

Nautilus Documentation

July 10, 2014

Christophe Cossou

Contents

1	TODO	5
2	Starting with Nautilus	5
2.1	Generic informations	5
2.1.1	Python scripts help	5
2.1.2	Comments in input files	5
2.1.3	Main parameter file	5
2.1.4	Browser Vs Simulation	5
2.2	Getting the Git repository	6
2.3	Compilation	6
2.4	Input files	6
2.5	Usefull tools	7
2.5.1	Cleaning a simulation folder	7
2.5.2	Git Prompt	7
3	Parameter file : parameters.in	8
3.1	Automatic test before computation	8
3.2	Simulation parameters	9
3.3	Time evolution of the physical structure	10
3.4	Grain temperature	10
3.5	Switches	10
3.6	Gas parameters	11
3.7	Grain parameters	11
4	Chemical network	13
4.1	Reaction files	13
4.2	Reaction types	13
5	Outputs	14
5.1	Informations : info.out	14
5.2	Abundances	14
5.2.1	Generating ASCII	14
6	Graphic display	14
6.1	Plot abundances	15
6.2	Compare abundances	15
7	For developpers	15
7.1	Unitary tests	16
7.2	Check before commit	16
7.3	How to write documentation with Doxygen	17
7.3.1	General informations	17
7.3.2	For a module	17
7.3.3	For a subroutine	17
7.4	How to generate Doxygen documentation	18
	Bibliography	18
	Index	19

1 TODO

- More details on `grain_tunneling_diffusion` flag need to be added by someone who understand it
- More details on `modify_rate_flag` flag need to be added by someone who understand it
- More details on `conservation_type` flag need to be added by someone who understand it.
- idem for `is_absorption`
- Reference for cosmic ray ionization rate standard value?
- What is the reference UV flux? Paper describing it?
- X ionization rate set to 0, is it normal?

2 Starting with Nautilus

Whether you are a developer or a user, you do not have the same expectations about this manual.

[§ 2] explains the basics, mainly how to get the repository, how to compile the simulation code, how to use the various tools.

[§ 4 on page 13] explains the chemical network, its format, and the various types of reactions available. A chemical network is given with the code, but you can use your own.

Input files are presented in [§ 2.4 on the following page], but the most important one, the only one you will modify daily (*parameters.in*) is dealt with in [§ 3 on page 8].

Output files are presented in [§ 5 on page 14]. Simulations information are displayed in *info.out* (see [§ 5.1 on page 14]).

[§ 6 on page 14] explain what are the Python scripts you can use to plot useful information (mainly abundances) of your simulations (single plot or comparison between several runs).

One last section [§ 7 on page 15] present technical specifications of the code and how to maintain it. For developers who wants more detailed explanations about the code, please refer to the Doxygen documentation, available in the Git repository at <http://nautilus.googlecode.com/git/html/index.html> or in the `html` repository of any local Git clone.

2.1 Generic informations

2.1.1 Python scripts help

From all the python script that come with the code, you can see all parameters and sometimes a few examples by typing:

```
script_name.py help
```

2.1.2 Comments in input files

For all input files, the comment character is `" !"` and can be either at the beginning of a line, or anywhere else. Meaningful character must be comprised inside the 80th first characters of a given line.

2.1.3 Main parameter file

parameters.in is the main parameter file, and is rewritten by the code itself each and every time you run the code.

2.1.4 Browser Vs Simulation

Beware, **nautilus** is generally the default file browser in GNU/Linux Gnome environment. Thus, by typing **nautilus** in a terminal, you will not launch the simulation program, but rather the X server and everything that comes with it (if you're in a remote SSH connection).

2.2 Getting the Git repository

First, you need to get the Git repository from *Googlecode* before actually using the code.

I assume you only want to use it, without ever putting on the distant repository anything. This way, you don't even need to have an account and a password.

In the parent directory where you want to put a **nautilus** folder containing the Git repository, type:

```
git clone https://code.google.com/p/nautilus/
```

2.3 Compilation

The script *Makefile.py* allow you to compile the code (see [§ 2.1.1 on the preceding page] for infos about Python scripts). The default compiler is *gfortran*.

If you want to just compile the code, type:

```
Makefile.py
```

Remark : Errors are in **.log* files associated with the module incriminated. Warnings are stored in a generic file *compilation.log*.



By default, all warnings issued by *ODEPACK* routines are masked (to increase readability, since nobody can modify this thing from scratch). An option **opkd** can force their display.

If you want to compile the binary *nautilus_outputs* that generate ASCII from the binary outputs:

```
Makefile.py output
```

If you want to compile the binary *nautilus_rates* that generate analysis of reaction rates:

```
Makefile.py rates
```

One particular option exist:

```
Makefile.py test
```

This option **test** use compilation options that allow comparison of the code between different versions and is to be used for *compare_simulations.py* (see [§ 7.2 on page 16]).

Remark : Mainly, these option avoid too hard optimization that can result in slightly different results when the code evolve (not because the maths changes but because of optimization only).

2.4 Input files

All input files have the same **.in* extension. An example simulation, containing all necessary input files is provided in the sub-folder **example_simulation**.

The main parameter file is *parameters.in* (see [§ 3 on page 8]).

abundances.in give initial abundances for a set of species (not necessarily all of them). Default minimum values are applied to the species not present in this file (this value is set by the parameter *minimum_initial_abundance* in *parameters.in* (see [§ 3 on page 8])).

element.in gives information about prime elements (base elements used to construct molecules) existing in the simulation (name and mass in Atomic mass unit).

There are 2 parameter files listing all reactions in a given phase (gas or grain):

- *gas_reactions.in*
- *grain_reactions.in*

and 2 parameter files for species present in a given phase (gas or grain) reactions:

- *gas_species.in*
- *grain_species.in*

Remark : Species in *gas_species.in*, *grain_species.in* are not necessarily species from one phase. These are species *involved* in a given phase reaction. But adsorption or desorption transform grain species into gas species, and vice versa.

activation_energies.in provide activation energies for some endothermic reactions.

surface_parameters.in provide information about various energies and parameters for diffusion and movements on the grain surface.

2.5 Usefull tools

2.5.1 Cleaning a simulation folder

The script *nautilus-clean.sh* helps you delete all output files to have a clean simulation folder, with input only

To clean the current working directory, launch:

```
nautilus-clean.sh
```

2.5.2 Git Prompt

You can display in your terminal useful information about a Git repository by modifying your bash profile (assuming you terminal uses Bash, which is almost always the case).

Add in your *.bashrc* or *.bash_profile*:

```
#####
# personnalisation du prompt avec la branche du projet git
#####

git_branch_name_prompt() {
    # Only do this if the current directory is a git repository
    if git status 2>/dev/null 1>/dev/null ; then
        # We keep only the first line, then the last word is our branch
        git_status_output=$(git status 2> /dev/null|head -1) || return

        branch_name() {
            # We get the last word with grep
            echo "$git_status_output"|grep -oE '[^ ]+$'
        }

        echo "($(branch_name))"
    fi
}

git_branch_colour_prompt() {
    # Only do this if the current directory is a git repository
    if git status 2>/dev/null 1>/dev/null ; then
        git_status_output=$(git status 2> /dev/null) || return

        find_pattern_in_status() {
            local pattern="$1"
            [[ "$git_status_output" =~ ${pattern} ]]
        }

        is_clean() {
            local clean_fr='(rien à valider, la copie de travail est propre)'
            local clean_en='(working directory clean)'
            find_pattern_in_status "($clean_fr|$clean_en)"
        }
    fi
}
```

```

is_local_changes() {
    local added_fr='Modifications qui seront validées : '
    local not_added_fr='Modifications qui ne seront pas validées : '
    local added_en='# Changes to be committed'
    local not_added_en='# Changes not staged for commit'
    find_pattern_in_status "($added_fr|$not_added_fr|$added_en|$not_added_en)"
}

is_untracked() {
    local untracked_fr='Fichiers non suivis:'
    local untracked_en='# Untracked files'
    find_pattern_in_status "($untracked_fr|$untracked_en)"
}

# local bold="\033[1m"
local no_colour="\033[0m"
local red="\033[31m"
local green="\033[32m"
local yellow="\033[33m"
local branch_colour=""

if is_untracked
then
    branch_colour=$red
elif is_local_changes
then
    branch_colour=$yellow
elif is_clean
then
    branch_colour=$green
fi

echo -e "$branch_colour"
fi
}

PS1="\h.\u:\[\033[35m\]\W\[\033[0m\]\[\$(git_branch_colour_prompt)\]\$
(git_branch_name_prompt)\[\033[0m\]\$ "

```

3 Parameter file : parameters.in

For generic informations, see [§ 2.1.2 on page 5].

parameters.in has the particularity to be re-written each time you launch a Nautilus simulation. This ensure several things :

- Parameters can be input in random ways, the code will sort them by categories
- New parameters, with default values will be added, to ensure retro-compatibility.

3.1 Automatic test before computation

In *parameters.in*:

```
preliminary_test = 1
```

When set to 1, the parameter *preliminary_test* will allow you to test thoroughly the chemical network and print information in the file *info.out*.

Remark : This parameter should be set to 0 only in the case of intensive campaign of simulations using the very same network, to avoid wasting computation time doing the same tests. But in any other cases, I advice you to leave it activated, because it only take around 1 second at the beginning of the simulation.

The tests currently made are:

- Check that grain species judging from their indexes are indeed grain species
- Check that all species have production AND destruction reactions (error if none, warning if only one)
- Check that each reaction is balanced in prime element and in electric charge
- Display a warning for each reaction having $\alpha=0$ (first parameter for reaction rate formula)
- Check that $T_{\min} < T_{\max}$ for each reaction
- For reaction with the same ID:
 - Check that they have the same reactants and products
 - Check that temperature ranges do not overlap.
- Check that each gas neutral species have a grain equivalent (excluding **GRAIN0** and **XH**).
- Check that each gas neutral species has an adsorption reaction (ITYPE=99) (excluding **GRAIN0** and **XH**).
- Check that each grain species has at least one reaction of each of the following types: 15, 16, 66, 67 (desorption reactions).
- Check that all index ranges associated with a reaction type join themselves to cover all the index range of all reactions.

3.2 Simulation parameters

`start_time = 1.000E+00`

In years, the first output time of the simulation (all simulation starts from $T = 0$).

`stop_time = 1.000E+01`

End of the simulation in years (and also the last output time)

`nb_outputs = 3`

Total number of outputs (including *start_time* and *stop_time*). This number will be used when *output_type* is **log** or **linear**.

`output_type = log`

Define the type of output you want. Possible values are **linear**, **log**, and **table**.

- **linear** : The spacing between the different output times will be linear
- **log** : The different output times will be log-spaced.
- **table** : The different output times are read from the file *structure_evolution.dat*. The parameter *nb_outputs* is then completely ignored.

`relative_tolerance = 1.000E-04`

Relative tolerance of the solver

`minimum_initial_abundance = 1.000E-40`

default minimum initial fraction abundance applied to species whose abundance is not specified in *abundances.in*.

3.3 Time evolution of the physical structure

```
is_structure_evolution = 1
```

If set, the physical structure we define will evolve with time. This evolution will be read from the file *structure_evolution.dat* that must exist in the simulation folder.

This file will have the following format:

```
! time    log(Av)    log(n)    log(T)
! (Myr)   log(mag)   log(cm-3) log(K)
0.000e+00 -1.231e+00 1.813e+00 1.698e+00
2.360e-01 -1.233e+00 1.758e+00 1.712e+00
```

We define respectively time, visual extinction, gas density and gas temperature.

Optionally, one can add a 5-th column to define also grain temperature:

```
! time    log(Av)    log(n)    log(Tg)    log(Td)
! (Myr)   log(mag)   log(cm-3) log(K)    log(K)
0.000e+00 -1.231e+00 1.813e+00 1.698e+00 1.500e+00
2.360e-01 -1.233e+00 1.758e+00 1.712e+00 1.510e+00
```

If so, the parameter *grain_temperature_type* must be set to:

```
grain_temperature_type = table
```

3.4 Grain temperature

You have four ways of defining grain temperature in the code. The parameter *grain_temperature_type* can have the following values:

fixed The grain temperature is fixed throughout the simulation. *initial_dust_temperature* defines this fixed value ;

gas The grain temperature is equal to the gas temperature, no matter what.

computed The grain temperature is calculated following an energy equilibrium with the gas in the structure

table The grain temperature is interpolated from the 5-th column of the *structure_evolution.dat* file. *is_structure_evolution* must be set to 1.

3.5 Switches

To activate (or not) accretion on dust grains and grain surface reactions:

```
is_grain_reactions = 1
```

To activate (or not) the self-shielding of H₂ and CO, related to visual extinction:

```
is_absorption = 1
```

To activate (or not) grain tunneling diffusion (and choose the type of grain tunneling diffusion:

```
grain_tunneling_diffusion = 0
```

Different types are:

0 : Thermal for H, H₂

1 : Quantum tunneling diffusion rate [s⁻¹] [Watson, 1976]

2 : Quantum tunneling diffusion rate [s⁻¹] [Hasegawa et al., 1992]

3 : Choose fastest

To choose whether or not we can modify some rates:

`modify_rate_flag = 1`

Different types are:

-1 : H+H only

1 : modify H

2 : modify H,H₂

3 : modify all

To choose whether or not we can modify some abundances:

`conservation_type = 0`

Different types are:

0 : Only electrons conserved

1 : element #1 conserved

2 : element #1 and #2 conserved

n : element #1...#n conserved

3.6 Gas parameters

`initial_gas_density = 2.000E+04`

initial gas density in particle/cm⁻³

`initial_gas_temperature = 1.000E+01`

initial gas temperature in K

`initial_visual_extinction = 1.500E+01`

initial visual extinction in magnitude

`cr_ionisation_rate = 1.300E-17`

cosmic ray ionization rate in s⁻¹. A standard value is $1.3 \cdot 10^{-17}$.

`x_ionisation_rate = 0.000E+00`

Ionisation rate due to X-rays in s⁻¹.

`uv_flux = 1.000E+00`

Scale factor for the UV flux, in unit of the reference flux. By choosing 1, you will use the nominal value.

3.7 Grain parameters

`initial_dust_temperature = 1.000E+01`

initial dust temperature in K, used when *grain_temperature_type* is **fixed**.

`initial_dtg_mass_ratio = 1.000E-02`

Total mass of dust divided by total mass of gas (dimensionless).

`sticking_coeff_neutral = 1.000E+00`

sticking coefficient for neutral species

`sticking_coeff_positive = 0.000E+00`

sticking coefficient for positive species

`sticking_coeff_negative = 0.000E+00`

sticking coefficient for negative species

`grain_density = 3.000E+00`

mass density of grain material in g/cm^3

`grain_radius = 1.000E-05`

grain radius in cm .

`diffusion_barrier_thickness = 1.000E-08`

Thickness of the barrier in cm that a surface species need to cross while undergoing quantum tunneling to diffuse from one surface site to another. This is used in the formalism [Hasegawa et al., 1992, see equation 10 (parameter a)].

`surface_site_density = 1.500E+15`

density of sites at the surface of the grains in $\text{number}/\text{cm}^2$.

`diff_binding_ratio = 5.000E-01`

Ratio (adimensioned) used to compute the DIFFUSION_BARRIER from the BINDING_ENERGY if not known

`chemical_barrier_thickness = 1.000E-08`

Parameter (in cm) used to compute the probability for a surface reaction with activation energy to occur through quantum tunneling. This is the thickness of the energy barrier [Hasegawa et al., 1992, See equation 6].

`cr_peak_grain_temp = 7.000E+01`

Peak grain temperature in K when struck by a cosmic ray.

`cr_peak_duration = 1.000E-05`

duration [s] of peak grain temperature

`Fe_ionisation_rate = 3.000E-14`

(cosmic) Fe-ion-grain encounter [$\text{s}^{-1}\text{grain}^{-1}$] for 0.1 micron grain. For cosmic photo desorptions, only Fe-ions are efficient to heat grains.

`vib_to_dissip_freq_ratio = 1.000E-02`

(dimensionless) For the RRK (Rice Ramsperger-Kessel) desorption mechanism. Ratio of the vibration frequency (proper energy of a species when it is created on a grain) to the dissipation frequency (energy needed by the molecule to be evaporated from the grain surface). This ratio help to determine if a species evaporate after its formation on the grain surface. Since the dissipation frequency is usually unknown, this ratio is a free parameter. A common value is 1%.

4 Chemical network

4.1 Reaction files

The files concerned are : *gas_reactions.in*, *grain_reactions.in* and *activation_energies.in*

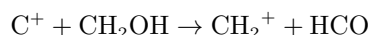
A typical reaction line is:

```

1 !      Reactants                      ->      Products
   |
   |      xxxxxxxxxxxxxxxxxxxxxxxx ITYPE Tmin   Tmax formula ID xxxxx
2 C+      CH2OH                      -> CH2+      HCO
   |                                     7.500E-10 -5.000E-01 0.000E+00 0.00e
   |      +00 0.00e+00   NA  4      10      280  3  6098 1  1

```

The reaction displayed here is:



The temperature range is $T \in [10; 280]$ K. The type of reaction is 4. The ID of the reaction is 6098. The formula used to compute reaction rate is 3, with the 3 parameters $A = 7.5 \cdot 10^{-10}$, $B = -0.5$ and $C = 0$. Other columns are ignored, as the "xxx" emphasize in the legend line associated.

Each species name is encoded with 11 characters.

All reaction files have the same format. Depending on the evolution of the code, the number of reactants (MAX_REACTANTS) or products (MAX_PRODUCTS) may vary (increase), so theses files must be modified to take that into account.

The following global variables are here to tell to the code that theses number have changed.

```

1 MAX_REACTANTS = 3 !< The maximum number of reactants for one reaction.
2 MAX_PRODUCTS = 5 !< The maximum number of products for one reaction.
3 MAX_COMPOUNDS = MAX_REACTANTS + MAX_PRODUCTS !< Total maximum number of compounds for
   one reaction (reactants + products)

```



Pay attention to the fact that some things might need manual modifications in the code. If this number change, `get_jacobian(N, T, Y, J, IAN, JAN, PDJ)` must be actualized, since each reactant and product has its own variable, a new one must be created for the new column possible.

4.2 Reaction types

Chemical reactions can be of several types.

Here is the list:


- 0 Gas phase reactions with GRAINS
- 1 Photodissociation/ionisation with cosmic rays (CR)
- 2 Gas phase photodissociations/ionisations by secondary UV photons generated by CR
- 3 Gas phase photodissociations/ionisations by UV
- 4-8 Bimolecular gas phase reactions - several possible formula
- 10-11 H₂ formation on the grains when IS_GRAIN_REACTIONS=0

Remark : Only one reaction for each of the two types (10 and 11).

- 14 Grain surface reactions
- 15 Thermal evaporation
- 16 Cosmic-ray evaporation
- 17-18 Photodissociations by Cosmic rays on grain surfaces
- 19-20 Photodissociations by UV photons on grain surfaces

- 21 Grain surface reactions
- 66 Photodesorption by external UV
- 67 Photodesorption by UV induced by cosmic rays
- 98 storage of H₂S under a refractory form
- 99 Adsorption on grains

5 Outputs

 All output files have the same ***.out** extension.

Output files are:

- *info.out*: Various information about the simulation (see [§ 5.1] for more details)
- *species.out*: The list of species and their corresponding index
- *elemental_abundances.out*: The prime elements abundances and mass at the beginning of the simulation. A **.tmp* version display the same infos, but at the last output.
- *abundances*.out*: In binary format (unformatted), abundances of all species, each file for a different output time. *nautilus_outputs* read theses files to generate ASCII files (see [§ 5.2] for more details).
- *rates*.out*: In binary format (unformatted), rates of all reactions, each file for a different output time. *nautilus_rates* read theses files to generate ASCII files.
- *abundances.tmp*: The abundances of all species at the last output in ASCII.

5.1 Informations : info.out

In the file *info.out*, the ID reference of the current version of nautilus, used to run the simulation is printed, as well as other useful information about the state of Nautilus and how the simulation was executed.

5.2 Abundances

Files are named *abundances.000001.out* and so on, for each output time. Outputs are stored in binary format. Format is as follow : On a first line: time in seconds

On a second line, information about the physical structure properties:
gas temperature [K], dust temperature [K], H₂ gas density [part/cm³] (assuming H₂ is the main reservoir for H), visual extinction [mag], x ionization rate [s⁻¹]

And finally a line containing the abundances relative to H for all species [number ratio]

5.2.1 Generating ASCII

First you need to compile the output binary. In the source directory, type:

```
Makefile.py output
```

Then, in your simulation folder, type:

```
nautilus_outputs
```

(this assume you have an alias, but absolute path also work)

6 Graphic display

Python script were created to help the user display useful information about their simulations.

6.1 Plot abundances

The script *nautilus-plot-abundances.py* allow you to display the time evolution of the abundances of one or more species:

```
nautilus-plot-abundances.py species=C0,H20
```

You can zoom in a given period of time (**tmin**, **tmax** or both):

```
nautilus-plot-abundances.py species=C0,H20 tmin=1e4 tmax=1e6
```

Even if you can modify and store the result in the graphic windows displayed, a default version is automatically written in **abundances.pdf**

Check the other options and detailed examples via:

```
nautilus-plot-abundances.py help
```

6.2 Compare abundances

The script *nautilus-compare-abundances.py* allow you to display the time evolution of the abundances of one or more species for all sub-folders of the current working directory, assuming each one is a simulation:

```
nautilus-compare-abundances.py species=C0,H20
```

If there is a lot of sub-folder, you can select those you want by:

```
nautilus-compare-abundances.py species=C0,H20 dir=simu1,simu2
```

All option existing for *nautilus-plot-abundances.py* applies here.

Even if you can modify and store the result in the graphic windows displayed, a default version is automatically written in **compare_abundances.pdf**

Check the other options and detailed examples via:

```
nautilus-compare-abundances.py help
```

7 For developpers

Generic information to start with:

- indentation is made by 2 spaces (no tabulation)
- Constants are in capital letters (even if **fortran** is not case sensitive)
- Variable need to be explicit about what they refer to. Use underscore "_" to separate words if need. Avoid word shortcuts as much as possible since you'd be the only one to understand your rules for the shortcuts definitions.
- Add comments for aim of each routine, description of each variable (do not forget input and output flags).
- Add comments about specific portion of the code that are less obvious
- use space before and after mathematical operator (=, +, -, *, /). For instance :

```
a=(b+c)/(d+2) ! WRONG
a = (b + c) / (d + 2) ! GOOD
```

- use space after (but not before) parenthesis and commas For instance:

```
call r(a,e,c,d) ! WRONG
call get_radius(a, e, c, d) ! GOOD
call get_radius(arg1=a, radius=e, optional=c, distribution=d) ! BETTER
```

Constants and global variables are defined in the module `global_variable.f90`. `nautilus_main.f90` contains all the main routines.

The three main programs are:

- `nautilus` in the source file `nautilus.f90` (compilation: `Makefile.py`)
- `nautilus_outputs` in the source file `nautilus_outputs.f90` (compilation: `Makefile.py output`)
- `nautilus_rates` in the source file `nautilus_rates.f90` (compilation: `Makefile.py rates`).

One last program exist to do some unitary tests on **Nautilus** (see [§ 7.1] for more details).

7.1 Unitary tests

A **fortran** program `unitary_tests` (`unitary_tests.f90`) was specifically designed to test some routines of the code separately. This is the main reason why a `nautilus_main.f90` file was created, because we need to access these routines separately from the `nautilus` program.

To run the unitary tests, use the Python script designed for that (he will compile the source code too):

```
unitary_tests.py
```

The code will display some information and ask you what test you want to display graphically (and generate the corresponding **.pdf**):

```
0 : av_interpolation
1 : density_interpolation
2 : gas_temperature_interpolation
3 : grain_temperature_interpolation
4 : test_av_read
5 : test_density_read
6 : test_gas_temperature_read
What test do you want to display? (0-6 ;
'all' treat them all ; 'l' display list again)
```

The principle of this code is to do some tests, store the results in data files, generate Gnuplot script files associated. To get the plots, you only have to generate it by:

```
gnuplot script_name.gnuplot
```

All files generated by the program are stored in the "tests" sub-folder. You can generate the **.pdf** manually if you are familiar with Gnuplot, and of course, view them separately.

Remark : It's up to you to write new routines in `unitary_tests.f90` and mimic the way I wrote the previous one to tests new functionality of the code.

Please keep in mind that you need to add a call `new_routine()` in the main program, just before `contains` to ensure your routine is executed.

7.2 Check before commit

You can have two types of modifications in the code, those that modify the outputs, and those that do not.

I created a Python script `compare_simulations.py` that can help you ensure the code do not change the results between two versions. It's up to you to check your modifications when the outputs are different though.

The basic use of this script is as follow (in the code main directory):

```
Makefile.py test && compare_simulation.py
```

If you want to update the "reference" version of the code, because outputs changed and you want a new reference, type:

```
compare_simulation.py rev=HEAD
```

rev can accept any commit reference, **HEAD**, **HEAD^** and a hashtag will work.

7.3 How to write documentation with Doxygen

7.3.1 General informations

Doxygen comments generally have a marker and the description. The character to declare the marker is @, but one need to start the commented line by > to declare there is something here.

If this is an inline comment, with the code on the left, comments are like this:

the code !< the doxygen description

just to say that the description refers to the code on the left.

To continue a description on another comment line, just double the comment character:

```
!> @brief I describe something
!! on several lines.
```

Remark : This will not make two lines in the generated documentation though. This is only a way to avoid never ending lines with thousands of characters.

7.3.2 For a module

Start the module file with:

```
1  !*****
2  ! MODULE: Module Name
3  !*****
4  !
5  !> @author
6  !> Module Author Name and Affiliation
7  !
8  ! DESCRIPTION:
9  !> @brief Brief description of what can be done in this module.
10 !! This description can be on several lines.
11 !! \n\n Do not forget the symbol "\n" to create a new line.
12 !
13 !*****
```

7.3.3 For a subroutine

Before the definition of the routine, add the following text:

```
1  !!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
2  !> @author
3  !> Routine Author Name and Affiliation.
4  !
5  ! DESCRIPTION:
6  !> @brief Brief description of routine.
7  !! Flow method (rate of change of position) used by integrator.
8  !! Compute |f$| \frac{d\lambda}{dt}, \frac{d\phi}{dt}, \frac{dz}{dt} |f$|
9  !
10 !!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
```

In the rest of the routine, do not forget to add comments for input and outputs of the routine:

```
1  implicit none
2
3  ! Inputs
4  real(double_precision), intent(in) :: delta_t !<[in] description
5
6  ! Outputs
7  integer, intent(out) :: istate !<[out] Description of the variable
8  !! that can be continued on another line.
9
10 ! Inputs/Outputs
11 real(double_precision), intent(inout) :: time !< [in,out] description
12 !! \n Continuation line.
13
14 ! Locals
15 real(double_precision) :: t !< The local time, starting from 0 to delta_t
```

7.4 How to generate Doxygen documentation

To generate **Doxygen** documentation, type the following command in the root Git repository:

```
doxygen doxygen.conf 2>doxygen.log
```

doxygen.conf is the parameter file of *Doxygen*. Errors will be redirected in the file **doxygen.log**.



The documentation (in the folder **html**/) is synchronized with the server repository. Thus, you may generate documentation only after major changes in the *Doxygen* documentation, to avoid too many changes for too few improvements.

Bibliography

- T. I. Hasegawa, E. Herbst, and C. M. Leung. Models of gas-grain chemistry in dense interstellar clouds with complex organic molecules. *ApJS*, 82:167–195, September 1992. doi: 10.1086/191713.
- W. D. Watson. Interstellar molecule reactions. *Reviews of Modern Physics*, 48:513–552, October 1976. doi: 10.1103/RevModPhys.48.513.

Index

.bash_profile, 7

.bashrc, 7

binary

nautilus, 16

nautilus_outputs, 6, 14, 16

nautilus_rates, 6, 14, 16

unitary_tests, 16

chemical network, 13

compare_simulations.py, 6, 16

compilation.log, 6

Doxygen, 17, 18

gfortran, 6

Googlecode, 6

input file

abundances.in, 6, 9

activation_energies.in, 7, 13

element.in, 6

gas_reactions.in, 6, 13

gas_species.in, 6, 7

grain_reactions.in, 6, 13

grain_species.in, 6, 7

parameters.in, 5, 6, 8

structure_evolution.dat, 9, 10

surface_parameters.in, 7

Makefile.py, 6

nautilus-clean.sh, 7

nautilus-compare-abundances.py, 15

nautilus-plot-abundances.py, 15

ODEPACK, 6

output file

abundances*.out, 14

abundances.tmp, 14

elemental_abundances.out, 14

info.out, 5, 8, 14

rates*.out, 14

species.out, 14

parameter

chemical_barrier_thickness, 12

conservation_type, 11

cr_ionisation_rate, 11

cr_peak_duration, 12

cr_peak_grain_temp, 12

diff_binding_ratio, 12

diffusion_barrier_thickness, 12

Fe_ionisation_rate, 12

grain_density, 12

grain_radius, 12

grain_temperature_table, 10

grain_temperature_type, 10, 11

grain_tunneling_diffusion, 10

initial_dtg_mass_ratio, 11

initial_dust_temperature, 10, 11

initial_gas_density, 11

initial_gas_temperature, 11

initial_visual_extinction, 11

is_absorption, 10

is_grain_reaction, 10

is_structure_evolution, 10

minimum_initial_abundance, 6, 9

modify_rate_flag, 11

nb_outputs, 9

output_type, 9

preliminary_test, 8

relative_tolerance, 9

start_time, 9

sticking_coeff_negative, 12

sticking_coeff_neutral, 11

sticking_coeff_positive, 12

stop_time, 9

surface_site_density, 12

uv_flux, 11

vib_to_dissip_freq_ratio, 12

x_ionisation_rate, 11

reaction types, 13

unitary tests, 16