finalModelState.output.cmp
|   |- jacobian.output.cmp
|   |- lossRates.output.cmp
|   |- mainSolverParameters.output.cmp
|   |- photolysisRates.output.cmp
|   |- photolysisRatesParameters.output.cmp
|   |- productionRates.output.cmp
|   |- speciesConcentrations.output.cmp
|- $TESTNAME.{fac|kpp}
|- $TESTNAME.out.cmp
```

The files marked with `[*]` and the directories marked with `[**]` are optional, depending on the model 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/` (see Sect. 4.2 for details). The file `$TESTNAME.out.cmp` should contain the exact copy of the expected terminal printout.

Each Model test is briefly described in the `tests/model_tests/INFO.md` file, which should be updated after a new test is added. New tests added to the `tests/tests/` directory are automatically picked up when running the commands `make modeltests` 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 Fortran source code should follow a consistent style (Sect. 5.3.1). Two Python scripts are available to check and, if necessary, 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 `tools/` directory and are executed from the *`Main Directory`* with the following commands:

```
python tools/fix_style.py src/filename.f90
python tools/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 original 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 Testsuite to run the Style and Indent tests (Sect. 5.2): each script is run over each source file and the output is sent to a `.cmp` file. If all `.cmp` files match the original files, 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 an issue 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, compliant with the Fortran 90/95 standard; 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 applicable, 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.` – they 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 `-` signs: e.g., `1.5e-9`.

◇ The names of the **variables** should begin with lowercase, while those of the **procedures** (subroutines and functions) should begin with uppercase. An exception to this rule is **third-party functions**, which should be uppercase. Both CamelCase or underscores are acceptable for multiple-word identifiers, but be consistent.

◇ All **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, Acknowledgements & Funding

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 (in alphabetical order):

- James Allsopp
- Will Drysdale
- Maarten Fabré
- Alfred Mayhew
- Killian Murphy
- Beth Nelson
- Mike Newland
- Marios Panagi

AtChem2 was developed from the original open source codebase of AtChem-online, which was created at the University of Leeds (and is now hosted at the University of York) by:

- Chris Martin
- Kasia Borońska

- Jenny Young
- Peter Jimack
- Mike Pilling

Model evaluation and testing of AtChem-online was done by Andrew Rickard (NCAS/University of 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 (in alphabetical order):

- Bill Bloss
- Peter Bräuer
- Nahid Chowdhury
- Stuart Lacy
- Vasilis Matthaios
- Paul Monks
- Christian Pfrang
- Jon Wakelin
- Rob Woodward-Massey

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

6.3 Funding

Funding has been provided, at different stages, by:

- EUROCHAMP project: https://www.eurochamp.org.
- Met Office: https://www.metoffice.gov.uk.
- National Centre for Atmospheric Science (NCAS): https://www.ncas.ac.uk.
- Natural Environment Research Council (NERC): https://nerc.ukri.org.

- ◇ University of Birmingham, Research Software Group.
- ◇ 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, K. P. Wyche, C. J. Evans, T. Carr, P. S. Monks, M. R. Alfarra, M. H. Barley, G. B. McFiggans, J. C. Young, and A. R. Rickard. Development and chamber evaluation of the MCM v3.2 degradation scheme for β-caryophyllene. *Atmospheric Chemistry and Physics*, 12(11):5275–5308, 2012. doi: 10.5194/acp-12-5275-2012.

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.