AtChem2 Manual

R. Sommariva, S. Cox

April 19, 2019

0.1 Introduction

AtChem2 is a modelling software designed to build and run atmospheric chemistry box-models using the Master Chemical Mechanism (MCM). It can also be used with other chemical mechanisms, as long as they are provided in the FACSIMILE format (see {[}{[}2.1 Chemical Mechanism{]}{]}).

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

The latest stable version of Atchem2 can be found at the releases page. The development version can be downloaded from the master branch or obtained via **git**. To install and run AtChem2 follow the instructions on the {[}{[}installation|1. Installation{]}{]} and {[}{[}dependencies|1.1 Dependencies{]}{]} pages.

AtChem2 is open source, under **MIT license**. For instructions on how to cite the model in publications, see the `CITATION.md` file. The contributors and funders of **AtChem-online** and **AtChem2** are listed in the {[}{[}credits page|Acknowledgements and Credits{]}{]}.

Bug reports, suggestions and contributions are welcome. In order to contribute to the model development, please follow the instructions in the corresponding {[}{[}wiki page|3. Model Development{]}{]}.

0.2 Installation

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

0.2.1 Download

There are two versions of AtChem2: the stable version and the development version, also known as `master branch` (see the [[model development page|3. Model Development]] for additional information). The source code can be obtained in two ways:

1. with **git**:

    1. Open the terminal. Move to the directory where you want to install AtChem2.
    2. Execute `git clone https://github.com/AtChem/AtChem2.git` (if using HTTPS) or `git clone git@github.com:AtChem/AtChem2.git` (if using SSH). This method will download the development version and it is recommended if you want to contribute to the model development.

2. with the **archive file** (`*.tar.gz` or `*.zip`):

    1. Download the archive file of the stable version (https://github.com/AtChem/AtChem2/releases) or of the development version (https://github.com/AtChem/AtChem2/archive/master.zip) to the directory where you want to install AtChem2.
    2. Open the terminal. Move to the directory where you downloaded the archive file.
    3. Unpack the archive file (e.g., `tar -zxvf v1.1.tar.gz` or `unzip master.zip`).

Depending on which of these methods you have used, the source code is now in a directory called `AtChem2` or `AtChem2-1.1` or `AtChem2-master`. This directory - which you can rename, if you want to - is the *main directory* of the model. In the documentation we will assume that the *AtChem2 main directory* is `$HOME/AtChem2`.

0.2.2 Requirements

AtChem2 needs the following tools:

- a Fortran compiler: the model compiles with GNU `gfortran` (version 4.8.5) and with Intel `ifort` (version 17.0)

- Python 2.7.x
- cmake
- Ruby 2.0 (optional)

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

In addition, AtChem2 has the following dependencies:

- the CVODE library
- the openlibm library
- the BLAS and LAPACK libraries
- numdiff (optional)
- FRUIT (optional)

For detailed instructions on the installation and configuration of the dependencies go to: [[1.1 Dependencies]].

0.2.3 Install

To install AtChem2:

1. Move to the *AtChem2 main directory* (`cd ~/AtChem2/`). Install the [[dependencies|1.1 Dependencies]] and take note of the name and path of the *dependencies directory* (in the following instructions, we will assume that the *dependencies directory* is `~/atchem-libraries/`).

2. Copy the `Makefile` in the `tools/` directory to the *main directory* (`cp tools/Makefile ./`).

3. From the the *main directory*, open the `Makefile` with a text editor. Set the variables `CVODELIB`, `OPENLIBMDIR`, `FRUITDIR` to the paths of the CVODE, openlibm and FRUIT libraries, as described in the [[dependencies page|1.1 Dependencies]]. Use the full path to the libraries, not the relative path (see issue #364). For example: `CVODELIB       = $(HOME)/atchem-libraries/cvode`
`OPENLIBMDIR  = $(HOME)/atchem-libraries/openlibm-0.4.1`
`FRUITDIR     = $(HOME)/atchem-libraries/fruit_3.4.3`

4. Execute `./tools/build.sh ./tools/mcm_example.fac`. This command compiles the model and creates an executable (`atchem2`) using the test mechanism file `mcm_example.fac` in the `tools/` directory.

5. Execute `./atchem2`. If the model has been installed correctly, you should see a message similar to this: "' ———————— Final statistics ———————— No. steps = 546 No. f-s = 584 No. J-s = 912 No. LU-s = 56 No. nonlinear iterations = 581 No. nonlinear convergence failures = 0 No. error test failures = 4

   Runtime = 0 Deallocating memory. "'

This means that AtChem2 has completed the test run without errors and is ready to be used. The directory structure of AtChem2 is described [[here|1.2 Model Structure]]. For instructions on how to set up, compile and execute the model go to: [[2. Model Setup and Execution]].

Note for macOS users

When you first run AtChem2, you may receive an error message like this:

```
dyld: Library not loaded: @rpath/libsundials_cvode.2.dylib
Referenced from: /Users/username/AtChem2/./atchem2
Reason: image not found
Abort trap: 6
```

In this case, type at the terminal prompt the following command (change the path to the CVODE library as appropriate):

```
export DYLD_LIBRARY_PATH=$(HOME)/atchem-libraries/cvode/lib
```

To make it permanent, add the command to your `~/.bash_profile` file. Advanced users may wish to use instead the accepted answer in this Stack Overflow post to hardcode `rpath` in this instance for each of `libsundials_cvode.2.dylib`, `libsundials_fvecserial.2.dylib`, `libsundials_vecserial.2.dylib`.

0.2.4 Tests (optional)

You can run the [[Test Suite|3.1 Test Suite]] to verify that AtChem2 has been installed properly and to make sure that changes to the code do not result in unintended behaviour. This is recommended if you want to contribute to the model development. Note that running the Test Suite requires the optional dependencies to be installed, as explained in the [[dependencies page|1.1 Dependencies]].

To run the tests, execute the following commands from the *AtChem2 main directory*: * `make alltests` runs all the tests (requires **numdiff** and **FRUIT**) * `make tests` runs only the behaviour tests (requires **numdiff**) * `make unittests` runs only the unit tests (requires **FRUIT**)

For more information on the Test Suite go to the corresponding [[wiki page|3.1 Test Suite]].

0.3 Dependencies

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

Before installing the dependencies, make sure that Fortran, Python, cmake and (optionally) Ruby are installed on your system, as explained in the [[installation page|1. Installation]].

0.3.1 Required dependencies

BLAS and LAPACK

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

CVODE

AtChem2 uses the CVODE library, which is part of the SUNDIALS suite, to solve the system of ordinary differential equation (ODE). The current version of CVODE is 2.9.0 (part of SUNDIALS 2.7.0) and can be installed using the `install_cvode.sh` script in the `tools/install/` directory.

1. Move to the *AtChem2 main directory* (e.g., `cd ~/AtChem2`).

2. Open the installation script (`tools/install/install_cvode.sh`) with a text editor:

   1. If LAPACK and BLAS are not in the default location on your system (see above), change the `LAPACK_LIBS` variable for your architecture (Linux or macOS), as appropriate.
   2. If you are not using the `gcc` compiler (`gfortran` is part of `gcc`), change the line `-DCMAKE_C_COMPILER:FILEPATH=gcc \` accordingly.

3. From the *AtChem2 main directory*, run the installation script (change the path of the *dependencies directory* as needed): `./tools/install/install_cvode.sh ~/atchem-libraries/`

If the installation is successful, there should be a working CVODE installation at `~/atchem-libraries/cvode/`. The path to the CVODE library is `~/atchem-libraries/cvode/lib/`.

openlibm

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

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

1. Move to the *AtChem2 main directory* (e.g., `cd ~/AtChem2`).
2. Run the installation script (change the path of the *dependencies directory* as needed): `./tools/install/install_openlibm.sh ~/atchem-libraries/`

If the installation is successful, there should be a working openlibm installation at `~/atchem-libraries/openlibm-0.4.1/`.

0.3.2 Optional dependencies

numdiff

numdiff is a program used to compare files containing numerical fields. It is needed only if you want to run the [[Test Suite|3.1 Test Suite]], a series of tests to ensure that the model works properly. Installation of numdiff is recommended if you want to contribute to the development of AtChem2.

Use `which numdiff` to check if the program is already installed on your system. If not, you can install it locally, for example in the *dependencies directory*. Use the script `install_numdiff.sh` in the `tools/install/` directory.

1. Move to the *AtChem2 main directory* (e.g., `cd ~/AtChem2`).
2. Run the installation script (change the path of the *dependencies directory* as needed): `./tools/install/install_numdiff.sh ~/atchem-libraries/numdiff/`
3. Move to your `$HOME` directory (`cd ~`). Open the `.bash_profile` file (or the `.profile` file, depending on your configuration) with a text editor. Add the following line at the bottom of the file (change the path of the *dependencies directory* as needed): `PATH=$PATH:$HOME/atchem-libraries/numdiff/bin`
4. Close the terminal.
5. Open the terminal and execute `which numdiff` to check that the program has been installed correctly.

FRUIT

FRUIT (FORTRAN Unit Test Framework) is a unit test framework for Fortran. It requires Ruby 2.0 and is needed only if you want to run the unit tests in the

[[Test Suite|3.1 Test Suite]]. Installation of FRUIT is recommended if you want to contribute to the development of AtChem2.

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

1. Move to the *AtChem2 main directory* (e.g., `cd ~/AtChem2`).
2. Run the installation script (change the path of the *dependencies directory* as needed): `./tools/install/install_fruit.sh ~/atchem-libraries/`

If the installation is successful, there should be a working FRUIT installation at `~/atchem-libraries/fruit_3.4.3/`.

0.4 Model Structure

AtChem2 is organized in several directories containing the source code, the compilation files, the chemical mechanism, the model configuration and output files, a number of scripts to install and compile the model, plotting tools in various programming languages, and the test suite files.

The directory structure has changed with the release of **version 1.1** (November 2018). The following table shows the new structure and, for reference, the previous one.

v1.0	v1.1	description
main directory	*main directory*	information files (changelog, citation, license, readme) and auxiliary files for the test suite (*N.B.*: the `.gcda` and `.gcno` files are generated by the Fortran compiler during the build process).
–	`mcm/`	data files related to specific versions of the MCM: lists of organic peroxy radicals (RO2), parameters to calculate photolysis rates.
–	`model/`	model files: chemical mechanism (`.fac`), configuration, input, output.
`modelConfiguration/`	`model/configuration/`	model configuration files and mechanism shared library.
–	`model/constraints/`	model constraints.
`environmentConstraints/`	`model/constraints/environment/`	constrained environment variables.
`environmentConstraints/`	`model/constraints/photolysis/`	constrained photolysis rates.
`speciesConstraints/`	`model/constraints/species/`	constrained chemical species.
`modelOutput/`	`model/output/`	model output: chemical species, environment variables and photolysis rates, diagnostic variables, formatted production and loss rates of selected species.
`instantaneousRates/`	`model/output/reactionRates/`	model output: reaction rates of every reaction in the chemical mechanism.
`obj/`	`obj/`	files generated by the Fortran compiler.
`src/`	`src/`	Fortran source files.
–	`src/gen/`	Fortran source files generated by the compiler from the chemical mechanism.

v1.0	v1.1	description
`tools/`	`tools/`	Python and shell scripts to build and compile AtChem2, using the chemical mechanism, the configuration and the constraints in the `model/` directory.
`tools/install/`	`tools/install/`	shell scripts to install the dependencies.
–	`tools/plot/`	scripts to plot the model results (gnuplot, Matlab/Octave, Python, R).
`travis/`	`travis/`	shell scripts to run the test suite.
`travis/tests/`	`travis/tests/`	behaviour tests.
–	`travis/unit_tests/`	unit tests.

The `model/` directory is the most important for the user: it includes the chemical mechanism, the configuration files, the model constraints and the model output. The `model/` directory can be given any name and it can also be located outside of the *AtChem2 main directory*.

There can be multiple `model/` directories (with different names) in the same location. As long as the correct paths are passed to the compilation and execution scripts, the model will compile and run. This approach gives the user the flexibility to run different versions of the same model or different models at the same time. For more information go to: [[2. Model Setup and Execution]].

0.5 Model Setup and Execution

AtChem2 is designed to build and run atmospheric chemistry box-models based upon the Master Chemical Mechanism (MCM, http://mcm.leeds.ac.uk/MCM/). This page explains how to set up, compile and run an atmospheric chemistry box-model with AtChem2. The directory structure of AtChem2 is described [[here|1.2 Model Structure]]. A working knowledge of the **unix shell** and its basic commands is *required* to use the AtChem2 model.

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

0.5.1 Mechanism file

The model requires a chemical mechanism in FACSIMILE format (`.fac`). The **mechanism file** can be downloaded from the MCM website using the extraction tool or assembled manually. The user can modify the `.fac` file as required with a text editor. This mechanism file is converted into a shared library and a set of associated data files in the compilation step below. For more information on the chemical mechanism go to: [[2.1 Chemical Mechanism]].

0.5.2 Configuration files

The **model configuration** is set via a number of text files located in the `model/configuration/` directory. The text files can be modified with a text editor. Detailed information on the configuration files can be found in the corresponding wiki pages:

- model and solver parameters - see [[2.2 Model Parameters]] and [[2.3 Solver Parameters]].
- environment variables - see [[2.4 Environment Variables]].
- photolysis rates - see [[2.5 Photolysis Rates and JFAC]].
- initial concentrations of chemical species and lists of output variables - see [[2.6 Config Files]].

The model constraints - chemical species, environment variables, photolysis rates - are located in the `model/constraints/` directory. For more information, go to: [[2.7 Constraints]].

0.5.3 Compilation

The script `build.sh` in the `tools/` directory is used to process the chemical mechanism file (`.fac`) and to compile the model. The script generates one Fortran file (`mechanism.f90`), one shared library (`mechanism.so`) and four configuration files (`mechanism.prod`, `mechanism.reac`, `mechanism.ro2`, `mechanism.species`) in the `model/configuration/` directory. Go to the [[chemical mechanism page |2.1 Chemical Mechanism]] for more information.

The script must be run from the *AtChem2 main directory* and takes four arguments (see the **Important Note 2** at the end of this section):

1. the path to the chemical mechanism file - no default (suggested: `model/`).
2. the path to the directory for the Fortran files generated from the chemical mechanism - default: `model/configuration/`.
3. the path to the directory with the configuration files - default: `model/configuration/`.
4. the path to the directory with the MCM data files - default: `mcm/`.

For example, if the `.fac` file is in the `model/` directory:

```
./tools/build.sh model/mechanism.fac model/configuration/ model/configura
```

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

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

Each model can be built by passing the correct path to the `build.sh` script (see the **Important Note 1** at the end of this section). For example:

```
./tools/build.sh model_1/mechanism.fac model_1/configuration/ model_1/con
./tools/build.sh model_2/mechanism.fac model_2/configuration/ model_2/con
```

Compilation is required only once for a given `.fac` file. If the user changes the configuration files, there is no need to recompile the model. Likewise, if the constraints files are changed, there is no need to recompile the model. This is because the model configuration and the model constraints are read by the executable at runtime. However, if the user makes changes to the `.fac` file, then the shared library `model/configuration/mechanism.so` needs to be recompiled from the source file `model/configuration/mechanism.f90` using the `build.sh` script.

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

0.5.4 Execution

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

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

The model can be run by executing the `atchem2` command from the *main directory*, in which case the executable will use the default configuration and output directories. Otherwise, the configuration and output directories need to be specified (see the **Important Note 2** at the end of this section).

For example, if the constraints are in the default directories (or not used), the model can be run by executing:

```
./atchem2 model/output/ model/output/reactionRates/ model/configuration/
```

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

```
./atchem2 model_1/output/ model_1/output/reactionRates/ model_1/confi
./atchem2 model_2/output/ model_2/output/reactionRates/ model_2/confi
```

Important Note 1

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

Important Note 2

The arguments need to be passed to the `atchem2` executable in the exact order, as listed above. This means that if - for example - the third argument needs to be specified, it is also necessary to specify the first and the second arguments, even if they have the default values. To avoid mistakes, the user can choose to always specify all the arguments. This behaviour also applies to the `tools/build.sh` script used to compile the model. Future versions of AtChem2 will adopt a simpler command-line interface.

0.5.5 Output

The model output is saved by default in the directory `model/output/`. The location can be modified by changing the arguments of the `atchem2` executable (see above).

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

- values of environment variables and concentrations of chemical species: `environmentVariables.` `speciesConcentrations.output`.
- values of photolysis rates and related parameters: `photolysisRates.output`, `photolysisRatesParameters.output`.
- loss and production rates of selected species (see [[2.6 Config Files]]): `lossRates.output`, `productionRates.output`.
- Jacobian matrix (if requested, see [[2.2 Model Parameters]]): `jacobian.output`.
- model diagnostic variables: `finalModelState.output`, `initialConditionsSetting.out` `mainSolverParameters.output`.

In addition, the reaction rates of all the reactions in the chemical mechanism are saved in the directory `reactionRates/`: one file for each model step, with the filename corresponding to the time in seconds.

0.5.6 Running on HPC

Atchem2 can be set up to run on a High Performance Computing (HPC) system. Compilation and configuration are the same as for a normal workstation. Typically, a job scheduler is used to allocate computing resources on an HPC system. A **submission script** is therefore needed to submit the AtChem2 models for execution.

The format and the syntax of the submission script depend on the specific software installed on the HPC system. For instructions on how to prepare a submission script for AtChem2, check the local documentation or ask the HPC system administrator.

An *example* submission script for the Portable Batch System (PBS) is shown below:

```
#PBS -o atchem2.log
#PBS -e atchem2_error.log
#PBS -N base_v1
#PBS -l walltime=15:00:00
#PBS -l vmem=10gb
#PBS -m bea
#PBS -l nodes=1:ppn=1

cd ~/AtChem2/
MODELDIR="base_model_v1"
./atchem2 $MODELDIR/model/output/ $MODELDIR/model/output/reactionRates/ $
```

0.6 Chemical Mechanism

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

0.6.1 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 reaction starts with the `%` character and ends with the `;` character. Comments - enclosed between the characters `*` and `;` - can be added to the mechanism, and will be ignored by the compiler. For example:

```
* conversion of A to C with rate coefficient of 1e-4 *;
% 1E-4 : A = C ;
```

The mechanism file is processed by the script `tools/build.sh`, as explained in the [[model setup page|2. Model Setup and Execution]]. For the build process to work, the `.fac` file must include four sections delimited by a single comment line, which allows the script to recognize the beginning of each section: 1. Generic rate coefficients 1. Complex reactions rate coefficients 1. Sum of peroxy radicals (see below) 1. Chemical Reactions

These comment lines must always be present, even though the respective sections can be empty. A minimal `.fac` file looks like this:

```
* Generic Rate Coefficients ;

* Complex reactions ;

* Peroxy radicals. ;

RO2 =  ;

* Reaction definitions. ;

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

A simple chemical mechanism in FACSIMILE format - with the first step of the atmospheric oxidation of ethanol - is shown below, as an example.

0.6.2 RO2 sum

The sum of organic peroxy radicals (RO2) is a key component of the Master Chemical Mechanism (see the MCM protocol papers: Jenkin et al., Atmos. Environ., 31, 81-104, 1997 and Saunders et al., Atmos. Chem. Phys., 3, 161-180, 2003). Since AtChem2 is designed primarily to run models based upon the MCM, the `.fac` file must contain a section with the RO2 sum. This section must be introduced by the comment line `* Peroxy radicals. ;` (see above) and has the format:

```
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 mechanism (or if the mechanism is not based upon the MCM), the RO2 sum must be left empty, e.g.:

```
RO2 = ;
```

Important: HO2 is a peroxy radical, but it is not an organic molecule. Therefore it should NOT be included in the RO2 sum.

The RO2 sum is automatically generated from the mechanism file during the build process, using the list of RO2 extracted from the MCM database. AtChem2 includes the list of all the organic peroxy radicals in version 3.3.1 of the MCM (`mcm/peroxy-radicals_v3.3.1`), which is used by default. Since v1.1, lists of organic peroxy radicals from other versions of the MCM are also included in the `mcm/` directory: see the file `mcm/INFO.md` for instructions on how to use previous versions of the MCM with AtChem2.

0.6.3 The MCM extraction tool

The MCM website provides a convenient tool which can be used to download the whole MCM, or subsets of it, in FACSIMILE format. After selecting the species of interest in the MCM browser, add them to the *Mark List*, then proceed to the MCM extraction tool and select *FACSIMILE* as format. Make sure to tick the boxes:

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

then press the *Extract* button to download the generated `.fac` file into a directory of choice (e.g., `model/`; see the [[model structure page|1.2 Model Structure]]). The mechanism can be modified with a text editor (if necessary) or directly used in AtChem2. More information about the MCM browser and the extractor tool can be found on the MCM website.

0.6.4 The build process

Atchem2 uses a Python script (`tools/mech_converter.py`, automatically called by `tools/build.sh` during the build process) to convert the chemical mechanism into a format that can be read by the Fortran code.

The script generates one Fortran file, one shared library, and four configuration files from the `*.fac` file:

- **mechanism.f90** contains the equations, in Fortran code, to calculate the rate coefficients of each reaction. By default, it is placed in `model/configuration/`.
- **mechanism.so** is the compiled version of `mechanism.f90`. By default, it is placed in `model/configuration/`.
- **mechanism.species** contains the list of chemical species in the mechanism. By default, it is saved in `model/configuration/`. The file has no header. The first column is the *ID number* of each species, the second column is the name of the species: `1 O    2 O3    3 NO    4 NO2`
- **mechanism.reac** and **mechanism.prod** contain the reactants and the products (respectively) in each reaction of the mechanism. By default, it is saved in `model/configuration/`. The files have a 1 line header with the number of species, the number of reactions and the number of equations in the Generic Rate Coefficients and Complex Reactions sections. The first column is the *ID number* of the reaction, the second column is the *ID number* of the species (from `mechanism.species`) which are reactants/products in that reaction: `29 71 139 numberOfSpecies numberOfReactions numberOfGenericComplex    1 1    2 1    3 1    3 2`
- **mechanism.ro2** contains the organic peroxy radicals (RO2). By default, it is saved in `model/configuration/`. The file has a comment line header (Fortran style). 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 Fortran comment: `! Note that this file is generated by tools/mech_converter.py based upon the file tools/mcm_example.fac. Any manual edits to this file will be overwritten when calling tools/mech_converter.py    23 !CH3O2    26 !C2H5O2    28 !IC3H7O2    29 !NC3H7O2`

The locations of the files generated during the build process can be modified by changing the second and the third argument of the script `tools/build.sh`. For more information and detailed instructions go to: [[2. Model Setup and Execution]].

0.6.5 Example mechanism file

```
* ---------------------------------------------------------------------- *;
* SIMPLE CHEMICAL MECHANISM                                              *;
* Chemical mechanism for ethanol - from MCM v3.3.1                       *;
* ---------------------------------------------------------------------- *;
*;
* Generic Rate Coefficients ;
*;
* Complex reactions ;
*;
* Peroxy radicals. ;
RO2 = HOCH2CH2O2 ;
*;
* Reaction definitions. ;
% 3.0D-12*EXP(20/TEMP)*0.05 : C2H5OH + OH = C2H5O ;
% 3.0D-12*EXP(20/TEMP)*0.9  : C2H5OH + OH = CH3CHO + HO2 ;
% 3.0D-12*EXP(20/TEMP)*0.05 : C2H5OH + OH = HOCH2CH2O2 ;
```

0.7 Model Parameters

The **model parameters** are set in the text file `model/configuration/model.parameters`; they control the general setup of the model.

- **number of steps** and **step size**. The duration of the model run is determined by the number of steps and the step size (in seconds). The step size controls the frequency of the model output for the chemical species listed in `outputSpecies.config` (see [[2.6 Config Files]]), and for the environment variables, the photolysis rates, the diagnostic variables.
  For example, a model runtime of 2 hours, with output every 5 minutes, requires 24 steps with a step size of 300 seconds (24x300 = 7200 sec = 2 hours). Possible values for these parameters are shown below, for reference.

- **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 (see [[2.7 Constraints]]). Two interpolation methods are currently implemented in AtChem2: piecewise constant (`1`) and piecewise linear (`2`). The default option is *piecewise linear interpolation*.

- **rates output step size**. Frequency (in seconds) of the model output for the production and loss rates of selected species. The species for which this parameter is required are listed in `outputRates.config` (see [[2.6 Config Files]]).

- **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 43200 means the model run starts at midday. The **model stop time** is automatically calculated as: `model start time + (number of steps * step size))`.
  Important: if one or more variables are constrained, the interval between the model start time and the model stop time must be equal or shorter than the time interval of the constrained data (see [[2.7 Constraints]]).

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

- **latitude** and **longitude**. Geographical coordinates (in degrees). By convention, latitude North is positive and latitude South is negative, longitude East is negative and longitude West is positive. Latitude and longitude are used only for the calculation of the Earth-Sun angles, which are needed for the MCM photolysis parameterisation (see [[2.5 Photolysis Rates and JFAC]]).

- **day** and **month** and **year**. Start date of the model simulation. The model time is 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. The reaction rates are saved in the directory `model/output/reactionRates/` as one file for each model step, with the name of the file corresponding to the time in seconds. In previous versions of AtChem, this output was called *instantaneous rates*.
  Note that this parameter is different from **rates output step size** (see above), which sets the frequency of a formatted output of reaction rates for selected species of interest. For more information go to: [[2.6 Config Files]].

0.7.1 Runtime reference values

For 1 day at 15 minute intervals:

```
96      number of steps
900     step size
```

For 2 days at 15 minute intervals:

```
192     number of steps
900     step size
```

For 2 days at 1 minute intervals:

```
2880    number of steps
60      step size
```

0.8 Solver Parameters

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

- **atol** (positive real) and **rtol** (positive real): absolute and relative tolerance values for the solver. Standard values for these parameters are listed below, for reference.
- **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 (in seconds) of the timesteps that the solver is allowed to use.
- **maximum number of steps in solver** (positive integer): maximum number of steps used by the solver before reaching **tout**, the next output time.
- **solver type** (integer): selects the linear solver to use: `1` for GMRES, `2` for GMRES preconditioned with a banded preconditioner, `3` for a dense solver. The default option is `2`.
- **banded preconditioner upper bandwidth** (integer): used in the case that `solver type = 2`.
- **banded preconditioner lower bandwidth** (integer): used in the case that `solver type = 2`.

0.8.1 Solver reference values

Standard solver tolerance values:

```
1.0e-04    atol
1.0e-06    rtol
```

0.9 Environment Variables

The **environment variables** define the physical parameters of the box-model, such as temperature, pressure, humidity, latitude, longitude, position of the sun, etc... These variables are set in the text file `model/configuration/environmentVariables.config`.

The environment variables can have a fixed (constant) value or can be constrained to measured values (`CONSTRAINED`), in which case the corresponding data file must be in the `model/constraints/environment/` directory (see [[2.7 Constraints]]). Some environment variables can be calculated by the model (`CALC`) and some can be deactivated if they are not used by the model (`NOTUSED`).

By default, most environment variables are set to a fixed value, corresponding to *standard environmental conditions* (listed below), or to `NOTUSED`.

0.9.1 TEMP

Ambient Temperature (K).

- fixed value
- constrained

Default fixed value = 298.15

0.9.2 PRESS

Ambient Pressure (mbar).

- fixed value
- constrained

Default fixed value = 1013.25

0.9.3 RH

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

- fixed value
- constrained
- not used

Default = NOTUSED (-1)

0.9.4 H2O

Water Concentration (molecules cm-3).

- fixed value
- constrained
- calculated -> requires **RH** set to fixed value or `CONSTRAINED`

Default fixed value = 3.91e+17

0.9.5 DEC

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

- fixed value
- constrained
- calculated -> requires **DAY** and **MONTH**, which are set in `model.parameters` (see [[2.2 Model Parameters]])

Default fixed value = 0.41

0.9.6 BLHEIGHT

Boundary Layer Height. It is required only if the model includes emission or deposition processes (it must be used in the chemical mechanism as a multiplier of the rate coefficient). The unit is typically in cm, but it depends on how the processes are parameterized in the chemical mechanism (see [[2.1 Chemical Mechanism]]).

- fixed value
- constrained
- not used

Default = NOTUSED (-1)

0.9.7 DILUTE

Dilution rate. It is required only if the model includes a dilution process (it must be used in the chemical mechanism as a multiplier of the rate coefficient). The unit is typically in s-1, but it depends on how the process is parameterized in the chemical mechanism (see [[2.1 Chemical Mechanism]]).

- fixed value
- constrained
- not used

Default value = NOTUSED (-1)

0.9.8 JFAC

Correction factor used to correct the photolysis rates (e.g., to account for cloudiness). The calculated photolysis rates are scaled by JFAC, which can have a value between `0` (photolysis rates go to zero) and `1` (photolysis rates are not corrected). JFAC is NOT applied to constant or constrained photolysis rates. For more information go to: [[2.5 Photolysis Rates and JFAC]].

- fixed value
- constrained
- calculated

Default fixed value = 1

0.9.9 ROOF

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

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

Default value = OPEN

0.9.10 Standard environmental conditions

```
Temperature = 25C
Pressure = 1 atm
Relative Humidity = 50%
Day, Month = 21 June
```

0.10 Photolysis rates

The photolysis rates are identified in [[FACSIMILE format|2.1 Chemical Mechanism]] as `J<n>`, where `n` is an integer determined by the MCM naming convention. The photolysis rates are calculated by AtChem2 using the MCM parametrization, as explained in more detail below. Each photolysis rate can also be set to a constant value or to constrained values.

The following rules apply:

1. If a photolysis rate is set as constant, it assumes the given value. Any other photolysis rate, without an explicitly defined constant value, is set to zero.

2. If one or more photolysis rates are set to constrained (and none is set to constant), they assume the values given in the corresponding constraint files. Any other photolysis rate is calculated.

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

The environment variable `ROOF` can also be used to turn the photolysis rates ON/OFF, which is useful for simulations of some environmental chamber experiments (see [[2.4 Environment Variables]]).

0.10.1 Constant photolysis rates

The typical scenario for constant photolysis rates is the use of a lamp in an environmental chamber. All the photolysis rates used in the mechanism need to be given a value (in `model/configuration/photolysisConstant.config`) otherwise they will be set to zero. This approach allows the user to model individual photolysis processes and/or to account for lamps that emit only in certain spectral windows. The format of the `photolysisConstant.config` file is described in the [[configuration files page|2.6 Config Files]].

0.10.2 Constrained photolysis rates

Photolysis rates can be constrained to measured values. In this case, the name of the constrained photolysis rate (e.g., `J2`) must be in `model/configuration/photolysisConstrained` and a file with the constraint data must be present in `model/constraints/photolysis/`. For more information go to: [[2.6 Config Files]] and [[2.7 Constraints]].

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

0.10.3 Calculated photolysis rates

AtChem2 implements the parametrization of photolysis rates used by the Master Chemical Mechanism. It is described in the MCM protocol papers: Jenkin et al., Atmos. Environ., 31, 81, 1997 and Saunders et al., Atmos. Chem. Phys., 3, 161, 2003.

The MCM parametrization calculates the photolysis rate of a reaction (`J`) with the equation:

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

where `l`, `m`, `n` are empirical parameters, `cosX` is the cosine of the solar zenith angle, `secX` is the inverse of `cosX` (i.e., `secX = 1/cosX`) and `tau` is the transmission factor. The empirical parameters are different for each version of the MCM. AtChem2 v1.1 includes the empircal parameters for version 3.3.1 in the file `mcm/photolysis-rates_v3.3.1`. This file also contains the transmission factor `tau`, which can be changed by the user (by default `tau = 1`). It is possible to use previous versions of the MCM parametrization: see the file `mcm/INFO.md` for instructions.

The solar zenith angle is calculated by AtChem2 using latitude, longitude, time of the day and sun declination (see [[2.2 Model Parameters]] and [[2.4 Environment Variables]]). The calculation is detailed in "The Atmosphere and UV-B Radiation at Ground Level" (S. Madronich, Environmental UV Photobiology, 1993).

0.10.4 JFAC

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

The environment variable `JFAC` is a constant or time-dependent parameter that can be used to correct the calculated photolysis rates for external factors not taken into account by the MCM parametrization, such as cloudiness. `JFAC` is defined as the ratio between a measured and the calculated photolysis rate. Typically `J4` (the photolysis rate of NO2) is used for this purpose, as it is one of the most frequently measured photolysis rates.

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

where `j(NO2)` is the measured value and `J4` is calculated with the MCM parametrization (see above). `JFAC` is by default 1, meaning that the calculated photolyis rates are not scaled; it can be set to any value between 0 and 1 (see [[2.4 Environment Variables]]) or it can be constrained (see [[2.7 Constraints]]). Note that only the photolysis rates calculated with the MCM parameterization are scaled by `JFAC`, the constrained and the constant photolysis rates are not.

`JFAC` can also be calculated at runtime. To do so, `JFAC` should be set to the name of the photolysis rate to be used as reference (e.g., `J4`) in `model/configuration/environmentV`
There should be an associated constraint file in `model/constraints/environment/`.
Important: this option is not working very well in the current version of AtChem2, so it is suggested to calculate `JFAC` offline and to constrain it (see issue #16).

0.11 Config Files

The **configuration files** contain the settings for the initial conditions, the constraints and the output of the model. These files complement the configuration settings of the model (in `model.parameters`) and of the solver (in `solver.parameters`), which are in the same directory. For more information go to: [[2.2 Model Parameters]] and [[2.3 Solver Parameters]]).

The configuration files have the extension `.config` and, by default, are in the directory `model/configuration/`. This directory also contains the files generated during the [[build process|2. Model Setup and Execution]] which describe the chemical mechanism (`mechanism.species`, `mechanism.reac`, `mechanism.prod`, `mechanism.ro2`), as explained in the [[chemical mechanism page|2.1 Chemical Mechanism]]. The location of the configuration files can be modified by changing the arguments of the script `tools/build.sh` (see [[2. Model Setup and Execution]]).

The content and the format of the `.config` files are described below. Note that the names of some files have changed with the release of **version 1.1** (November 2018).

0.11.1 environmentVariables.config

This file contains the settings for the environment variables, which are described in detail in the related [[wiki page|2.4 Environment Variables]]. If an environment variable is constrained, there must be a corresponding data file in `model/constraints/environment/` (see [[2.7 Constraints]]).

0.11.2 initialConcentrations.config

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

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

The chemical species not included in this file are automatically initialized to the default value `0`. It is not necessary to initialize the constant and the constrained species (i.e., those listed in `speciesConstant.config` and `speciesConstrained.config`).

The environment variables are set in `environmentVariables.config` (see above) and should not be included in this file.

0.11.3 outputRates.config

This file (called `productionRatesOutput.config` and `lossRatesOutput.config` in v1.0) lists the chemical species for which detailed production rates and loss rates are required. The file has one column, with one species per line.

The frequency of this output is controlled by the **rates output step size** parameter in `model.parameters` (see [[2.2 Model Parameters]]). The format of the corresponding output files - `lossRates.output` and `productionRates.output` - is designed to facilitate the analysis of production and destruction rates for selected species of interests (rather than processing the output files in `model/output/reactionRates/`):

```
     time        speciesNumber    speciesName    reactionNumber        r

3.600000E+003          8              OH               15          0.0000
3.600000E+003          8              OH               20          0.0000
3.600000E+003          9              HO2              16          0.0000
3.600000E+003          9              HO2              17          0.0000

7.200000E+003          8              OH               15          0.0000
7.200000E+003          8              OH               20          0.0000
7.200000E+003          9              HO2              16          0.0000
7.200000E+003          9              HO2              17          0.0000
```

0.11.4 outputSpecies.config

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

The frequency of this output is controlled by the **step size** parameter in `model.parameters` (see [[2.2 Model Parameters]]).

0.11.5 photolysisConstant.config

This file lists the photolysis rates that are constant. The file has three columns: the first column is the number that identifies the photolysis rate (e.g., `1`), the second column is the value of the photolysis rate in **s-1** (e.g., `1e-5`), the third column is the name of the photolysis rate (e.g., `J1`). The photolysis rates are named according to the MCM naming convention. If no photolysis rate is constant, the file should be left empty.

If one or more photolysis rates is set to a constant value, the others (i.e., those not listed in `photolysisConstants.config`) are set to zero. For more information go to: [[2.5 Photolysis Rates and JFAC]].

0.11.6 photolysisConstrained.config

This file (called `constrainedPhotoRates.config` in v1.0) lists the photolysis rates that are constrained. The file has one column, with one photolysis rate per line (e.g., `J1`). The photolysis rates are named according to the MCM naming convention. If no photolysis rate is constrained, the file should be left empty. If a photolysis rate is constrained, there must be a corresponding data file in `model/constraints/photolysis/` (see [[2.7 Constraints]]).

The photolysis rates that are not listed in `photolysisConstrained.config` are calculated by AtChem2 using the MCM parametrization and the parameters in `mcm/photolysis-rates_v3.3.1`. Older versions of the MCM photolysis parametrization can be used, as explained in the file `mcm/INFO.md`. For more information go to: [[2.5 Photolysis Rates and JFAC]].

0.11.7 speciesConstant.config

This file (called `constrainedFixedSpecies.config` in v1.0) lists the chemical species that are constant. The file has two columns: the first column is the list of constant species, the second column is the corresponding concentration (in **molecules cm-3**). If no chemical species is constant, the file should be left empty.

If a chemical species is constant, it does not need to be initialized: the values set in `speciesConstant.config` override those set in `initialConcentrations.config`.

0.11.8 speciesConstrained.config

This file (called `constrainedSpecies.config` in v1.0) lists the chemical species that are constrained. The file has one column, with one species per line. If no chemical species is constrained, the file should be left empty. If a chemical species is constrained, there must be a corresponding data file in `model/constraints/species/` (see [[2.7 Constraints]]).

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

0.12 Constraints

AtChem2 can be run in two modes:

- unconstrained: all variables are calculated by the model from the initial conditions, set in the [[model configuration files|2.6 Config Files]].
- constrained: one or more variables are constrained, i.e. the solver forces their value to a given value. The variables that are not constrained are calculated by the model.

The constrained values must be provided as separate files for each constrained variable. The format of the constraint files is described below. By default, the files with the constraining data are in `model/constraints/species/` for the chemical species, `model/constraints/environment/` for the environment variables, and `model/constraints/photolysis/` for the photolysis rates. The default directories can be modified by changing the arguments of the `atchem2` executable (see [[2. Model Setup and Execution]]).

0.12.1 Constrained variables

Environment variables

All environment variables, except `ROOF`, can be constrained. To do so, set the variable to `CONSTRAINED` in `model/configuration/environmentVariables.config` and create the file with the constraining data. The name of the file must be the same as the name of the variable, e.g. `TEMP` (without extension). See also: [[2.4 Environment Variables]].

Chemical species

Any chemical species in the chemical mechanism can be constrained. To do so, add the name of the species to `model/configuration/speciesConstrained.config` and create the file with the constraining data. The name of the file must be the same as the name of the chemical species, e.g. `CH3OH` (without extension). See also: [[2.6 Config Files]].

Photolysis rates

Any of the photolysis rates in the chemical mechanism can be constrained. The photolysis rates are identified as `J<n>`, where `n` is an integer (see [[2.5 Photolysis Rates and JFAC]]). To constrain a photolysis rate add its name (`Jn`) to `model/configuration/photolysisConstrained.config` and create the file with the constraining data. The name of the file must be the same as the name of the photolysis rate, e.g. `J4` (without extension). See also [[2.6 Config Files]].

0.12.2 Constraint files

The files with the constraining data are text files with two columns separated by spaces. The first column is the time in **seconds** from midnight of day/month/year (see [[2.2 Model Parameters]]), the second column is the value of the variable in the appropriate unit. For the chemical species the unit is **molecules cm-3** and for the photolysis rates the unit is **s-1**; for the environment variables see the related [[wiki page|2.4 Environment Variables]]. For example:

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

The time in the first column of a constraint file can be negative. AtChem2 interprets the negative timestamps as "seconds *before* midnight of day/month/year" (see [[2.2 Model Parameters]]). This can be useful to allow correct interpolation of the variables at the beginning of the model run (see below).

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

0.12.3 Interpolation

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

The user can average the constraints so that they are all at the same timescale or can use the data with the original timestamps. Both approaches have advantages and disadvantages in terms of how much pre-processing work is required, and in terms of model accuracy and integration speed. Whether all the constraints have the same timescale or not, the solver interpolates between data points using the interpolation method selected in the `model/configuration/model.parameters` file (see [[2.2 Model Parameters]]). The default interpolation method is piecewise linear, but piecewise constant interpolation is also available.

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

As mentioned above, the model start and stop time *must be* within the time interval of the constrained data to avoid interpolation errors or model crash. If data is not supplied for the full runtime interval, then the *final* value will be used for all times both *before the first data point* and *after the last data point*. This behaviour is likely to change in future versions, at least to avoid the situation where the last value is used for all times before the first (see issue #294).

A warning is printed for all evaluations outside of the supplied time interval. Users may find it useful to supply data that covers a short time *beyond* the final model time, which may be used by the solver.

0.13 Tools

The `tools/` directory contains a number of auxiliary scripts to install, build and compile AtChem2, and to plot the results of the model:

- shell script to compile the model: `build.sh`.
- Python scripts to process the chemical mechanism: `fix_mechanism_fac.py`, `mech_converter.py`.
- Python scripts to enforce a consistent [[coding style|3.2 Style Guide]]: `fix_indent.py`, `fix_style.py`.
- Ruby script to run the unit tests: `fruit_generator.rb`.
- example chemical mechanism in FACSIMILE format: `mcm_example.fac`.
- `install/` directory containing scripts to install the [[dependencies|1.1 Dependencies]].
- `plot/` directory containing scripts to plot the model results.

In addition, the `tools/` directory contains a copy of the `Makefile`, which has to be copied to the *main directory* and modified as explained in the [[installation page|1. Installation]].

0.13.1 Plot tools

The plotting scripts in `tools/plot/` are only intended to give a quick view of the model results. It is suggested to use a proper data analysis software (e.g., R, Octave/MATLAB, Igor, Origin, etc...) to process and analyze the model results. The scripts are written in various programming languages, but they all produce the same output: a file called `atchem2_output.pdf` in the given directory (e.g., `model/output/`).

From the *main directory*:

```
gnuplot -c tools/plot/plot-atchem2.gp model/output/
octave tools/plot/plot-atchem2.m model/output/
python tools/plot/plot-atchem2.py model/output/
Rscript --vanilla tools/plot/plot-atchem2.r model/output/
```

N.B.: the matlab script (`plot-atchem2.m`) is compatible with both Octave and MATLAB. GNU Octave is an open-source implementation of MATLAB.

0.14 Model Development

Two versions of Atchem2 are available:

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

2) the development version: which is indicated by a version number with the suffix `-dev` (e.g., **v1.1-dev**), and can be downloaded from the `master branch` (https://github.com/AtChem/AtChem2/archive/master.zip) or obtained via **git**.

AtChem2 is under active development, which means that the `master branch` may sometimes be a few steps ahead of the latest stable release. The [[test suite|3.1 Test Suite]] is designed to ensure that changes to the code do not cause unintended behaviour or unexplained differences in the model results, so the development version is usually safe to use, although caution is advised.

The roadmap for the development of Atchem2 can be found here.

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

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

1. Fork the official repository (`AtChem/AtChem2`) to your github account (`username/AtChem2`).

2. Configure git so that `origin` is your fork (`username/AtChem2`) and `upstream` is the official repository (`AtChem/AtChem2`). The output of `git remote -v` should look like this:
```
origin   git@github.com:username/AtChem2.g
(fetch)  origin  git@github.com:username/AtChem2.git
(push)  upstream    git@github.com:AtChem/AtChem2.git
(fetch)  upstream    git@github.com:AtChem/AtChem2.git
(push)
```

3. Create a new branch in your local repository. Make your edits on the branch, commit and push. Before committing, it is advised to run the [[test suite|3.1 Test Suite]] locally to verify whether the changes could cause any problem.

4. Submit a pull request, together with a brief description of the proposed changes. One of the admins will review the edits and approve them or ask for additional changes, as appropriate.

Contributions can also be submitted via email or via the issues page.

A [[Style Guide|3.2 Style Guide]] is available for code contributions. Note that style and indentation of the code are also checked by the [[test suite|3.1 Test Suite]].

0.15 Test Suite

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

To begin using CI on code modifications, create a Pull Request on github from your own fork to `AtChem/AtChem2` (see [[3. Model Development]] for instructions on how to set up **git**). Once the PR is created, Travis CI will automatically run build, unit and behaviour tests on 2 architectures (linux and OSX). Pull requests should only be merged once the Travis CI has completed with passes on both architectures. This is indicated by the meassage: "All checks have passed".

In order to run the Testsuite on your local machine, call `make alltest` from the *main directory*. This will run each of the 3 classes of test in this order: * unit tests: checks that small fragments of code generate the expected outputs; * build test: checks that an example program builds and runs successfully; * behaviour tests: builds each of a number of test setups in turn, and checks that they generate the expected outputs.

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

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

0.15.1 Adding new unit tests

To add new unit tests, do the following: 1. Navigate to `travis/unit_tests`. This contains several files with the ending `*_test.f90`. IF the new test to be added fits into an existing test file, edit that file - otherwise, make a new file, but it must follow that pattern of `*_test.f90`. It is suggested that unit tests covering functions from the source file `xFunctions.f90` should be named `x_test.f90`. 1. The file must contain a module with the same name as the file, i.e. `*_test`. It must `use fruit`, and any other modules as needed. 1. The module should contain subroutines with the naming scheme `test_*~. These subroutines must take no arguments (and, crucially, not have any brackets` for arguments either –subroutine test_calcis correct, butsubroutine test_calc()is wrong). 1. Each subroutine should call one or more assert functions (usuallyassert_equals(), assert_not_equals(), assert_true()orassert_false() These assert functions act as the arbiters of pass or failure - each assert must pass for the subroutine to pass, and each subroutine must pass for the unit tests to pass. 1. The assert functions have the following syntax:

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

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

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

0.15.2 Adding new behaviour tests

To add a new behaviour test called '$TESTNAME' to the Testsuite, you should provide the following:

Each input $TESTNAMEshouldhaveasubdirectory`travis/tests/$TESTNAME/containing the following files in the following structure (*indicates that this file/directory is optional dependent on the configuration used in the test, while+indicates that this directory should be populated with the required files for the constraints declared in file in themodel/configuration` directory):

```
|- mcm
|  |- photolysis-rates_v3.3.1
|  |- peroxy-radicals_v3.3.1
|- model
|  |- configuration
|  |  |- $TESTNAME.fac
|  |  |- environmentVariables.config
|  |  |- mechanism.reac.cmp
|  |  |- mechanism.prod.cmp
|  |  |- mechanism.species.cmp
|  |  |- mechanism.ro2.cmp
|  |  |- model.parameters
|  |  |- outputSpecies.config
|  |  |- outputRates.config
|  |  |- *photolysisConstant.config
|  |  |- *photolysisConstrained.config
|  |  |- solver.parameters
|  |  |- *speciesConstrained.config
|  |  |- *speciesConstant.config
|  |  |- initialConcentrations.config
|  |  `- a .gitignore file containing
|  |
|  |       # Ignore everything in this directory
|  |       *
|  |       # Except the following
|  |       !*.config
|  |       !*.parameters
|  |       !.gitignore
|  `- constraints
|     |- *+environment (1)
|     |  `- a .gitignore file containing
|     |      # Ignore nothing in this directory
|     |
|     |          # Except this file
|     |          !.gitignore
|     |
|     |- *+photolysis (1)
|     |  `- a .gitignore file containing
|     |              # Ignore nothing in this directory
|     |
|     |          # Except this file
|     |          !.gitignore
|     |
|     `- *+species (1)
|        `- a .gitignore file containing
|                # Ignore nothing in this directory
|
|                # Except this file
|                !.gitignore
|- output
|  |- reactionRates/ (3)
|  |- concentration.output.cmp
|  |- environmentVariables.output.cmp
|  |- errors.output.cmp
|  |- finalModelState.output.cmp
|  |- initialConditionsSetting.output.cmp
|  |- jacobian.output.cmp
|   |- lossRates.output.cmp
|   |- mainSolverParameters.output.cmp
|   |- photolysisRates.output.cmp
|   |- photolysisRatesParameters.output.cmp
|   `- productionRates.output.cmp
|- $TESTNAME.out.cmp (2)
```

Notes on this structure: 1. if any environment variables (resp. species, photolysis) are to be constrained by data from a file (as set in `model/configuration/environmentVariabl` `model/configuration/speciesConstrained.config`, `model/configuration/photoly` the subdirectories in `model/constraints/` (`environment/`, `species/`, `photolysis/`) should contain data files with filename equal to the constrained variable name. 1. the file `$TESTNAME.out.cmp`, should contain a copy of the expected screen output; 1. the subdirectory `reactionRates`, should contain a `.gitignore` file and a copy of each of the appropriate files normally outputted to `reactionRates`, with each suffixed by `.cmp`. The `.gitignore` file should contain

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

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

0.16 Style Guide

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

- `tools/fix_style.py` edits files in-place to try to be consistent with the style guide (passing two arguments sends the output to the second argument, leaving the input file untouched, and is thus the safer option). This script is by no means infallible.; therefore, when using the script (by invoking `python tools/fix_style.py filename`), it is strongly recommended to have a backup of the file to revert to, in case this script wrongly edits.
  This script is also used in the [[test suite|3.1 Test Suite]] to check a few aspects of the styling. This works by running the script over the source file and outputting to a `.cmp` file: if the copy matches the original file, then the test passes.
- `tools/fix_indent.py` works similarly, but checks and corrects the indentation level of each line of code. This is also used within the [[test suite|3.1 Test Suite]].

0.16.1 Style recommendations

General principles

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

Specific recommendations

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

0.17 Credits

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

- Chris Martin (PhD thesis)
- Kasia Borońska
- Jenny Young
- Peter Jimack
- Mike Pilling

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

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

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

Additional contributions by:

- Peter Bräuer
- Vasilis Matthaios
- Marios Panagi

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

0.17.1 Funding

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