catalysis-hub.org Documentation

Kirsten Winther, Max J. Hoffmann

Mar 22, 2019

Contents

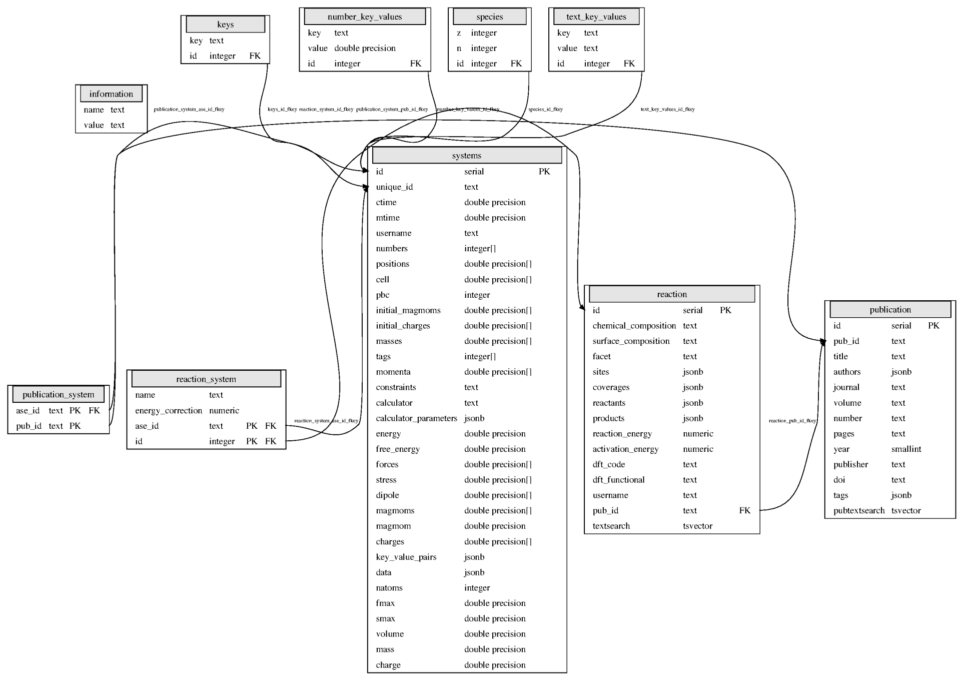
CHAPTER 1

Topics

Below you can find documentation about the database structure and GraphQL.

1.1 Database schema

An overview of the database schema is given in the figure below.



The database structure builds upon the ASE-database (https://wiki.fysik.dtu.dk/ase/ase/db/db.html) that uses the tables: *systems*, *species*, *keys*, *text_key_values*, *number_key_values*, *information*. These are used for storing atomic structures and calculational information.

On top is the tables *reaction* and *publication* which are used to store reaction energies and publication info for CatApp v2.0.

The tables *reaction_system* and *publication_system* links the ASE and CatApp parts together.

1.2 GraphQL Queries

Go to the backend interface at http://api.catalysis-hub.org/graphql to start using our graphQL browser.

Type your query in the left panel. In order to perform queries on the *reactions* table start with:

```
{reactions(first:2)}
```

And type *command* + *return* to see the result on the right hand. This should return the *id* of the first two reactions in the database. Notice that the left hand side is updated as well.

See the tutorials at http://catalysis-hub.readthedocs.io/en/latest/tutorials to learn more.

1.3 GraphQL Cheat Sheet

Tables:

- reactions
- publications
- systems
- reactionSystems

Start your query with the table name followed by a query:

```
{table(field:value)}
```

Query fields:

- Text fields:
  - field=value: (field: “value”)
  - field@>value: (field: “~value”)
  - distinct field: (field: “~”, distinct: true)
- Integer / Float fields:
  - field=value: (field: value)
  - field>value: field:value, op: ”>”)
- Special fields:
  - first, last : int
  - distinct: true/false
  - before, after : str id
  - order: sort by column: (order: “field”) or (order: “-field”)
  - op: [‘=’, ‘>’, ‘<’, ‘>=’, ‘<=’, ‘!=’]
- Output fields (systems table)
  - InputFile(format: ”vasp”)
- Special attributes:
  - totalCount: # entries
  - pageInfo: pagination

Incude the special attributes like this:

```
{table(<field>:<value>){
  totalCount
  pageinfo
  edges{
    node{
      id
    }
  }
}}
```

CHAPTER 2

Submitting data

Please submit your electronic structures calculations for surface reactions! Data submissions will be part of the Surface Reactions app at http://www.catalysis-hub.org/energies (essentially CatApp v2.0) and is open to all institutions. Furthermore, the atomic structures that are part of your reaction can be utilized for other apps.

Your publication/dataset will be listed on the http://www.catalysis-hub.org/publications page, with a link to the publishers homepage (if already published). Your data will be easily accessible to other researchers, who will be able to browse through reaction energies and atomic structures from your publication.

2.1 SUNCAT group members

If you're a member of the SUNCAT group, you can add your data to one of the folders on the Sherlock or SUNCAT cluster:

- SHERLOCK2 : /home/users/winther/data_catapp/
- SUNCAT : /nfs/slac/g/suncatfs/data_catapp/

Start by activating the corresponding virtualenv. On SHERLOCK2

```
. /home/users/winther/data_catapp/CATHUBENV/bin/activate
```

or on SUNCAT

```
. /nfs/slac/g/suncatfs/data_catapp/CATHUBENV/bin/activate
```

to use the local installation of CatHub. You should see a *(CATHUBENV)* at the beginning of your prompt now to indicate that all python script and libraries are first imported from the virtualenv. To return your shell to the previous state simply type:

```
deactivate
```

or log out. Now you can go straight to *cathub organize* .

Alternatively you can install your own version of CatHub - see instructions below.

2.2 Installing CatHub

CatHub is a python module that is used to interface the Surface Reactions database of Catalysis-Hub, directly from a python script of the command line. CatHub will be used to arrange your data into folders and submit your data to the server. To install CatHub, together with the ASE dependency, use pip:

```
pip install git+https://github.com/kirstenwinther/CatHub.git#egg=cathub --upgrade --
↪user --process-dependency-links
```

which will install a developer version of ASE with an enhanced database module, CatHub and all their dependencies.

To test that the cathub cli is working, start by typing:

```
cathub
```

And you should see a list of subcommands. If it's not working you probably have to add the installation path to `PATH` in your ~/.bashrc. This would typically be `export PATH=~/.local/bin:${PATH}` for Linux, and `export PATH~/Library/PythonX.Y/bin:${PATH}` for Mac.

2.3 Organizing data

You have two options for organizing your data:

- cathub organize: For larger systematic datasets without reaction barriers, this approach will create folders and and arrange your data-files in the right location for you.
- cathub make_folders: For smaller or more complicated datasets with reaction barriers, this method will only create your folders, and you will have to drop the files in the right location yourself.

In either case no data will be uploaded to catalysis-hub.org/publications before you run *cathub db2server* . . . . Once you uploaded data it will be held in a moderation stage which you can inspect yourself at catalysis-hub.org/upload and delete yourself to iterate until all structures and energies look as expected. Once you are satisfied with your uploaded dataset there will be a "Release" button that will notify the platform administrator that the dataset is ready for release.

2.4 cathub organize

This tool will take all your structure files from a general folder and organize them in the right folder-structure that can be used for data submission. Note: this approach does not work for transition states / barrier calculations. And it will still need a lot of manual file organization for co-adsorbate configurations. While we are working on this cathub organize might still give you a nice head start with file organization.

To learn about the organize command, type:

```
cathub organize --help
```

To read the data from a general folder, type:

```
cathub organize <FOLDER> -a ADS1,ADS2 -c <dft-code> -x <xc-functional> -f <facet> -S
↪<crystal structure>
```

Use the `-a` option to specify which adsorbates to look for. Also, please use the `-c` and `-x` options to specify the DFT code and xc-functional respectively. Furthermore, you are highly encurraged to use the `-f` and `-S` options to specify the surface facet and crystal structure when applicable.

This will generate an organized folder named `<FOLDER>.organized`. Please open the .txt file `<FOLDER>.organized/publication.txt`, and update it with info about your publication. It should look something like this:

```
volume: 8
publisher: Wiley
doi: 10.1002/cssc.201500322
title: "The Challenge of Electrochemical Ammonia Synthesis: A New Perspective on the␣
↪Role of Nitrogen Scaling Relations"
journal: ChemSusChem
authors: [Montoya, Joseph H., Tsai, Charlie, Vojvodic, Aleksandra, Norskov, Jens K.]
year: 2015
email: winther@stanford.edu
number: 13
tags: []
pages: 2140-2267
```

Remember your `email` since it will be used to log in at http://www.catalysis-hub.org/upload. Note that authors should be a list, with names in the form "lastname, firstname M.".

Please go through the created folder and rename folders to make your data easier to localize later. For example, a structure folder like Pt16_structure, could be changed to Pt16_fcc or Pt16_bcc respectively. Please do not use spaces in folders or file-names.

If you, for example, have calculations with different facets, you can also split them into separate folders, run `cathub organize -f <facet>`, and them merge the organized folders together afterwards with `cp -R organized1 organized2`.

2.5 cathub make_folders

An alternative to `cathub organize`. This tool will create the right folder structure for you, but you must dump your files yourself.

To learn about the make_folders command type:

```
cathub make_folders --help
```

Then create a folder in your user-name, 'cd' into it and type:

```
cathub make_folders --create-template TEMPLATE_NAME
```

This will create a template (txt/yaml) file, that you should update with your publication and reaction info. The template should look similar to this:

```
reactions:
-   reactants: [2.0H2Ogas, -1.5H2gas, star]
    products: [OOHstar@top]
-   reactants: [CCH3star@bridge]
    products: [Cstar@hollow, CH3star@ontop]
-   reactants: [CH4gas, -0.5H2gas, star]
    products: [CH3star@ontop]
journal: JACS
year: 2017
email: winther@stanford.edu
number: 1
crystal_structures: [fcc, hcp]
```

(continues on next page)

(continued from previous page)

```
volume: 1
DFT_functionals: [BEEF-vdW, HSE06]
authors: ['Doe, John', 'Einstein, Albert']
pages: 23-42
publisher: ACS
doi: 10.NNNN/....
title: "Fancy title"
bulk_compositions: [Pt]
DFT_code: Quantum Espresso
facets: ['111']
```

Consult `cathub make_folders --help` again for detailed instructions on how to specify the types of reactions and surfaces.

Then type:

```
cathub make_folders <TEMPLATE>
```

And your folders will be created. You can check that they look right with `tree -F <FOLDER>`. The template above will produce the following folder tree:

```
$tree -F MontoyaChallenge2015/

MontoyaChallenge2015
├── Quantum\ Espresso/
│   └── BEEF-vdW/
│       ├── Co_fcc/
│       │   ├── 111/
│       │   │   ├── 0.5H2gas_star__Hstar@bridge/
│       │   │   │   ├── MISSING:H_slab
│       │   │   │   └── MISSING:TS?
│       │   │   ├── 0.5H2gas_star__Hstar@fcc/
│       │   │   │   ├── MISSING:H_slab
│       │   │   │   └── MISSING:TS?
│       │   │   ├── 0.5H2gas_star__Hstar@hollow/
│       │   │   │   ├── MISSING:H_slab
│       │   │   │   └── MISSING:TS?
│       │   │   ├── 0.5H2gas_star__Hstar@ontop/
│       │   │   │   ├── MISSING:H_slab
│       │   │   │   └── MISSING:TS?
│       │   │   ├── 0.5N2gas_0.5H2gas_star__NHstar@bridge/
│       │   │   │   ├── MISSING:NH_slab
│       │   │   │   └── MISSING:TS?
│       │   │   ├── 0.5N2gas_0.5H2gas_star__NHstar@hollow/
│       │   │   │   ├── MISSING:NH_slab
│       │   │   │   └── MISSING:TS?
│       │   │   ├── 0.5N2gas_star__Nstar@hollow/
│       │   │   │   ├── MISSING:N_slab
│       │   │   │   └── MISSING:TS?
│       │   │   └── MISSING:empty_slab
│       │   └── MISSING:Co_fcc_bulk
│       └── gas/
│           ├── MISSING:H2_gas
│           └── MISSING:N2_gas
└── publication.txt
```

Then add your atomic structure output files to the right folders. The files can be in any format that ASE can read, and must contain the total potential energy from the calculation. ASE trajectory (.traj) files are generally preferred. If you're using Vasp, please add your OUTCAR files as `<name>.OUTCAR`. Your structures will include the adsorbed atoms/molecules, empty slabs, and gas phase species for your reactions. Also, if you have done calculations for the bulk geometries, please include them as well. All gas phase species involved must be added to the `<publication>/<dft code>/<dft functional>/gas/` folder. Also, notice that dummy files named `MISSING:..` have been placed in the folders, to help you determine the right location for your files. It's recommended to write a script to transfer files from one folder-structure to another in a systematic way, for example using `shutils.copyfile('/path/to/initial/file', '/path/to/final/file')`.

2.6 Reading into database

After adding all your structure files (or after running cathub organize), read your structures into a local database file with the command:

```
cathub folder2db <FOLDER>
```

If anything is wrong with your files, or anything is missing, you should receive appropriate error messages. When reading of the folder is complete, a table with a summary with reaction energies will be printed in you terminal. Please verify that everything looks right. Also, a database file has been written at `<FOLDER>/<DBNAME>.db`.

Upload your data to the server by typing:

```
cathub db2server <DBNAME>.db
```

and follow the feedback in the terminal. Your data will not be made accessible from catalysis-hub.org before you have approved. Send an email to Kirsten Winther, winther@stanford.edu, and request to have your data made public. Please include the email you defined above.

CHAPTER 3

Tutorials

3.1 Searching for reaction energies

Here you will learn how to use the webpage to search for chemical reactions. Go to http://www.catalysis-hub.org/energies to use the build in search function. This feature corresponds to *CatApp v2.0* where you can access reaction energies and barriers for different surfaces.

1) Type something in the reactants field and press the search button (for example CH3CH2OH*). How much is returned? Try to decrease the number of reactions by filling out the `Products`, `Surface` and `Facet` fields.
2) Look at the atomic structures associated with a reaction of choice. Hint: press the cube icon to the left of the reaction. (Note: a few of the reactions doesn't have structures associated with them).
3) Interested in knowing how the data is pulled from the database backend? Click the `GRAPHQL QUERY` button below the list of reactions, and see how you search looks from the backend interface.

3.2 GraphQL

These tutorials will focus on how to use the GraphQL interface to the database. You might also want to read the documentation at http://catalysis-hub.readthedocs.io/en/latest/topics .

Go to http://api.catalysis-hub.org/graphql to get started.

3.2.1 A simple query

Type your query in the left panel. In order to perform queries on the *reactions* table try this:

```
{reactions(first:2)}
```

And type *command* + *return* to see the result in the right panel. This should return the *id* of the first two reactions in the database. Notice that the left hand side is updated as well.

A general note: Always include one of the 'first' or 'last' field in your query to limit the number of results. (Otherwise things could get slow!).

Now try to add more columns after *id* and see what happens. For example:

```
{reactions(first:2) {
  edges {
    node {
      id
      Equation
      chemicalComposition
      reactionEnergy
    }
  }
}}
```

For a complete list of all the tables in the database, and the associated columns, see the *Docs* tab on the top right of the GraphiQL page. There is also a schema overview posted at http://docs.catalysis-hub.org/reference/schema.html .

In order to make selections on the result, add more fields after `(first:2)`. For example:

```
{reactions(first:2, reactants: "CO", chemicalComposition: "Pt")}
```

Notice that it's possible to construct queries for all the existing columns in a table.

3.2.2 Searching for publications

1) Find all titles and DOI's from publications with year=2017.
2) How many publications are there with year>2015?
3) How many publications are authored by Thomas Bligaard? Hint: use the pubtexsearch field. You can list the total number of results using the *totalCount* field:

   ```
   {publications {
     totalCount
     edges {
       node {
         id
         authors
       }
     }
   }}
   ```

   Verify that you get the same result by using `(authors:  "~bligaard")`
4) Find Michal Bajdich's paper with "Oxygen Evolution" in the title.

3.2.3 Using the reactions table

1) Order all the reactions with respect to increasing reaction energy and print out the first 100 results.
2) Find the reactions with the lowest activation energy. Hint: take care of 'null' results by requesting that the activation energy should be > 0.
3) Find the number of reactions with H2O on the left hand side and OH on the right hand side. How many distinct reactions does that give rise to? What happens when you add the state (star or gas) to your query? What happens when you add `chemicalComposition:  "~"`

4) Chose a few of the reactions from the query before, and get all the chemical formula of the atomic structures beloning to them Hint: you can call the 'systems' table inside the 'reactions' table.
5) Find the publication year for the first 10 reactions in exercise 1)

3.2.4 Combining tables

1) Find Julia Schuman's recent paper, and list all the reactions belonging to the paper. Hint: you can either go through the publication table:

```
{publications(first:10) {
  edges {
    node {
      id
      title
      pubId
      authors
      reactions
      }
    }
}}
```

   or use the *pubId* field to query directly on the reactions table to make additional queries:

```
reactions(pubId: "")}
```

2) Using the `(pubID:)` solution suggested above, list all the distinct a) reactions b) surfaces from Julia's publication.
3) Chose one of Julia's reactions and find the *aseId* of the empty slab. Hint: It has `"name"="star"` in the *reactionSystems* table. Copy the aseId and use it to find all the reactions that are linked to that particular empty slab.

3.3 Custom GraphQL clients

3.3.1 Calling the Backend from the Command Line

This is easy using *curl* or *wget*:

```
curl -XPOST https://api.catalysis-hub.org/graphql --data 'query={systems(last: 10 ) {
  edges {
    node {
      energy Cifdata
          }
  }
}}'
```

Check out jq, yaml2json, json2yaml, or glom for terse json processing on the command-line. Try this:

```
curl -XPOST https://api.catalysis-hub.org/graphql --data 'query={systems(last: 10 ) {
  edges {
    node {
      energy Cifdata
          }
```

(continues on next page)

(continued from previous page)

```
  }
}}' | jq '.data.systems.edges[].node.Cifdata' | sed -e 's/"//g' | split - -l 1␣
↪structure_ -d

sed -i 's/\\n/\n/g' structure_*
```

to write structures into many files. Or try this:

```
curl -XPOST https://api.catalysis-hub.org/graphql --data 'query={
  reactions( reactants:"CO") {
    totalCount
    edges {
      node {
        chemicalComposition
        reactionEnergy
        sites
      }
    }
  }
}
' | jq -r '.data.reactions.edges[].node | [.reactionEnergy,.chemicalComposition, .
↪sites ] | @csv'
```

for creating a CSV output.

3.3.2 Calling the Backend from a Python Script

Write a short python script with a GraphQL query of your choice. The script should look something like this:

```python
import requests
import pprint

root = 'http://api.catalysis-hub.org/graphql'

query = \
"""
{}
"""

data = requests.post(root, {'query': query}).json()
pprint.pprint(data)
```

And see the result printed in the terminal. How would you like to save the data?

3.3.3 Calling the Backend from a Perl Script

Write a short Perl script with a GraphQL query of your choice. This result could look something like this

```perl
#!/usr/bin/env perl

require LWP::UserAgent;

my $uri = 'http://api.catalysis-hub.org/graphql';
my $json = <<'EOF';
```

(continues on next page)

(continued from previous page)

```
{"query": "{
  reactions(first: 10) {
    edges {
      node {
        Equation
        chemicalComposition
        reactionEnergy
      }
    }
  }
} "}
EOF

# remove newlines
$json =~ s/(\n|\r)//g;

my $req = HTTP::Request->new( 'POST', $uri );
$req->header( 'Content-Type' => 'application/json' );
$req->content( $json );

my $lwp = LWP::UserAgent->new;
my $res = $lwp->request( $req );

print $res->content . "\n";
```

3.3.4 Calling the Backend from JavaScript

```
#!/usr/bin/env node

var axios = require('axios');

axios.post('http://api.catalysis-hub.org/graphql',
  {query:`
  {
  reactions(first: 10) {
    edges {
      node {
        Equation
        chemicalComposition
        reactionEnergy
      }
    }
  }
}
    `}
).then(function(response){
  console.log(JSON.stringify(response.data))
})
```

3.3.5 Calling the Backend from Coffee Script

```
#!/usr/bin/env coffee
```

(continues on next page)

(continued from previous page)

```
axios = require 'axios'

axios.post 'http://api.catalysis-hub.org/graphql', {query:"{
  reactions(first: 10) {
    edges {
      node {
        Equation
        chemicalComposition
        reactionEnergy
      }
    }
  }
}
    "}
.then (response) ->
  console.log JSON.stringify response.data
```

3.3.6 Connecting to the database server with psql

This exercise requires that you have postgreSQL installed, so you can use the *psql* terminal client. Also you need the password for the *catvisitor* user, or optionally your own user account. Contact Kirsten Winther at winther@stanford.edu for question.

Type into the terminal:

```
psql --host=catalysishub.c8gwuc8jwb7l.us-west-2.rds.amazonaws.com
--port=5432 --username=catvisitor --dbname=catalysishub
```

And write the password when prompted.

Now you can start writing SQL statements directly against the database server. Try for example:

```
SELECT title, year from publication LIMIT 10;
```

and see the output. Please use the `LIMIT` clause to limit the number of results, or specify `id=int`. See https://www.postgresql.org/docs/9.6/static/index.html for documentation on the SQL language and postgres.

3.4 Connecting to the database with ASE db

For this exercise you need to have a recent version of ASE installed. See https://wiki.fysik.dtu.dk/ase/install.html .

1) Now use the ASE cli to connect. Type this in the terminal (with an updated DB_PASSWORD):

   ```
   ase db postgresql://catvisitor:$DB_PASSWORD@catalysishub.
   c8gwuc8jwb7l.us-west-2.rds.amazonaws.com:5432/catalysishub Pt3Co
   ```

(Note: this query is probably going to take some time. We're still working on optimizing the ASE database part.)

2) Write a python script to connect via ase.db.connect. Hint: the connect() function will take the same server URL as used in the previous exercise.

   You can now use the select() function to make queries against the database. See https://wiki.fysik.dtu.dk/ase/ase/db/db.html for documentation.

3.5 Using the CatHub cli

This CLI will be used for collecting data from remote sources.

Build Status

3.5.1 Installation

Install with pip using

```
pip install catkit
```

3.5.2 Usage

Run `cathub`, like so

```
cathub --help
```

or with any of its sub-commands, like so

```
cathub make_folders_template --help
```

3.5.3 Examples

To create an .json input file for a folder structure

```
cathub make_folders_template project1.json --create-template
```

To create a folder structure from a .json input file

```
cathub make_folders_template project1.json
```

Reading folders into sqlite3 db file:

```
cathub folder2db <foldername>
```

Sending the data to the Catalysis Hub server:

```
cathub db2server <dbfile>
```

Querying the Catalysis Hub database:

```
cathub reactions -q reactants=CO -q chemicalComposition=~Pt

cathub publications -q title=~Evolution -q year=2017
```

3.6 IPython Notebook tutorials for API usage

The following notebooks demonstrate how make interactive use of the Catalysis-Hub.Org API. The recommend of using these Notebook is Jupyter Lab.

3.6.1 GraphQL Querying

3.6.2 Retrieve ASE Atoms object through GraphQL

3.6.3 Prototype Search

3.6.4 Search Structures and Cut Slabs

3.6.5 Get Spacegroup and Wyckoff Sites from Structure

3.7 Partner Projects

The catalysis-hub.org database is large and its possibilities are manifold. Fortunately there are two partner projects that are best digested together: CatKit and CatLearn.

CHAPTER 4

Reference

Complete reference of each and every function.

4.1 App

API for GraphQL enhanced queries against catapp and ase-db database

Some Examples:

- Get total number of rows in table (in this case reactions):

```
{reactions (first: 0) {
  totalCount
   edges {
     node {
       id
    }
  }
}}
```

- Filter by reactants and products from reactions:

```
{reactions(reactants: "H2O", products: "OH") {
  edges {
    node {
      reactants
      products
      reactionEnergy
      activationEnergy
    }
  }
}}
```

- Filter by several reactants or products from reactions:

```
{reactions(products: "Nstar+CH3star") {
  edges {
    node {
      reactants
      products
      reactionEnergy
      activationEnergy
    }
  }
}}
```

- Author-name from publications:

```
{publications(authors: "~Bajdich") {
  edges {
    node {
      reactions {
        chemicalComposition
        reactants
        products
        reactionEnergy
       }
     }
    }
  }}
```

- Full text search in reactions (reactants, products, chemical composition, facet):

```
{reactions(textsearch: "CO CH 111") {
  edges {
    node {
      reactants
      products
      publication {
        title
        authors
      }
    }
  }
}}
```

- Full text search in publications (title, authors, year):

```
{publications(pubtextsearch: "oxygen evolution bajdich 2017") {
  edges {
    node {
      title
      authors
      year
      reactions {
        reactants
        products
     }
    }
  }
}}
```

- Distinct reactants and products from reactions (works with and without “~”):

```
{reactions(reactants: "~OH", products: "~", distinct: true) {
  edges {
    node {
      reactants
      products
      reactionEnergy
    }
  }
}}
```

- ASE structures belonging to reactions:

```
{reactions(reactants: "~OH", first:1) {
   edges {
     node {
       systems {
         Cifdata
       }
     }
   }
 }}
```

- Get all distinct DOIs:

```
{publications {
   edges {
     node {
       doi
     }
   }
 }}
```

- Get all entries published since (and including) 2015:

```
 {publications(year: 2015, op: "ge", last:1) {
   edges {
     node {
      id
      year
      systems {
         keyValuePairs
      }
    }
  }
}}
```

class api.**CountableConnection**(**args*, ***kwargs*)
Bases: graphene.relay.connection.Connection

static resolve_total_count(*root*, *info*)

total_count = <graphene.types.scalars.Int object>

class api.**CustomSQLAlchemyObjectType**(**args*, ***kwargs*)
Bases: graphene_sqlalchemy.types.SQLAlchemyObjectType

class api.**FilteringConnectionField**(*type*, **args*, ***kwargs*)
Bases: graphene_sqlalchemy.fields.SQLAlchemyConnectionField

RELAY_ARGS = ['first', 'last', 'before', 'after']

SPECIAL_ARGS = ['distinct', 'op', 'jsonkey', 'order']

classmethod get_query(*model*, *info*, ***args*)

class api.**Information**(**args*, ***kwargs*)
Bases: *api.CustomSQLAlchemyObjectType*

class api.**Key**(**args*, ***kwargs*)
Bases: *api.CustomSQLAlchemyObjectType*

class api.**Log**(**args*, ***kwargs*)
Bases: *api.CustomSQLAlchemyObjectType*

class api.**NumberKeyValue**(**args*, ***kwargs*)
Bases: *api.CustomSQLAlchemyObjectType*

class api.**Publication**(**args*, ***kwargs*)
Bases: *api.CustomSQLAlchemyObjectType*

reactions = <graphene.types.structures.List object>

systems = <graphene.types.structures.List object>

class api.**Query**(**args*, ***kwargs*)
Bases: graphene.types.objecttype.ObjectType

information = <api.FilteringConnectionField object>

key = <api.FilteringConnectionField object>

logs = <api.FilteringConnectionField object>

node = <graphene.relay.node.NodeField object>

number_keys = <api.FilteringConnectionField object>

publications = <api.FilteringConnectionField object>

reaction_systems = <api.FilteringConnectionField object>

reactions = <api.FilteringConnectionField object>

species = <api.FilteringConnectionField object>

systems = <api.FilteringConnectionField object>

text_keys = <api.FilteringConnectionField object>

class api.**Reaction**(**args*, ***kwargs*)
Bases: *api.CustomSQLAlchemyObjectType*

reaction_systems = <graphene.types.structures.List object>

systems = <graphene.types.structures.List object>

class api.**ReactionSystem**(**args*, ***kwargs*)
Bases: *api.CustomSQLAlchemyObjectType*

class api.**Species**(**args*, ***kwargs*)
Bases: *api.CustomSQLAlchemyObjectType*

class api.**System**(**args*, ***kwargs*)
Bases: *api.CustomSQLAlchemyObjectType*

log = <graphene.types.structures.List object>

publication = <graphene.types.structures.List object>

static resolve__input_file(*self*, *info*, *format='py'*)

Return the structure as input for one of several DFT codes as supported by ASE. Default format is "py". Run:

```
{systems(last: 1) {
  totalCount
  edges {
    node {
      InputFile(format:"")
    }
  }
}}
```

to show available formats. Try one of the available formats like,:

```
{systems(last: 10) {
  totalCount
  edges {
    node {
      InputFile(format:"espresso-in")
    }
  }
}}
```

to generate QE input.

class api.**TextKeyValue**(**args*, ***kwargs*)

Bases: *api.CustomSQLAlchemyObjectType*

api.**get_filter_fields**(*model*)

Generate filter fields (= comparison) from graphene_sqlalcheme model

#.. automodule:: models #:members: #:undoc-members: #:show-inheritance:

4.2 Apps

HTTP Request to the Apps backend can be made in Python using e.g. the *requests* library for GET requests, like so:

```
#!/usr/bin/env python

import json
import pprint

import requests

url = 'http://api.catalysis-hub.org/apps/prototypeSearch/facet_search/'
r = requests.get(url,
                  params={
                      'search_terms': [
                          'hollandite',
                      ],
                      'facet_filters': '["spacegroup:87"]'
                  }
                  )
pprint.pprint(json.loads(r.content))
```

or for POST requests, like so:

```
#!/usr/bin/env python

import json
import pprint

import requests

url = 'http://api.catalysis-hub.org/apps/prototypeSearch/get_structure/'

r = requests.post(url,
    json={
        'parameters': '[3.1]',
        'species': '["S"]',
        'spacegroup': 221,
        })
pprint.pprint(json.loads(r.content))
```

That is, every function below that is declared either as a GET or a POST request can be translated into a corresponding HTTP request by replacing every dot ('.') with a slash ('/') and passing in arguments either as params=… (GET) or json=… (POST) .

Please refer to the following API documentation for details.

4.2.1 ActivityMaps

class `apps.activityMaps.`**`ReactionModel`**(*xlabel=None, ylabel=None, zlabel=None, reference="*)

Bases: `object`

`get_raw_systems`(*filters*)

`get_xyz`(*systems*)

`apps.activityMaps.`**`graphql_query`**(*products='products: "O"', reactants=", facet=", limit=5000*)

`apps.activityMaps.`**`systems`**(*request=None*)

GET: Get systems for given reactions

Parameters **`activityMap`** (*`str`*) – request Map like OER, NRR, HER. Defaults to CO_Hydrogenation_111.

Returns

The corresponding systems in the database.

- reference(str): Reference for activity map.
- systems(list): Corresponding systems.

Return type dict

Examples

```
curl {ROOT}/apps/activityMaps/systems/?activityMap=OER
curl {ROOT}/apps/activityMaps/systems/?activityMap=CO_Hydrogenation_111
```

```
  {
"reference": "[1] Friebel, Daniel, Mary W. Louie, Michal Bajdich, Kai E. Sanwald,␣
↪Yun Cai, Anna M. Wise, Mu-Jeng Cheng et al. "Identification of highly active Fe␣
↪sites in (Ni, Fe) OOH for electrocatalytic water splitting." Journal of the␣
↪American Chemical Society 137, no. 3 (2015): 1305-1313. DOI: 10.1021/ja511559d␣
↪[2] Man, Isabela C., Hai\u2010Yan Su, Federico Calle\u2010Vallejo, Heine A.␣
↪Hansen, Jos\u00e9 I. Mart\u00ednez, Nilay G. Inoglu, John Kitchin, Thomas F.␣
↪Jaramillo, Jens K. N\u00f8rskov, and Jan Rossmeisl. "Universality in oxygen␣
↪evolution electrocatalysis on oxide surfaces." ChemCatChem 3, no. 7 (2011):␣
↪1159-1165. DOI: 10.1002/cctc.201000397",
"systems": [
  {
    "facet": "3ML",
    "formula": "Ir16Sr4O51",
    "uid": "5b0b436e4d3d07c3fb7a4cee6d5975f1",
    "x": 1.5028540934200003,
    "y": 1.3901226701799998,
    "z": -0.3208143060400004
  },
  {
    "facet": "100",
    "formula": "Ir24O53",
    "uid": "b33747e9868b9514639752f1b58e2f03",
    "x": 1.4204331210799999,
    "y": 0.44616836241,
    "z": -0.4164322559
  }, ] }
```

4.2.2 AtoMl

4.2.3 Bulk Enumerator

apps.bulkEnumerator.**get_bulk_enumerations**(*request=None*)

Return a list of prototypes names

Parameters

- **stoichiometry** (*str, optional*) – Stoichiometry separated by '_'. Defaults to '1'.
- **num_type** (*str, optional*) – Limit by number of 'atoms' or number of 'wyckoff' sites. Possible values are 'atoms' or 'wyckoff'. Defaults to 'atoms'.
- **num_start** (*int, optional*) – Mininum number of sites. Defaults to 1.
- **num_end** (*int, optional*) – Maximum number of sites. Defaults to 1.
- **SG_start** (*int, optional*) – Lowest spacegroup to consider. Can be between 1 and 230. Defaults to 1.
- **SG_end** (*int, optional*) – Hightest spacegroup to consider. Can be between 1 and 230. Defaults to 10.

Returns

Dictionary of input and corresponding enumerations.

dict {input }: Input parameters. list [enumerations]: List of possible enumeations. dict {enumerations}: {

name: (str) Prototype Name, natom: (int) Number of Atoms, parameters: [str] Free parameters, spaceGroupNumber: (int) The number of the spacegroup (1-230), specie_permutations: [str] Equivalent permutations, species: [str] The possible occupations, wyckoffs [str] The Wyckoff sites,

}

Return type dict {input, enumerations }

`apps.bulkEnumerator.`**`get_structure`**(*request=None*)

Construct structure from wyckoff positions, species, and other parameters

Parameters

- **`wyckoffPositions`** (*[str]*) – List of Wyckoff positions (length one strings).
- **`wyckoffSpecies`** (*[str]*) – Corresponding list of elements.

`apps.bulkEnumerator.`**`get_wyckoff_from_cif`**(*request=None*)

Function clone of get_wyckoff_from_structure, except working w/ string input instead of file upload.

`apps.bulkEnumerator.`**`get_wyckoff_from_structure`**(*request=None*)

`apps.bulkEnumerator.`**`get_wyckoff_list`**(*request=None*)

Return a list of possible wyckoff position belonging to a certain spacegroup.

Parameters

- **`spacegroup`** (*int, optional*) –
- **`tolerance`** (*float, optional*) –

`apps.bulkEnumerator.`**`mstripb`**(*liste*)

`apps.bulkEnumerator.`**`stripb`**(*string*)

4.2.4 CatKitDemo

4.2.5 Pourbaix diagrams

4.2.6 Prototype Search

4.2.7 Utilities

`apps.utils.`**`ase_convert`**(*instring, informat=None, outformat=None, atoms_in=False, atoms_out=False*)

Enter a input file that is understood by ASE and return a string in a different format as written by ase.

4.3 Schema Types

- *Query*
- *Objects*
  - *Information*
  - *InformationCountableConnection*
  - *InformationCountableEdge*

4.3.1 Query

The ID of the object

4.3.2 Objects

Information

The ID of the object.

InformationCountableConnection

InformationCountableEdge

The item at the end of the edge

A cursor for use in pagination

Key

The ID of the object.

KeyCountableConnection

KeyCountableEdge

The item at the end of the edge

A cursor for use in pagination

NumberKeyValue

The ID of the object.

NumberKeyValueCountableConnection

NumberKeyValueCountableEdge

The item at the end of the edge

A cursor for use in pagination

PageInfo

When paginating forwards, are there more items?

When paginating backwards, are there more items?

When paginating backwards, the cursor to continue.

When paginating forwards, the cursor to continue.

Publication

The ID of the object.

PublicationCountableConnection

PublicationCountableEdge

The item at the end of the edge

A cursor for use in pagination

Reaction

The ID of the object.

ReactionCountableConnection

ReactionCountableEdge

The item at the end of the edge

A cursor for use in pagination

ReactionSystem

The ID of the object.

ReactionSystemCountableConnection

ReactionSystemCountableEdge

The item at the end of the edge

A cursor for use in pagination

Species

The ID of the object.

SpeciesCountableConnection

SpeciesCountableEdge

The item at the end of the edge

A cursor for use in pagination

System

The ID of the object.

SystemCountableConnection

SystemCountableEdge

The item at the end of the edge

A cursor for use in pagination

TextKeyValue

The ID of the object.

TextKeyValueCountableConnection

TextKeyValueCountableEdge

The item at the end of the edge

A cursor for use in pagination

4.3.3 Scalars

Boolean

The `Boolean` scalar type represents `true` or `false`.

Float

The `Float` scalar type represents signed double-precision fractional values as specified by IEEE 754.

ID

The `ID` scalar type represents a unique identifier, often used to refetch an object or as key for a cache. The ID type appears in a JSON response as a String; however, it is not intended to be human-readable. When expected as an input type, any string (such as `"4"`) or integer (such as `4`) input value will be accepted as an ID.

Int

The `Int` scalar type represents non-fractional signed whole numeric values. Int can represent values between -(2^53 - 1) and 2^53 - 1 since represented in JSON as double-precision floating point numbers specifiedby IEEE 754.

JSONString

JSON String

String

The `String` scalar type represents textual data, represented as UTF-8 character sequences. The String type is most often used by GraphQL to represent free-form human-readable text.

4.3.4 Interfaces

Node

An object with an ID

The ID of the object.

CHAPTER 5

Troubleshooting

Assuming nothing helps, how to debug, get more help, file bug reports, pull-request etc..

CHAPTER 6

Indices and tables

- genindex
- modindex
- search

Python Module Index

Index