scaks Documentation

Release scaks

Zhengjiang Shao

Dec 02, 2018

Contents:

1 Introduction **1**
- 1.1 Why Micro-kinetic Model for Catalyst? . . . 1
- 1.2 Why *scaks*? . . . 1

2 Installation **3**
- 2.1 *scaks* Installation . . . 3
- 2.2 scaks-Hub Installation . . . 4

3 Tutorial **5**
- 3.1 Solve a micro-kinetic model . . . 5
- 3.2 Solve model in scaks-Hub . . . 10
- 3.3 Parse reaction expressions . . . 15
- 3.4 Build a model in scaks-Hub . . . 17
- 3.5 [Animation] Run a job in scaks-Hub . . . 21
- 3.6 Register hybrid method for Hybrid Newton iteration . . . 21
- 3.7 Define your own on-the-fly analysis plugin . . . 22

4 API **23**
- 4.1 models.KineticModel . . . 24
- 4.2 models.MicroKineticModel . . . 24
- 4.3 parsers.rxn_parser . . . 24
- 4.4 parsers.ParserBase . . . 24
- 4.5 parsers.AbsoluteEnergyParser . . . 24
- 4.6 parsers.RelativeEnergyParser . . . 24
- 4.7 parsers.KMCParser . . . 24
- 4.8 solvers.SolverBase . . . 24
- 4.9 solvers.MeanFieldSolver . . . 24
- 4.10 solvers.rootfinding_iterators . . . 24
- 4.11 solvers.MeanFieldSolver . . . 24
- 4.12 correctors.ThermodynamicCorrector . . . 24
- 4.13 plotters.EnergyProfilePlotter . . . 24
- 4.14 utilities.check_utilities . . . 24
- 4.15 utilities.coordinate_utilities . . . 24
- 4.16 utilities.format_utilities . . . 24
- 4.17 datatabase.elements_data . . . 24
- 4.18 datatabase.lattice_data . . . 24
- 4.19 datatabase.thermo_data . . . 24

4.20 descriptors.descriptors 24
4.21 descriptors.component_descriptors 24
4.22 mpicommons 24
4.23 plugins.hybrid_methods 24
4.24 plugins.analysis 24

CHAPTER 1

Introduction

Welcome to the *scaks* documentation!

scaks is a Python implementation of the Micro-kinetic model solving model with a user-friendly web GUI distributed under the GPLv3 license. It's an acronym for Micro Kinetics Analysis for Catalyst.

You can always find the latest stable version of the program here: https://github.com/pytlab/scaks

This documentation describes version 1.0.0

1.1 Why Micro-kinetic Model for Catalyst?

Rational design of catalysis assisted by means of first-principle calculation is current one of the most important topic in field of heterogeneous catalysis because the traditional try-and-error method can't meet the rapidly increasing demands of catalyst industry development. In the process of rational design and catalysis screening implementation, solving the micro-kinetic model provides the theoretical basis for describing the turnover frequency and selectivity.

1.2 Why *scaks*?

Despite the great development of microkinetics simulation methods and software, the implementations of kinetics analysis with relatively fixed code architect are not flexible and scalable enough for more and more complex catalytic system simulation. Besides, the absence of user-friendly interfaces is also the obstacle on the road for chemical researchers without programming knowledges to use those programs.

To this end, we present a Python module called "scaks" with web GUI to help researchers to solve microkinetics more easily by lowering the learning and using barriers. As a complete Python module which can be imported in other user customized programs, scaks provide robust and flexible interfaces to help expert users create one or more models and solve them at a time with many components built in such as powerful reaction expression parser, object-oriented based energy profile plotter and so on. In order to provide a more friendly and interactive user interfaces for other chemical researchers, we also use the famous Python micro web framework Flask and the web front-end framework Bootstrap to build a web application using scaks as the back-end calculation core. Then we can run the microkinetic

model application on both local and remote server and make it possible for a scalable and large scaled cloud computing service.

CHAPTER 2

Installation

2.1 *scaks* Installation

scaks can be installed in different ways. In current version, scaks has no C or C++ backend code, therefore it can be easily installed even from source code.

2.1.1 Prerequisite

As scaks provides a higher level Message Passing Interfaces, you need to install an implementation of MPI on your machine before installing mpi4py.

MPI implementations: 1. MPICH 2. OpenMPI 3. Microsoft MPI

2.1.2 Via Pip (Recommended)

```
pip install scaks
```

2.1.3 From source

```
git clone --recursive https://github.com/PytLab/scaks.git
cd scaks
python setup.py install
```

2.1.4 Installation check

To ensure you have installed correctly on your machine, try to import `scaks` in Python shell:

```
>>> import scaks
```

For developers

We also provide unit tests for developers, if you have clone the repository, just

```
cd scaks/tests
python scaks_test.py
```

2.2 scaks-Hub Installation

2.2.1 Install and run scaks-Hub example locally

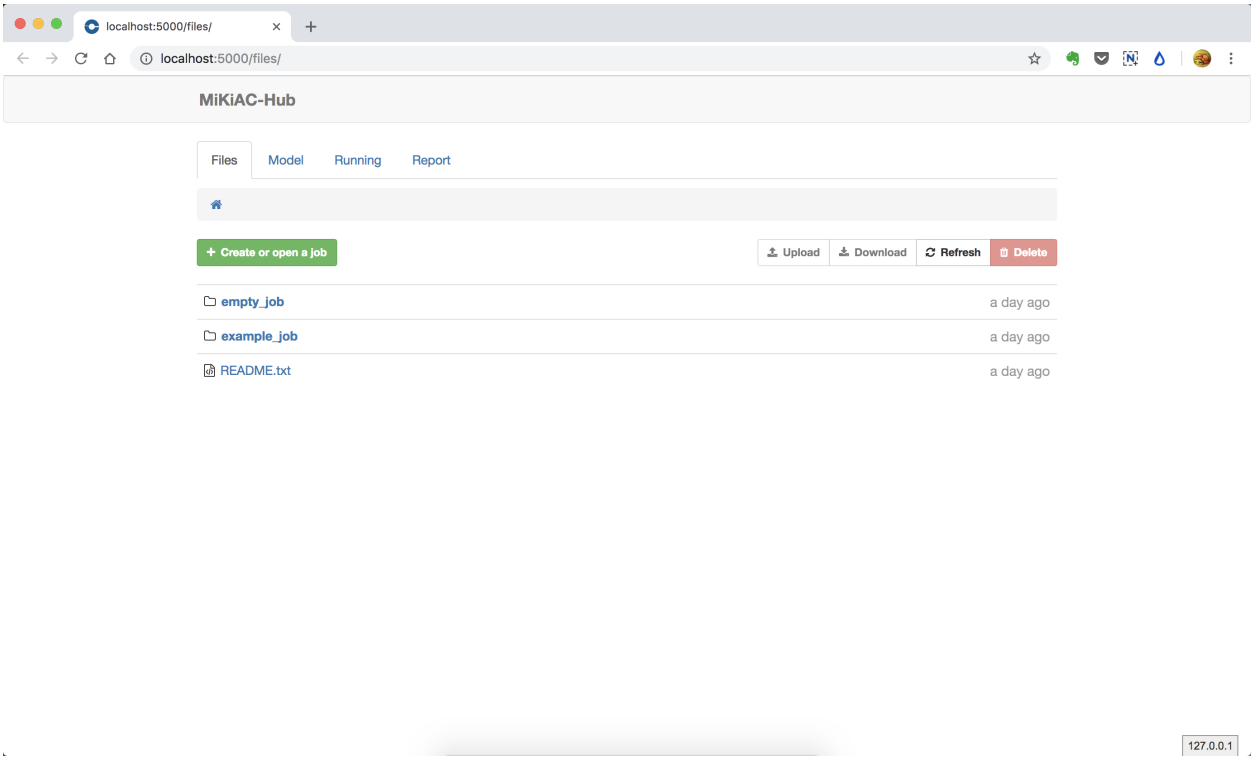
Clone the repository:

```
git clone --recursive git@github.com:PytLab/scaks-hub.git
```

Run the scaks-hub:

```
cd scaks-hub/example
../scaks-hub runserver
```

Your browser will open automatically, if not, open your browser and visit http://localhost:5000



CHAPTER 3

Tutorial

Perhaps the easiest way to get started with your own Micro-kinetic modeling is to have a quick look at a few usage examples. We will take a look at few examples from the simplest CO Oxidation reaction on Pt100 surface using both scaks API and web GUI.

3.1 Solve a micro-kinetic model

scaks.models module provide different model classes for different kinetic model construction(only micro-kinetic model in current version). Model can constructed from either a setup dict data structure for running in interactive shell or a setup input file for submitting a computing job.

3.1.1 Using setup file

By providing a set of input files including model definition, energy information, script using scaks API can parse all those files and run the model automatically.

- `pt-100.mkm`: Input file for micro-kinetic model information:

```
rxn_expressions = [
    'CO_g + *_s -> CO_s',
    'O2_g + 2*_s <-> O-O_2s -> 2O_s',
    'CO_s + O_s <-> CO-O_2s -> CO2_g + 2*_s',
]

# Gas pressure.
species_definitions = {}
species_definitions['CO_g'] = {'pressure': 1.32*10**-7}
species_definitions['O2_g'] = {'pressure': 5.26*10**-7}
species_definitions['CO2_g'] = {'pressure': 1.32*10**-7}

# Site info.
species_definitions['*_s'] = {'site_name': 'top', 'type': 'site', 'total': 1.0}
```

(continues on next page)

(continued from previous page)

```
# Temperature.
temperature = 500  # K

unitcell_area = 9.0e-20
active_ratio = 4./9.

parser = "RelativeEnergyParser"
solver = "SteadyStateSolver"
corrector = "ThermodynamicCorrector"
plotter = "EnergyProfilePlotter"

rate_algo = "CT"
rootfinding = "MDNewton"
tolerance = 1e-50
max_rootfinding_iterations = 100
```

- `rel_energy.py`: Energy information for each elementary reaction:

```
Ga, dG = [], []

# CO_g + *_s -> CO_s
Ga.append(0.0)
dG.append(-2.09)

# O2_g + 2*_s -> 2O_s
Ga.append(0.07)
dG.append(-2.39)

# CO_s + O_s <-> CO-O_2s -> CO2_g + 2*_s
Ga.append(0.39)
dG.append(-0.46)
```

With these input files prepared, we can use scaks's API to construct a micro-kinetic model:

```
from scaks.models.micro_kinetic_model import MicroKineticModel

model = MicroKineticModel(setup_file='pt-100.mkm')
```

While constructing the model, all model components like parser, solver and plotter are instantialized automatically. Thus, we can use components to parse data and solve model:

```
model.parser.parse_data()   # Parse input data
model.solver.get_data()     # Pass data to solver
model.run()                 # Solve micro-kinetic model
```

3.1.2 Using setup dictionary

Setup dictionary is a Python data structure containing essential information for constructing a preliminary Micro-Kinetic Model. It must contain the following informations (keys):

- `rxn_expressions`: All elementary reaction expressions
- `species_definitions`: Species information in reaction path such as gas name and partial pressure, adsorption site types and total coverages.

- `temperature`: Reaction temperation in K
- `parser`: The parser name for data and input file parsing

The model definition dict for CO oxidation on Pt(100) surface could be written as:

```
model_dict = dict(
    rxn_expressions = [
        'CO_g + *_s -> CO_s',
        'O2_g + 2*_s <-> O-O_2s -> 2O_s',
        'CO_s + O_s <-> CO-O_2s -> CO2_g + 2*_s',
    ],

    species_definitions = {
        'CO_g': {'pressure': 1.32e-7},
        'O2_g': {'pressure': 5.26e-7},
        'CO2_g': {'pressure': 1.32e-7},
        '*_s': {'site_name': '111', 'type': 'site', 'total': 1.0},
    },

    temperature = 500,
    parser = "RelativeEnergyParser",
)
```

Construct corresponding micro-kinetic model:

```
from scaks.models.micro_kinetic_model import MicroKineticModel
model = MicroKineticModel(setup_dict=model_dict)
```

Use constructed model to generate file template for energy data input:

1.Absolute energy input file template:

```
model.generate_absolute_energies_file('./abs_energy.py')
```

Then scaks will parse all model information in reaction expressions to create a `abs_energy.py` with below content in current directory:

```
# Absolute energies for all species.
absolute_energies = {

    'CO2_g': 0.0, # eV

    'CO_g': 0.0, # eV

    'O2_g': 0.0, # eV

    'CO_s': 0.0, # eV

    'O_s': 0.0, # eV

    'CO-O_2s': 0.0, # eV

    'O-O_2s': 0.0, # eV

    '*_s': 0.0, # eV

}
```

2.Relative energy input file template:

```
model.generate_relative_energies_file('./rel_energy.py')
```

Then scaks will parse all model information in reaction expressions to create a `rel_energy.py` with below content in current directory:

```
# Relative Energies for all elementary reactions.
Ga, dG = [], []

# CO_g + *_s -> CO_s
Ga.append()
dG.append()

# O2_g + 2*_s <-> O-O_2s -> 2O_s
Ga.append()
dG.append()

# CO_s + O_s <-> CO-O_2s -> CO2_g + 2*_s
Ga.append()
dG.append()
```

After inputting energy data in the template file, you can instantialize solver explicitly and put it in micro-kinetic model:

```
from scaks.solvers.steady_state_solver import SteadyStateSolver

solver = SteadyStateSolver(model)
model.set_solver(solver)
```

With solver integrated, model can parse data and solve the model:

```
model.parser.parse_data('./rel_energy.py')
model.solver.get_data()
model.run()
```

3.1.3 Use script to run a job

Besides instantiate model using scaks API, we provide a simple `run.py` script to parse those input files and solve the micro-kinetic model automatically, the details of the script:

```
import logging
import sys
import time

from scaks.compatutil import subprocess
from scaks.mpicommons import mpi
from scaks.models.micro_kinetic_model import MicroKineticModel
from scaks.utilities.format_utilities import convert_time

# Custom parameters.
OdeInterval = 0.001          # ODE integration time interval.
OdeEnd = 1          # ODE integration time limit.
OdeOutput = True            # Output ODE integration data or not.
CalcXRC = True              # Calculate Degree of Rate Control(XRC) or not.
ProductionName = "CO2_g"  # Production name of your model.
OdeOnly = False             # Do ODE integration only.
```

(continues on next page)

(continued from previous page)

```
if "__main__" == __name__:
    # Clean up current dir.
    subprocess.getstatusoutput("rm -rf out.log auto_*")

    # Set script logger.
    logger = logging.getLogger("model.MkmRunScript")

    # Get setup file.
    status, output= subprocess.getstatusoutput("ls *.mkm | tail -1")
    if status:
        if mpi.is_master:
            logger.error(output)
            logger.info("Exiting...")
        sys.exit(1)

    start = time.time()
    try:
        # Build micor-kinetic model.
        model = MicroKineticModel(setup_file=output)

        # Read data.
        parser = model.parser
        solver = model.solver
        parser.parse_data()
        solver.get_data()

        # Initial coverages guess.
        trajectory = solver.solve_ode(time_span=OdeInterval,
                                      time_end=OdeEnd,
                                      traj_output=OdeOutput)
        init_guess = trajectory[-1]

        # Run.
        model.run(init_cvgs=init_guess,
                  solve_ode=OdeOnly,
                  coarse_guess=False,
                  XRC=CalcXRC,
                  product_name=ProductionName)
    except Exception as e:
        if mpi.is_master:
            msg = "{} exception is catched.".format(type(e).__name__)
            logger.exception(msg)
        raise e

    # Time used.
    end = time.time()
    t = end - start
    h, m, s = convert_time(t)

    if mpi.is_master:
        logger.info("Time used: {:d} h {:d} min {:f} sec".format(h, m, s))
```

Just use Python to execute the script to run the job:

```
python run.py
```

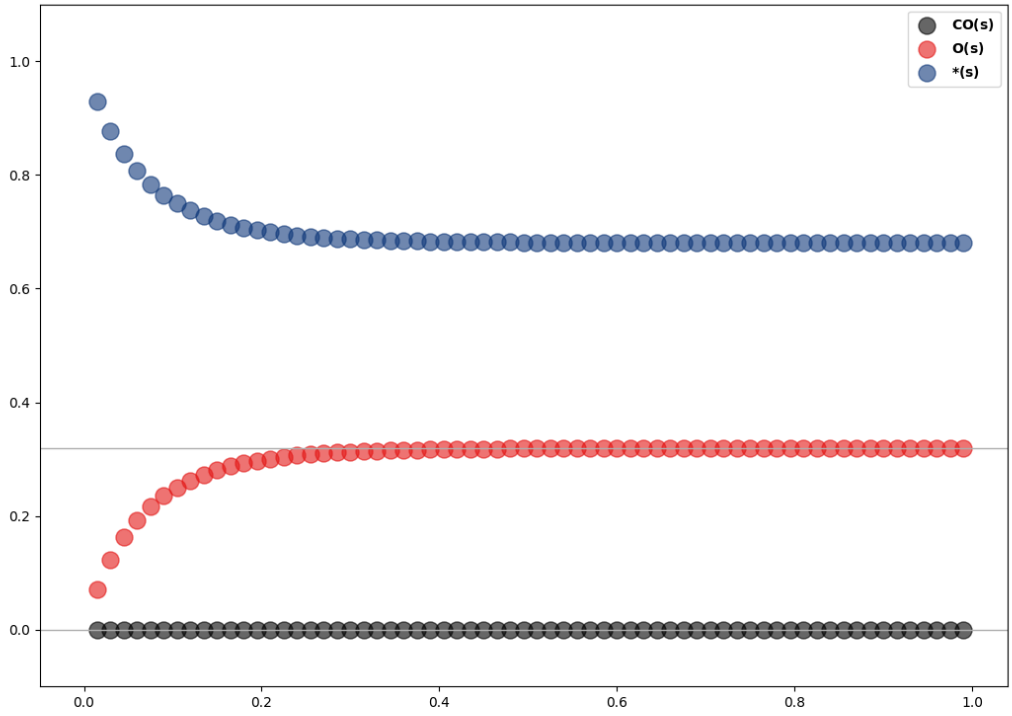
Of course, instead of using built-in script, users can write their own script with mikac API to customize the function-

ality.

After the solving is finished, new output files are generated in current directory:

- `out.log`: All output log information such as TOFs, reversibilities, steady state coverages and so on.
- `data.pkl`: Serialized result data such as turnover frequencies (TOF), steady state coverages and so on. Variable to be dumped is controlled in setup file.
- `auto_ode_coverages.py`: A python module file containing ODE integration data

With ODE plotting script in scaks package, the ODE integration trajectory can be visualized:

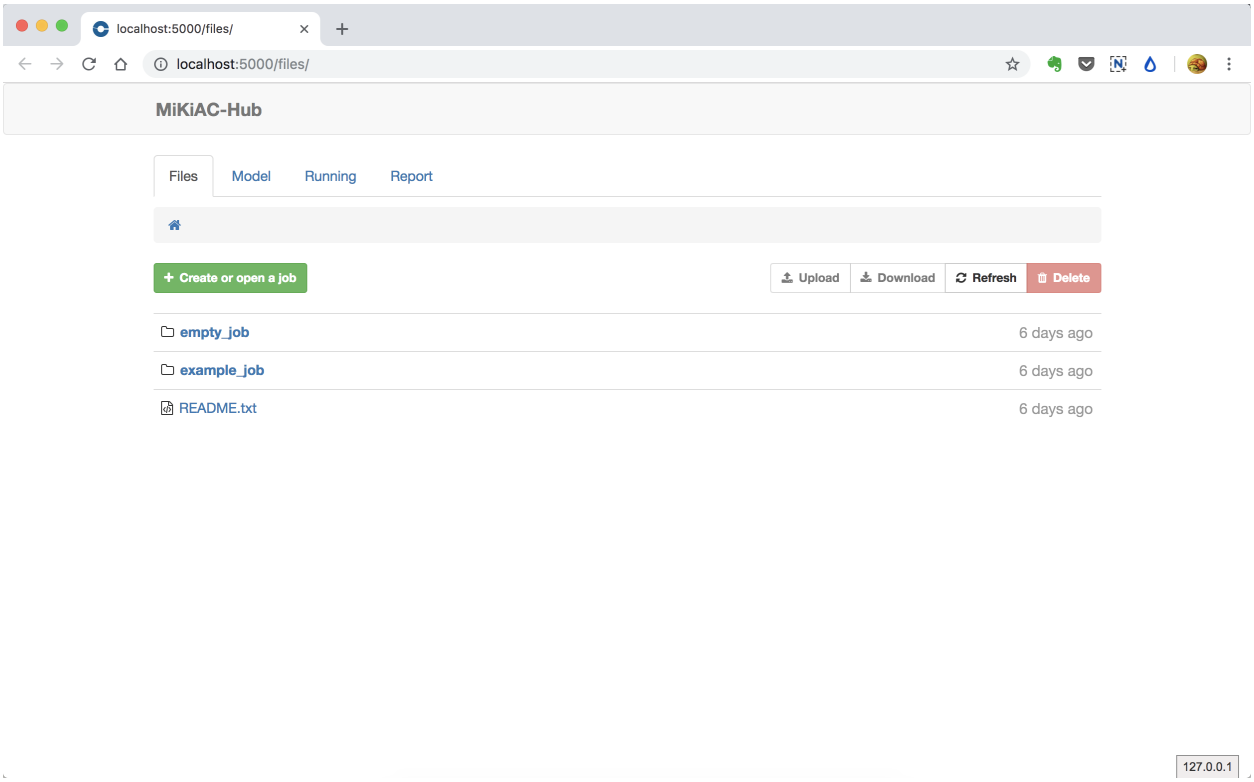


3.2 Solve model in scaks-Hub

scaks also has a web application named scaks-Hub to help researcher build and solve micro-kinetic model more easily.

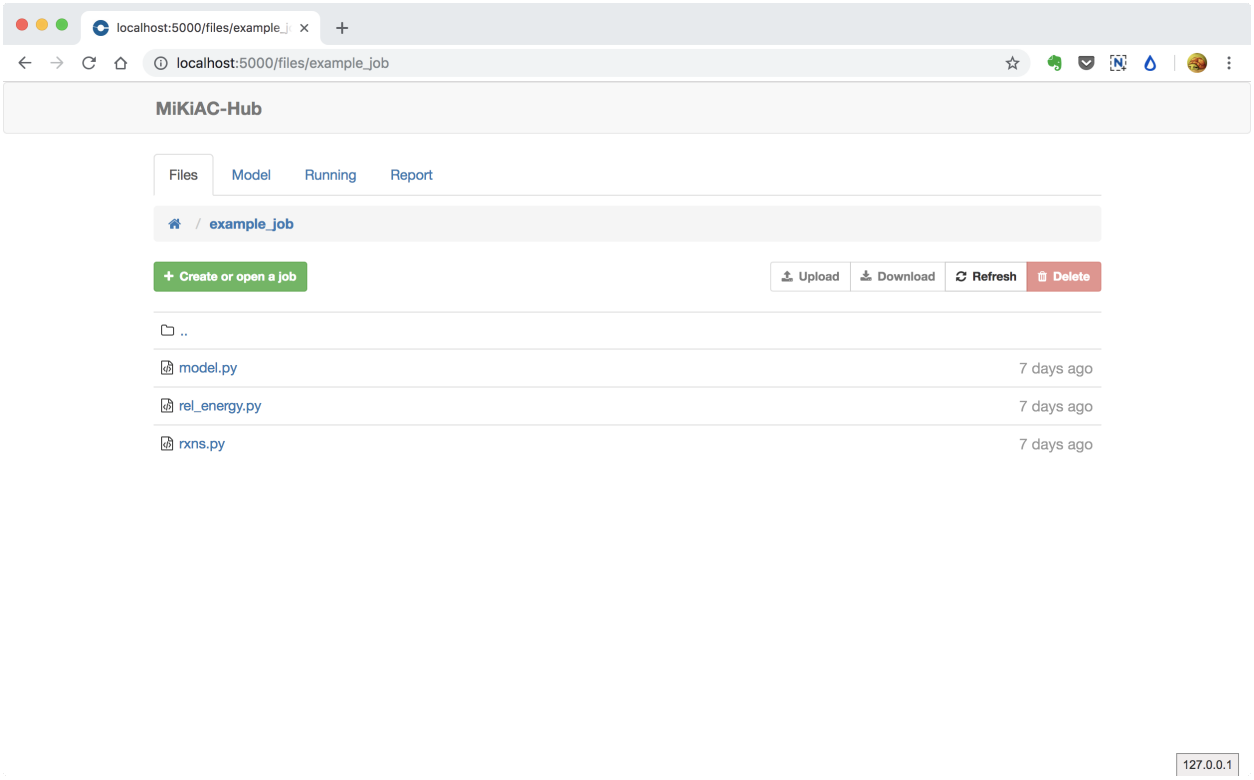
3.2.1 File system

scaks-Hub uses a file explorer as the main user interface

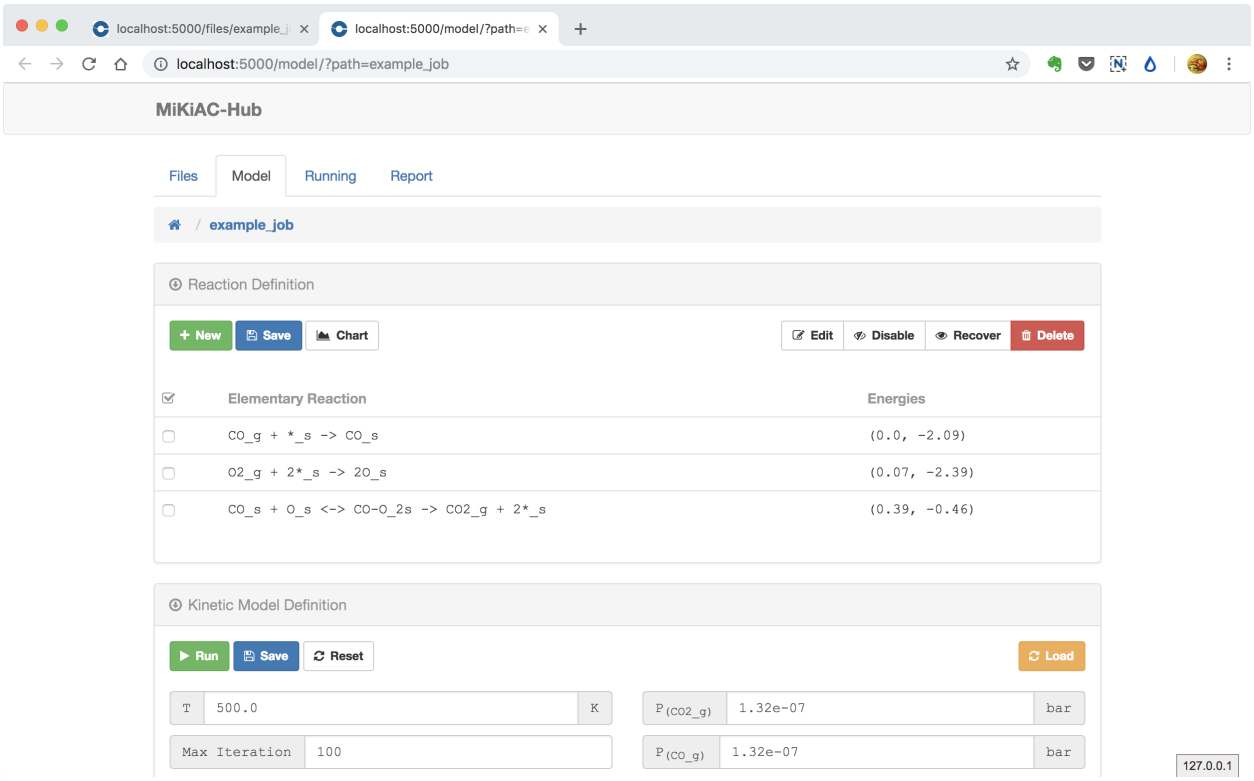


3.2.2 Open an existing model in model panel

Now you can enter the `example_job` directory where model setup file and energy data file have already existed.



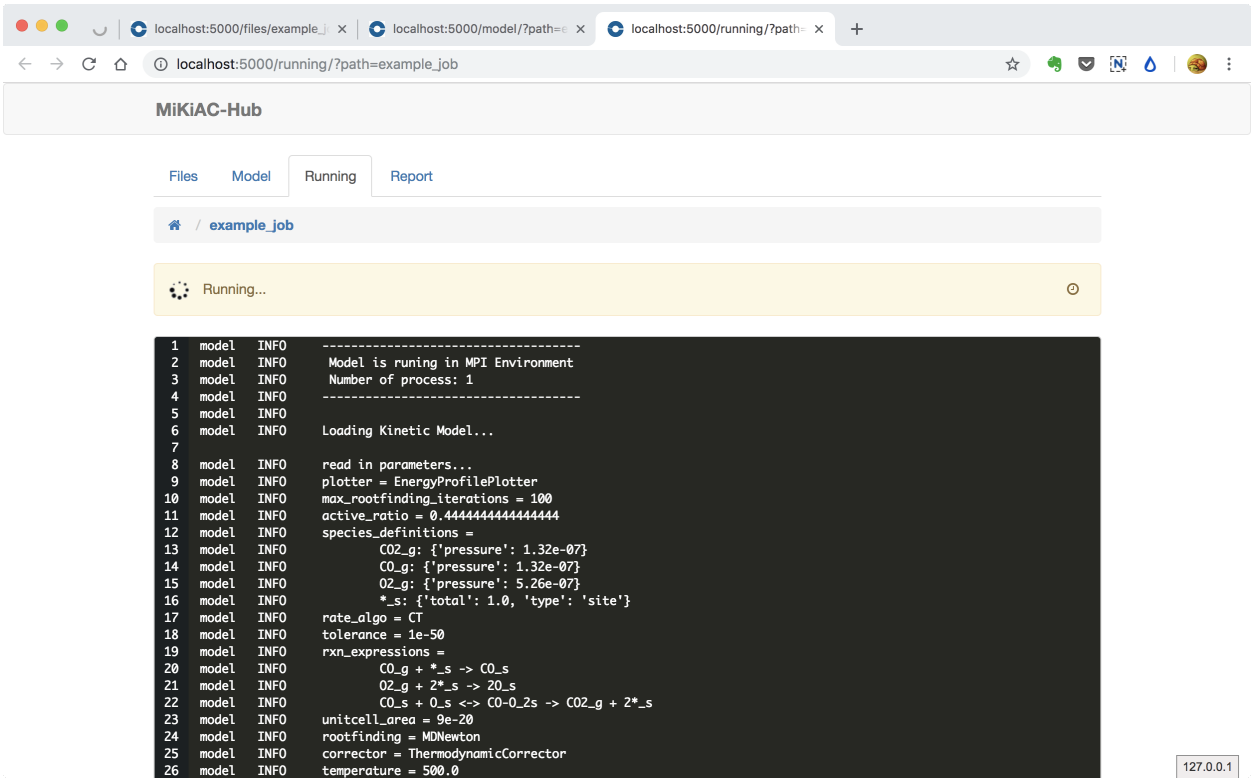
Then you can click the green button to open the job panel. scaks-Hub will read all those files and fill the form in model panel automatically.



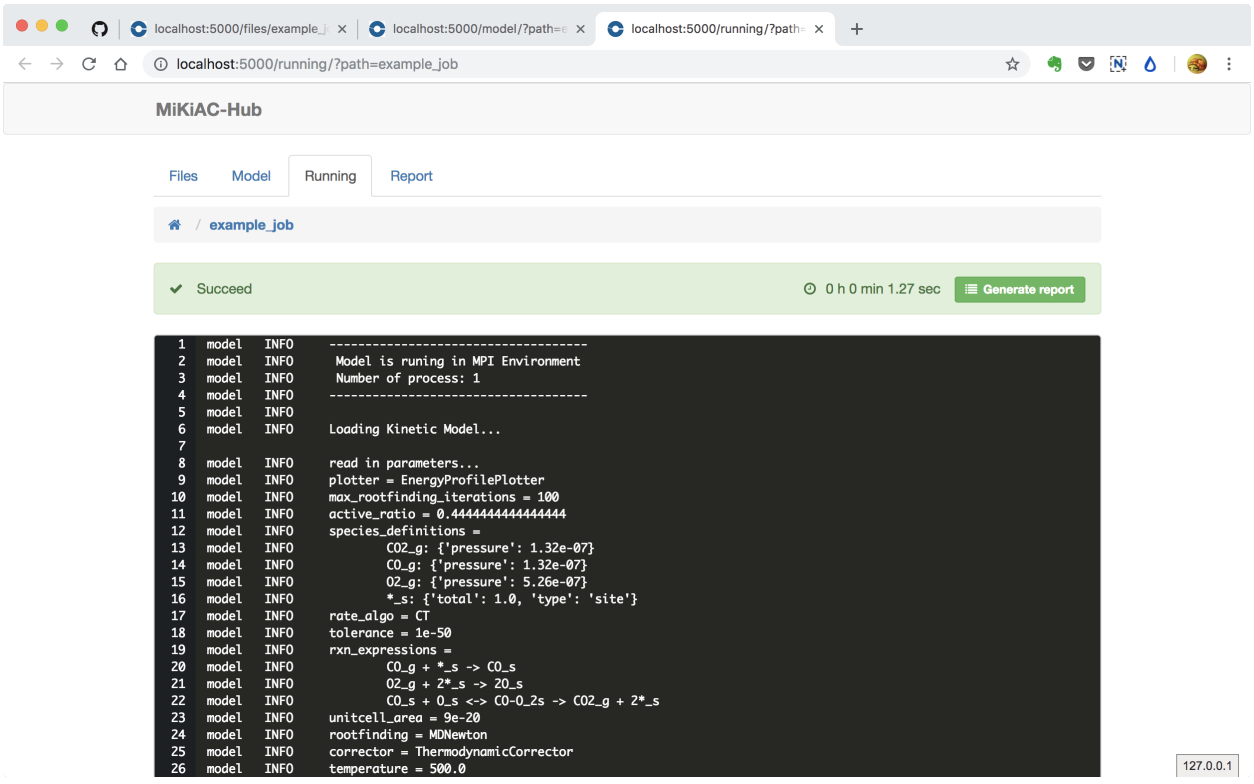
3.2.3 Run a job

After all inputs prepared, you can click the green Run button to call scaks core to solve current model. Then the running panel will be opened and all solving information will be continously updated and displayed in the code block.

The job is running:

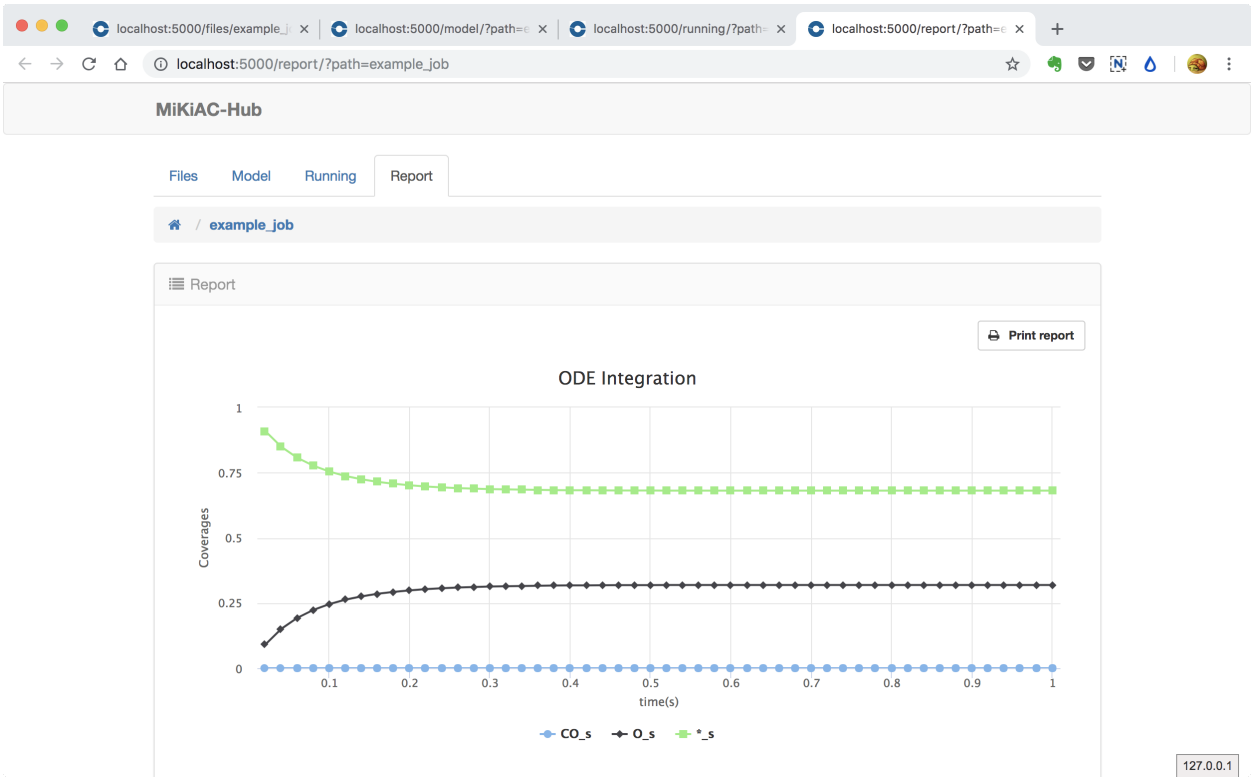


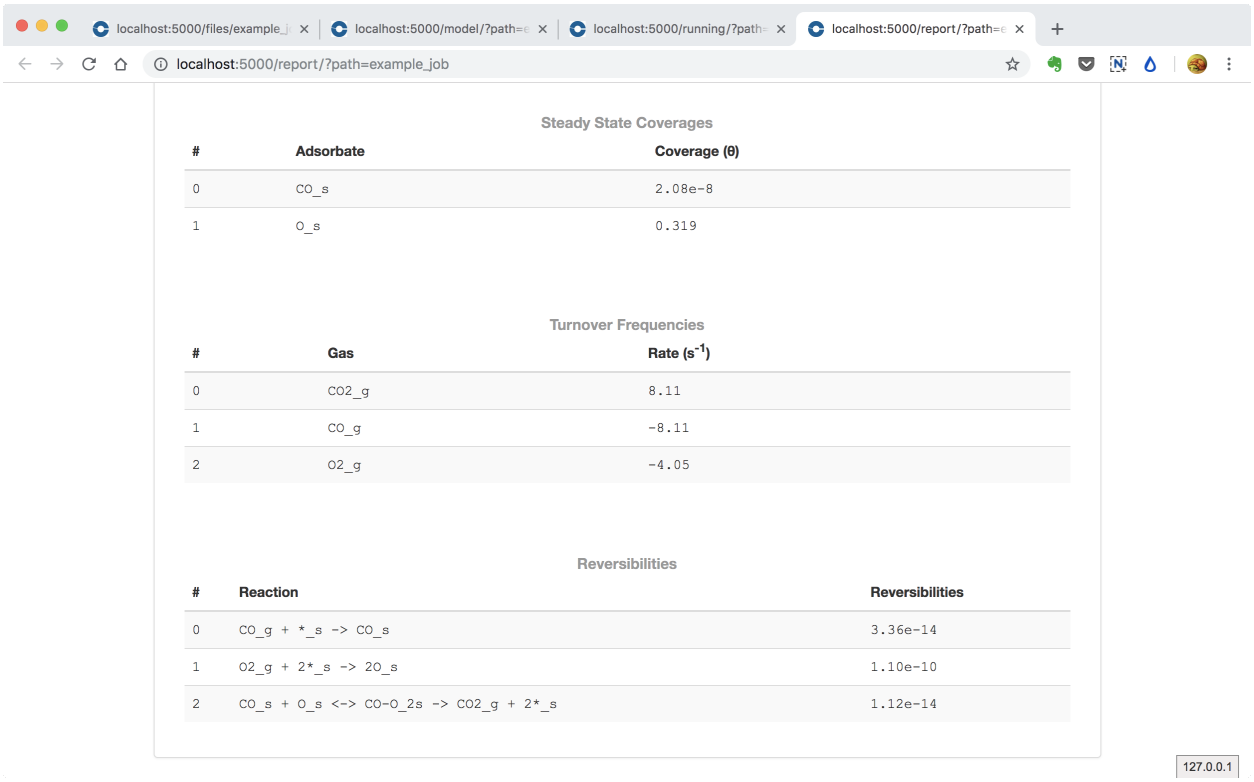
The job is finished:



3.2.4 Generate job report

As the log information is not very friendly for users, scaks-Hub provides a report for each completed job. You can click the green Generate report buttong, then a report panel will be opened. All related results such as ODE integration trajectory, steady state coverages, reversibilities and turnover frequencies.





3.3 Parse reaction expressions

scaks has built in a powerful reaction expression parser based on Python's regular expression engines. With the help of reaction parser, you can check the site and mass conservation to make sure scaks can receive correct reaction expressions and solve correctly.

3.3.1 Species name convetions in scaks

All species expressions in scaks consist of two parts connected with a underline _: species name and site name.

For example, a `CO` gas molecule adsorbed on a site named `s` can be written as `CO_s`.

For the transition state species in scaks, it must have a − character indicating a bond like `O-O_2s`.

Below are some common species expression examples:

- `O2_2s`: A O2 (oxygen molecule) adsorbed on two `s`sites
- `CO_b`: A CO (carbon dioxide molecule) adsorbed on one `b` site
- `O-O_2t` A transition state for O2 dissociation on two `s` sites

3.3.2 Reaction expression conventions in scaks

Each elementary reaction expression consists of different states: initial state, transition state(optional), final state.

For example, a CO adsorption on a single `s` process can be expressed as:

```
CO_g + *_s -> CO_s
```

As we can see, we can use arrows <-> or -> to connect all states to an elementary reaction expression.

If a process is a chemical reaction, usually it has a transition state in the middle. For example, a O2 dissociation adsorption process can be written as:

```
O2_g + 2*_s <-> O-O_2s -> O_s + O_s

# or

O2_g + 2*_s <-> O-O_2s -> 2O_s
```

3.3.3 Parse an elementary reaction

`scaks.parser.rxn_parser` provide three main classes to represent different levels in a reaction expression: `RxnEquation` for reaction equation expression, `ChemState`for chemical state and `ChemFormula` for chemical formula.

In this section, we use scaks reaction parser API to parse elementary reaction expression using a O2 dissociation adsorption process as an example.

Create reaction equation

```
from scaks.parsers.rxn_parser import RxnEquation

rxn_expression = 'O2_g + 2*_s <-> O-O_2s -> 2O_s'

# Create a reaction equation object
rxn = RxnEquation(rxn_expression)
```

Revert equation

```
reverse_rxn = rxn.revert()
reverse_rxn.rxn_equation()  # Output: '2O_s <-> O-O_2s -> O2_g + 2*_s'
```

Check conservation

```
rxn.check_conservation()  # return True
```

If we construct an unconserved reaction equation, an exception will be raised.

3.3.4 Balance a set of elementary reactions

Reaction parser in scaks can help researchers to balance a set of elementary reactions to obtain the total reaction equation.

Here we take CO Oxidation as an example, the elementary reactions are listed below:

```
[
    'CO_g + *_s -> CO_s',
    'O2_g + 2*_s <-> O-O_2s -> 2O_s',
    'CO_s + O_s <-> CO-O_2s -> CO2_g + 2*_s'
]
```

After model has been built, we can get the total reaction expression:

```
model.parser.get_total_rxn_equation()
```

the output would be:

```
'2CO_g + O2_g -> 2CO2_g'
```

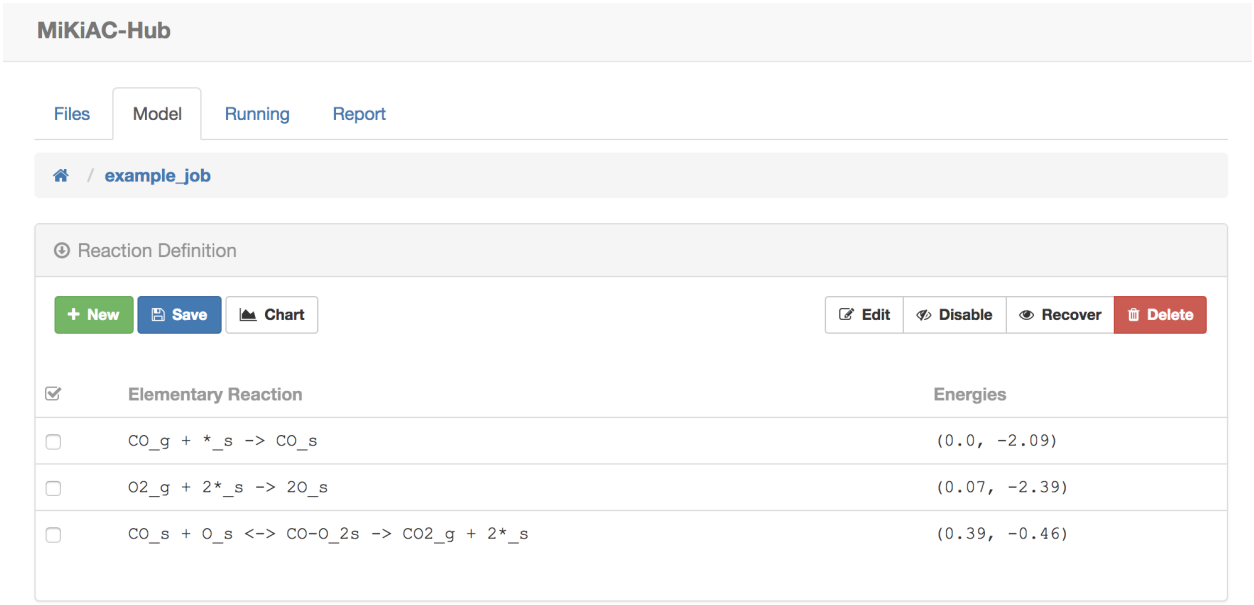
3.4 Build a model in scaks-Hub

In this section, we introduce how to build a micro-kinetic model in scaks-Hub. All information needed for model construction are input in model panel of scaks-Hub. There are two main sub-panel in model panel:

1. Reaction Definition panel: input reaction and energy information
2. Kinetic Model Definition panel: input model conditions and parameters such as temperature, algorithm type and so on.

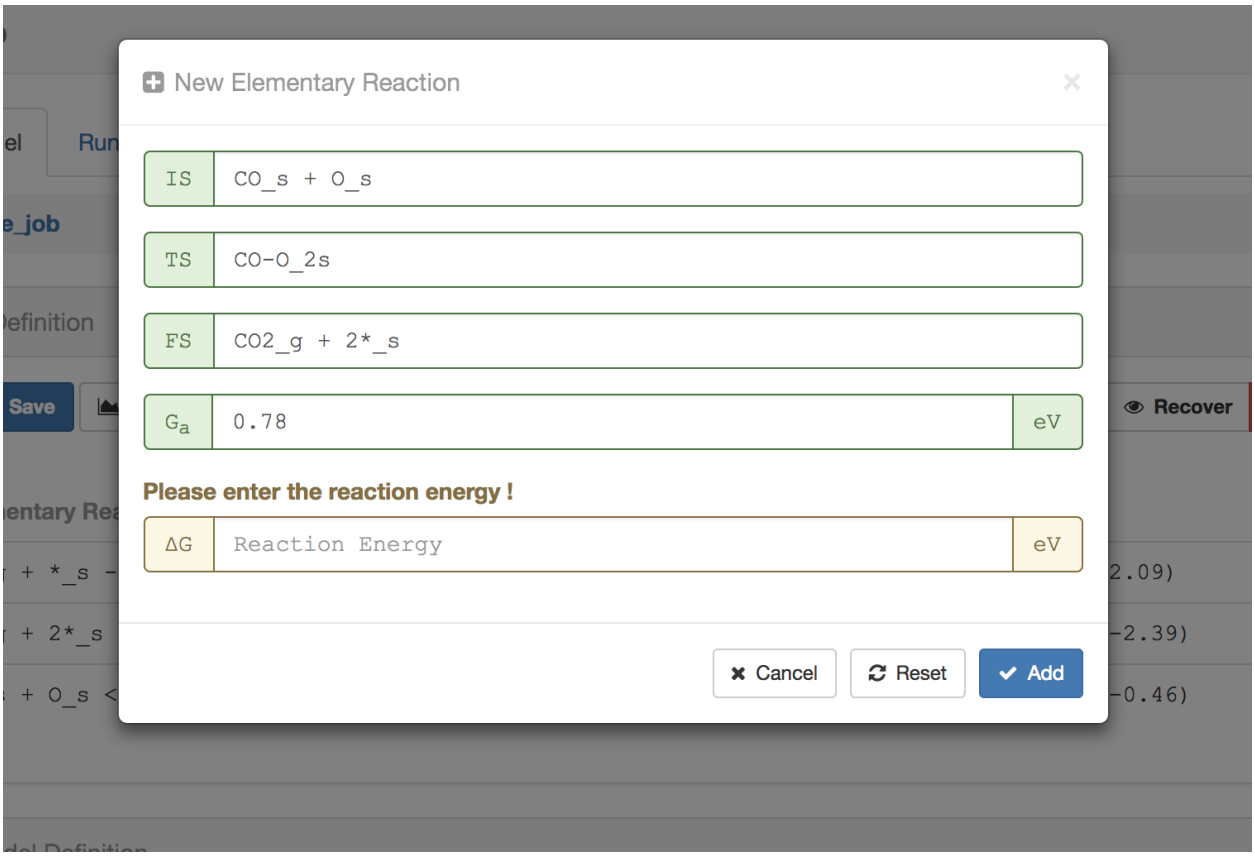
3.4.1 Reaction Definition panel

You can add, edit, hide and delete elementary reaction and energy information in Reaction Definition panel



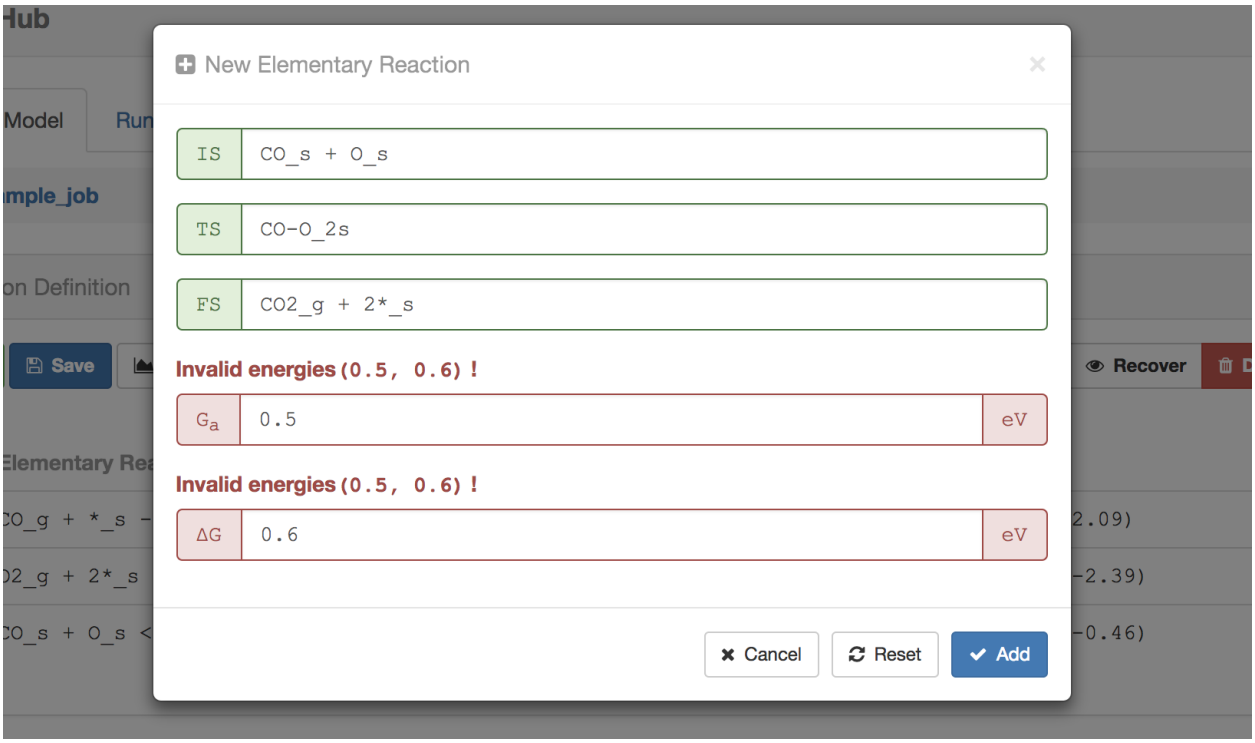
Add new elementary reaction

Click the **+ New** button on upper left, a reaction definition modal would pop up for inputing elementary reaction information.

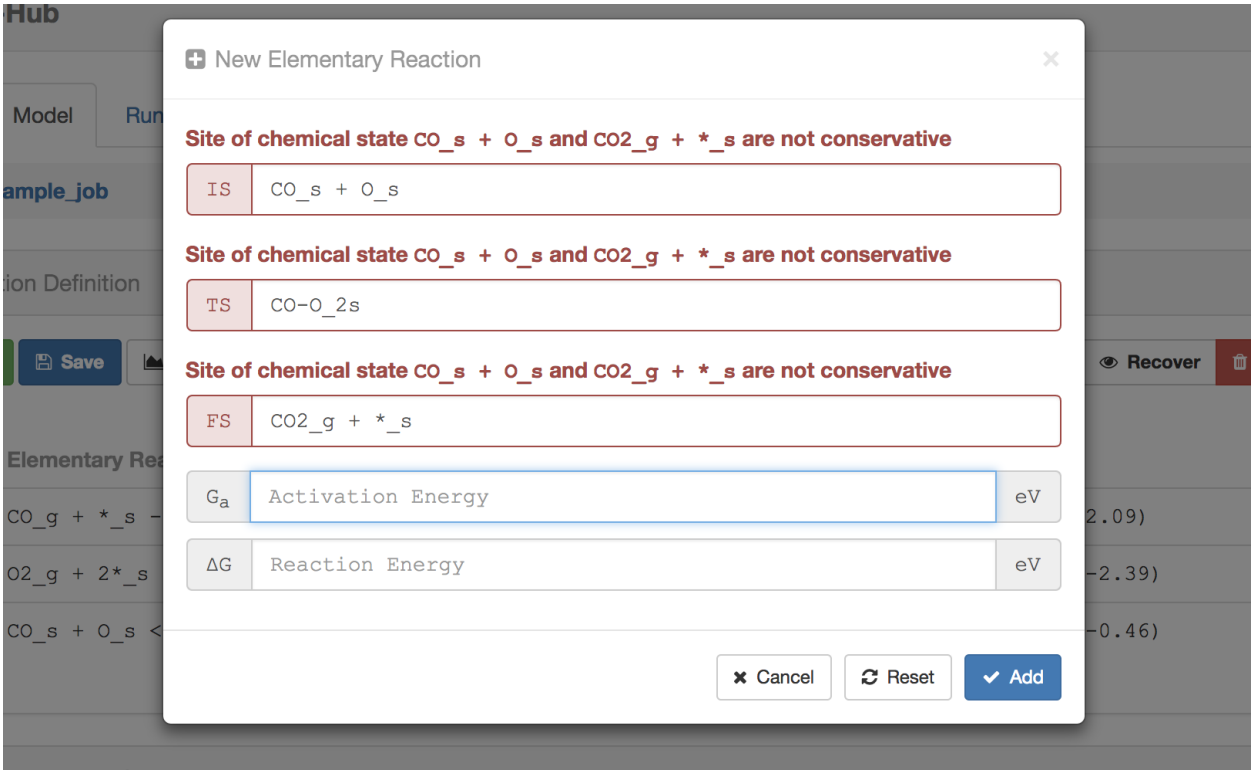


scaks-Hub also can check the correctness of the input reaction expression and energy values.

Energy barrier is less than reaction energy:

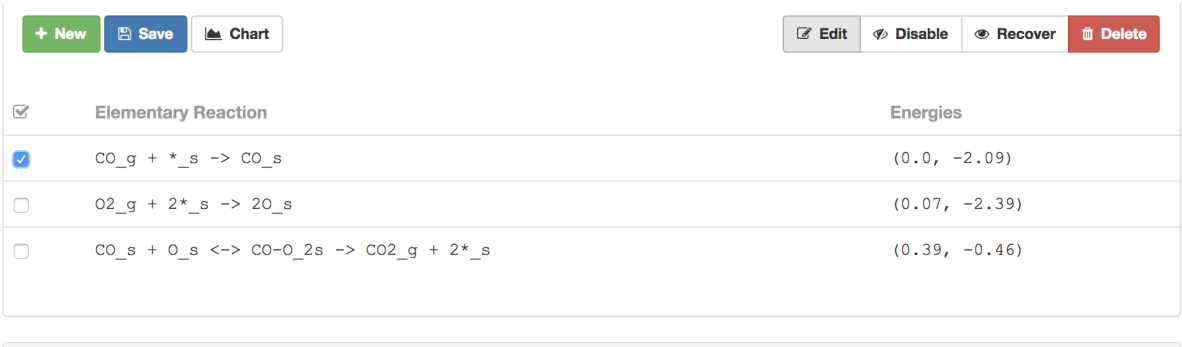


Input invalid reaction expression:



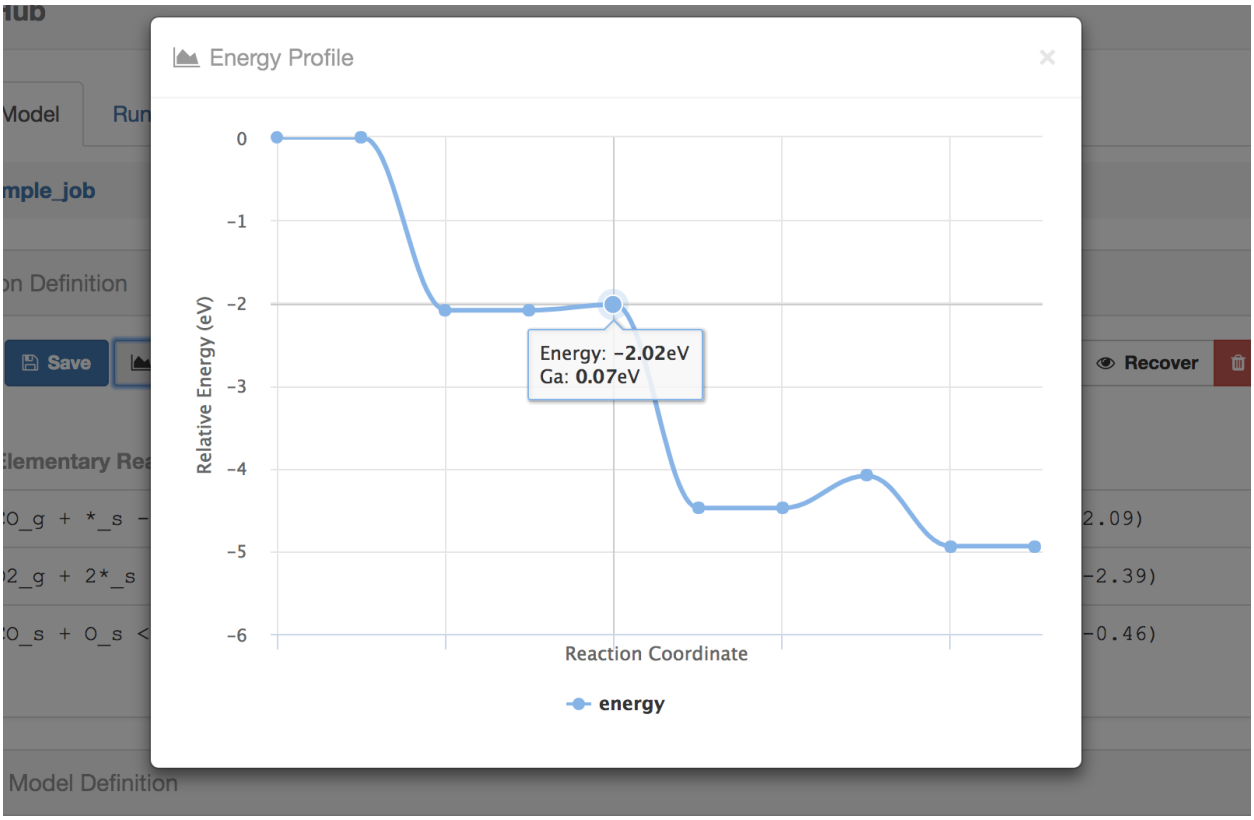
Edit an existing reaction

If you want to edit an existing reaction, just select its select box at the beginning of the reaction information line and click the **Edit** button, a similar modal with reaction addition will appears.



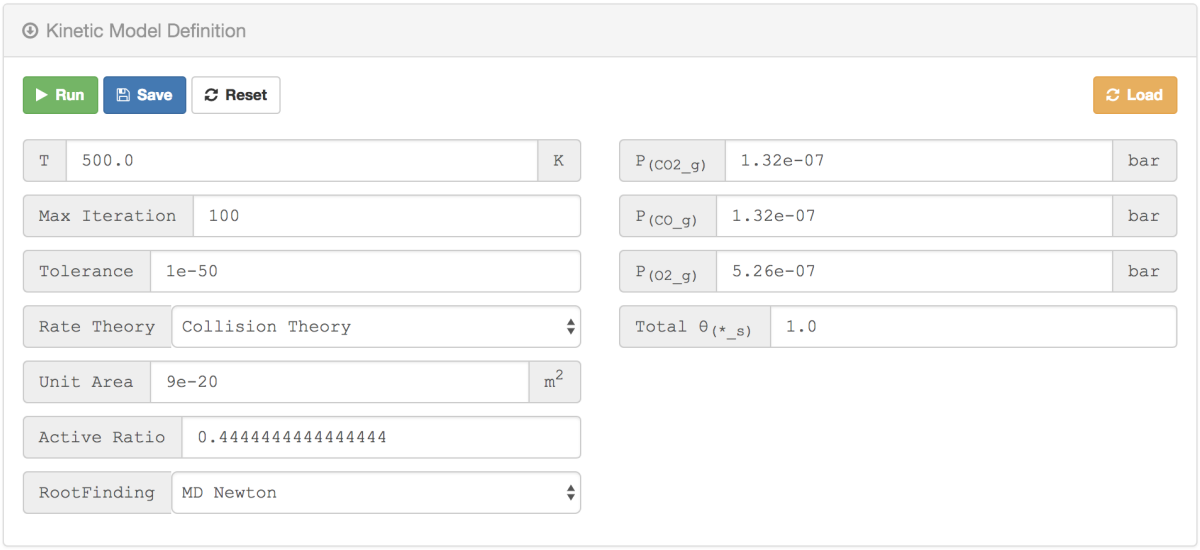
Visualize energy profile

Besides operating elementary reactions, you can also visualize the energy profile for selected reactions:



3.4.2 Kinetic Definition panel

Kinetic Definition panel is used to input model related parameters such as temperatures, partial pressures for gases, total coverage and solving iteration related parameters



3.5 [Animation] Run a job in scaks-Hub

Here we use GIF animiation to show operations for running a job in scaks-Hub.

- File system in scaks-Hub
- Open an existing job and edit reactions in Reaction Definition panel
- Visualize energy profile for different elementary reaction set
- Edit an existing reaction, scaks-Hub can parse it and do conservation check
- Set all model parameters and run a job to solve the model
- After the job is completed, generate a job report for that run

3.6 Register hybrid method for Hybrid Newton iteration

SCAKS supports more Pythonic way to register your own hybrid method to our Hybrid Newton iteration flow.

SCAKS provides a `hybrid_method_register` Python decorator for you to define and register your custom hybrid method to generate new initial coverages for next Newton's iteration. For example, if you have create a micro-kinetc model `model`, you can

```
@model.hybrid_method_register
def ODE_integration(model, N):
    if model.log_allowed:
        model.logger.info('Use ODE integration to get new initial coverages...')

    end = 10
    span = 1e-2
    init_cvgs = model.solver.coverages

    new_cvgs = model.solver.solve_ode(time_end=end,
                                      time_span=span,
                                      initial_cvgs=init_cvgs)[-1]

    if model.log_allowed:
        model.logger.info('generate new initial coverages - success')

    return new_cvgs
```

Here there are two arguments for any hybrid method:

1. `model`: current micro-kinetic model instance You can obtain different components like parser, solver ... to give you essential data for new coverages guess.
2. `N`: hybrid method attemp times Sometimes the coverages guess is related to the guess attemp time, e.g. the ODE integration strategy, `N` could be used for this purpose.

The return value must be a list containing new coverages guess.

Then when the model starts to run, the registered hybrid method would be invoked everytime the Newton's method fails to converge.

3.7 Define your own on-the-fly analysis plugin

For more flexible analysis the iteration process, SCAKS provides a plugin mechanism that allow users to be able to define their own plugins and register them to the model. With this, we can output or dump any data before, during or after the Newton iteration.

It's pretty easy for you to define your own plugin. Here we write a on-the-fly analysis plugin to dump the errors during the Newton iteration for visualization or other purpose.

1. create a 'MicroKinetcModel`name`model' as we described in other tutorial
2. Inherit the `scaks.plugins.analysis.OnTheFlyAnalysis` class to define custom analysis class

```
from scaks.plugins.analysis import OnTheFlyAnalysis

@model.analysis_register
class DumpTrajectory(OnTheFlyAnalysis):
    interval = 1

    def setup(self, model, outer_counter):
        self.errors = []

    def register_step(self, model, inner_counter, outer_coutner):
        self.errors.append(float(model.solver.error))

    def finalize(self, model, outer_counter):
        with open('newton_traj.py', 'w') as f:
            content = 'errors = {}\n'.format(self.errors)
            f.write(content)
        model.logger.info('Dump newton iteration trajectory to newton_traj.py')
```

3. Just run the model, your analysis methods in plugin will be invoked automatically. Enjoy it!

CHAPTER 4

API

4.1 models.KineticModel

4.2 models.MicroKineticModel

4.3 parsers.rxn_parser

4.4 parsers.ParserBase

4.5 parsers.AbsoluteEnergyParser

4.6 parsers.RelativeEnergyParser

4.7 parsers.KMCParser

4.8 solvers.SolverBase

4.9 solvers.MeanFieldSolver

4.10 solvers.rootfinding_iterators

4.11 solvers.MeanFieldSolver

4.12 correctors.ThermodynamicCorrector

4.13 plotters.EnergyProfilePlotter

4.14 utilities.check_utilities