BASIC EXAMPLES

1  First Steps  
2  Reaction Datasets  
3  TorsionDrive Datasets  

3  
7  
11  
These examples demonstrate how to use the QCVerse ecosystem in a variety of situations. To run these examples locally, please install qcportal with either conda or pip.

**Installing qcportal using conda:**

```
conda install qcportal -c conda-forge
```

or with pip:

```
pip install qcportal
```

You can run this notebook online in a Binder session or view it on Github.
The Molecular Sciences Software Institute hosts the Quantum Chemistry Archive (QCArchive) and makes this data available to the entire Computational Molecular Sciences community free of charge. The QCArchive is both a database to view, analyze, and explore existing data as well as a live instance that continuously generates new data as directed by the community.

The primary interface to this database in Python is a through a FractalClient from the qcportal package which can be downloaded via pip (pip install -e qcportal) or conda (conda install qcportal -c conda-forge). A new FractalClient automatically connects to MolSSI's central server and has access to all data contained within the QCArchive.

```python
import qcfractal.interface as ptl
client = ptl.FractalClient()
```

### 1.1 Finding Collections

One of the main ways to explore the QCArchive is to examine Collections which are structures that allow easy manipulation of data in preset ways. Several example of Collections contained within the QCArchive are as follows:

- **Dataset** - A dataset where each record is a single molecule and many methods can be applied to each record.
- **ReactionDataset** - A dataset where each record is a combination of molecule (e.g. interaction and reaction energies) and many methods can be applied to each record.
- **OpenFFWorkflow** - A workflow collection for torsiondrives and constrained optimization developed with the Open Force Field Initiative.
- **TorsionDriveDataset** - A dataset which organizes many molecular torsion scans together for data exploration, analysis, and methodology comparison (see the TorsionDrive Dataset example for more details).

```python
client.list_collections().head()
```

(continues on next page)
1.2 Exploring Collections

Collections can be obtained by pulling their data from the central server. A collection is primarily metadata and extremely large collections can be pulled in a few seconds. For this example we will explore S22 dataset which is a small interaction energy dataset of 22 common dimers such as the water dimer, methane dimer, and more. To obtain this collection:

```python
[4]: ds = client.get_collection("ReactionDataset", "S22")
print(ds)
ReactionDataset (name='S22', id='5c8159a4b6a2de3bd1e74306', client='https://api.qcarchive.molssi.org:443/')
```

1.3 Statistics and Visualization

Visual statics and plotting can be generated by the `visualize` command:

```python
[5]: ds.visualize(method="B2PLYP", basis=['def2-svp', 'def2-tzvp'], bench="S220", kind="violin")
```

Data type cannot be displayed: application/vnd.plotly.v1+json, text/html

1.4 Next steps

Congratulations! You have taken the first steps to exploring the data within the QCArchive. Please consider viewing the Reaction Dataset and the TorsionDrive Dataset examples to do a more in depth look at these Collections and what you can do with them.
Feel free to explore the data you access through these examples in detail. When you connect a FractalClient to the server without a username and password, the data is open to explore and cannot alter what is saved on the server itself. So if you change your local data, the server data remains untouched!

You can run this notebook online in a Binder session or view it on Github.
ReactionDatasets are datasets where the primary index is made up of linear combinations of individual computations. For example, an interaction energy dataset would have an index of the complex subtracted by the individual monomers to obtain a final interaction energy. This idea can extended to standard reaction energies, conformational defect energies, and more. This dataset type has been developed by the QCArchive Team in conjunction with:

- David Sherrill
- Lori Burns
- Daniel Nascimento
- Dom Sirianni

To begin, we can connect to the MolSSI QCArchive server:

```python
import qcportal as ptl
client = ptl.FractalClient()
print(client)

FractalClient(server_name='The MolSSI QCArchive Server', address='https://api.qcarchive.molssi.org:443/', username='None')
```

The current ReactionDatasets can be explored below:

```python
client.list_collections("ReactionDataset").head()
```

<table>
<thead>
<tr>
<th>collection</th>
<th>name</th>
<th>tagline</th>
</tr>
</thead>
<tbody>
<tr>
<td>ReactionDataset</td>
<td>A21</td>
<td>Equilibrium complexes from A24 database of sma...</td>
</tr>
<tr>
<td></td>
<td>A24</td>
<td>Interaction energies for small bimolecular com...</td>
</tr>
<tr>
<td></td>
<td>ACONF</td>
<td>Conformation energies for alkanes</td>
</tr>
<tr>
<td></td>
<td>AlkBnd12</td>
<td>Binding energies of saturated and unsaturated ...</td>
</tr>
<tr>
<td></td>
<td>AlkIsod14</td>
<td>Isodesmic reaction energies for alkanes N=3--8</td>
</tr>
</tbody>
</table>

### 2.1 Exploring a Dataset

For this example, we will explore S22 dataset which is a small interaction energy dataset of 22 common dimers such as the water dimer, methane dimer, and more. To obtain this collection:

```python
ds = client.get_collection("ReactionDataset", "S22")
print(ds)

ReactionDataset (name='S22', id='5c8159a4b6a2de3bd1e74306', client='https://api.qcarchive.molssi.org:443/)
```
This dataset automatically comes with some Contributed Value data, or data that has been provided rather than explicitly computed. This data is often either experimental data or very costly benchmarks taken from literature. As these Datasets are based off of Pandas DataFrames, we can directly access the underlying DataFrame to see the data provided:

```
[4]: ds.df.head()
   S220  S22a  S22b
Ammonia Dimer   -3.17  -3.15  -3.133
Water Dimer     -5.02   -5.07   -4.989
Formic Acid Dimer -18.61 -18.81 -18.753
Formamide Dimer  -15.96 -16.11 -16.062
Uracil Dimer HB  -20.65 -20.69 -20.641
```

Here we used .head() to access the first five records in the ReactionDataset.

All Collections that have Dataset in the name (including ReactionDataset) have a history available to them to list the data that has been computed. In this case we will filter our history by the DFT method B2PLYP and the basis set def2-SVP

```
[5]: ds.list_history(method="B2PLYP", basis="def2-SVP")

<table>
<thead>
<tr>
<th>driver</th>
<th>program</th>
<th>method</th>
<th>basis</th>
<th>keywords</th>
</tr>
</thead>
<tbody>
<tr>
<td>stoichiometry</td>
<td>psi4</td>
<td>b2plyp</td>
<td>def2-svp</td>
<td>scf_default</td>
</tr>
<tr>
<td>energy</td>
<td>scf_default</td>
<td>cp</td>
<td>default</td>
<td></td>
</tr>
</tbody>
</table>
```

Here we can see that there are five primary keys in the computation:

- **driver** - The type of computation, this can be energy, gradient, Hessian, and properties.
- **program** - The program used in the computation.
- **method** - The quantum chemistry, semiempirical, AI-model, or force field used in the computation.
- **basis** - The basis used in the computation.
- **keywords** - A keywords alias used in the computation – these keywords aliases reference KeywordSets (see advanced tutorials).

In addition, there is also the stoichiometry field which is unique to ReactionDatasets. There exist several ways to compute the interaction energy (counterpoise-corrected (cp), non-counterpoise-corrected (default), and Valiron–Mayer function counterpoise (vmfc)). The stoichiometry field allows for the selection of this particular form.

### 2.2 Querying Data

To obtain the data for the various historical computations we must query them from the server. Here we will automatically pull all relevant computations that match our query:

```
[6]: ds.get_history(method="B3LYP-D3M")
   ds.df.head()
   S220  S22a  S22b  B3LYP-D3M/def2-svp
Ammonia Dimer   -3.17  -3.15  -3.133  -6.248386
Water Dimer     -5.02   -5.07   -4.989  -9.002674
Formic Acid Dimer -18.61 -18.81 -18.753  -25.933297
Formamide Dimer  -15.96 -16.11 -16.062  -21.689185
```

(continues on next page)
2.3 Statistics and Visualization

Visual statistics and plotting can be generated by the `visualize` command:

```python
[7]: ds.visualize(method=["B3LYP", "B3LYP-D3", "B3LYP-D3M"], basis=["def2-tzvp"], groupby="D3")
```

Data type cannot be displayed: text/html, text/vnd.plotly.v1+html

```python
[8]: ds.visualize(method=["B3LYP", "B3LYP-D3", "B2PLYP", "B2PLYP-D3"], basis="def2-tzvp", groupby="D3", kind="violin")
```

Data type cannot be displayed: text/html, text/vnd.plotly.v1+html

You can run this notebook online in a Binder session or view it on Github.
CHAPTER THREE

TORSIONDRIVE DATASETS

An individual TorsionDrive is a specific workflow in the QCArchive ecosystem that computes the energy profile of rotatable dihedral (torsion) for use in Force Field Fitting. This workflow has been developed by the QCArchive Team in conjunction with:

- Lee-Ping Wang
- John Chodera
- Chaya Stern
- Yudong Qiu
- The Open Force Field Initiative

The top-level TorsionDrive code can be found here.

The TorsionDrive Dataset organizes many TorsionDrives together for data exploration, analysis, and methodology comparison. To begin, we can connect to the MolSSI QCArchive server and query all known TorsionDrive Datasets:

```
[1]: import qcportal as ptl
client = ptl.FractalClient()
client
[1]: FractalClient(server_name='The MolSSI QCArchive Server', address='https://api.qcarchive.molssi.org:443/', username='None')

[2]: client.list_collections("TorsionDriveDataset")
```

```
tagline          collection           name                                      
TorsionDriveDataset OpenFF Fragmenter Phenyl Benchmark Phenyl substituent torsional barrier heights.  
                     OpenFF Group1 Torsions                                           
                     None                                                      
```

One of the datasets can be obtained in the canonical manner:

```
[3]: ds = client.get_collection("TorsionDriveDataset", "OpenFF Fragmenter Phenyl Benchmark barrier heights.
```

### 3.1 Exploring the Dataset

Each row of the dataset is comprised of an Entry which contains a list of starting molecules for the TorsionDrive, the dihedral of interest (in this case zero-indexed), and the scan resolution of the dihedral. For the purposes of the
underlying DataFrame each row is the name of this Entry.

```python
[4]: ds.df.head()
Empty DataFrame
Columns: []
Index: [c1c[cH:1][c:2](cc1)[C:3](=[O:4])O, c1[cH:1][c:2](cncl)[C:3](=[O:4])O, ...
    Cc1c[cH:1][c:2](cn1)[C:3](=[O:4])O]
```

New computations are based off specifications which contain many additional parameters to tune the torsiondrive as well as the underlying computational method. Here, we can list all specifications that are attributed to this dataset.

```python
[5]: ds.list_specifications()

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>UFF</td>
<td>UFF gradient evaluation with RDKit</td>
</tr>
<tr>
<td>B3LYP-D3</td>
<td>B3LYP-D3 evaluation with Psi4</td>
</tr>
</tbody>
</table>
```

As we can see here we have several specifications at different levels of theory. It is important to recall that these Collections are “live”. New specifications can be added and individual TorsionDrives can be under computation. To see the current status of each specification the `status` function is provided:

```python
[6]: ds.status(["uff", "b3lyp-d3"])

<table>
<thead>
<tr>
<th>Specification</th>
<th>Status</th>
<th>Count</th>
</tr>
</thead>
<tbody>
<tr>
<td>UFF</td>
<td>COMPLETE</td>
<td>165</td>
</tr>
<tr>
<td>B3LYP-D3</td>
<td>COMPLETE</td>
<td>226</td>
</tr>
<tr>
<td>INCOMPLETE</td>
<td>62</td>
<td>1</td>
</tr>
</tbody>
</table>
```

```python
[7]: ds.status(["b3lyp-d3"], collapse=False, status="COMPLETE").head()

<table>
<thead>
<tr>
<th>Name</th>
<th>B3LYP-D3</th>
</tr>
</thead>
<tbody>
<tr>
<td>clc[cH:1]<a href="cc1">c:2</a><a href="">C:3</a>O</td>
<td>COMPLETE</td>
</tr>
<tr>
<td>c1[cH:1]<a href="cncl">c:2</a><a href="">C:3</a>O</td>
<td>COMPLETE</td>
</tr>
<tr>
<td>[cH:1]ncnc[c:2]1<a href="">C:3</a>O</td>
<td>COMPLETE</td>
</tr>
<tr>
<td>[cH:1]nccc[c:2]1<a href="">C:3</a>O</td>
<td>COMPLETE</td>
</tr>
<tr>
<td>Cc1c[cH:1]<a href="cn1">c:2</a><a href="">C:3</a>O</td>
<td>COMPLETE</td>
</tr>
</tbody>
</table>
```

### 3.2 Visualizing the TorsionDrives

TorsionDrives can be visualized via the `visualize` command. Multiple torsiondrives can be plotted on the same graph for comparison.

```python
[8]: ds.visualize(["[CH3:4][O:3][c:2][c:1]cccc1", "[CH3:4][O:3][c:2][c:1]ccnc1"], "B3LYP-D3", units="kJ / mol")
```

Data type cannot be displayed: application/vnd.plotly.v1+json, text/html
3.3 Computational Requirements

We can check the number of optimizations and individual gradient evaluations by using the `count` method. By default the `count` method shows the number of individual geometry optimizations:

```python

[CH3:4][O:3][c:2]1[ch:1]ccnc1 49 53
```

In addition, to individual optimization runs we can also find a sum of how many gradient evaluations were performed in the course of the run.

```python

CH3:4][O:3][c:2]1[ch:1]cccc1 606 601
[CH3:4][O:3][c:2]1[ch:1]ccnc1 538 680
```

3.4 Deep data inspection

The TorsionDriveDataset also allows exploration of every single computation and molecule created during the individual TorsionDrive executions. Examples below this point will only be for those who wish to explore all of the individual computations in a TorsionDrive Dataset.

These TorsionDrive Datasets are different from canonical ReactionDataset and Dataset collections as each item in the underlying Pandas DataFrame is a TorsionDriveRecord object which contains links to all data used in the TorsionDrive. We can observe the these in the underlying DataFrame:

```python
[11]: ds.df.head()

              UFF     B3LYP-D3
               \   
clc[ch:1][c:2](cc1)[C:3]([O:4])O TorsionDriveRecord(id='5c951108cc2095305535df47.
   \.. c1[ch:1][c:2](cnc1)[C:3]([O:4])O TorsionDriveRecord(id='5c951108cc2095305535df48.
   \..  [ch:1]1cncc[c:2]1[C:3]([O:4])O TorsionDriveRecord(id='5c951108cc2095305535df49.
   \..  [ch:1]1cc(nc[c:2]1[C:3]([O:4])O)O[-] TorsionDriveRecord(id='5c951108cc2095305535df50.
   \.. C1lc[ch:1][c:2](cn1)[C:3]([O:4])O TorsionDriveRecord(id='5c951108cc2095305535df51.
   \..                      B3LYP-
   \.. clc[ch:1][c:2](cc1)[C:3]([O:4])O TorsionDriveRecord(id='5c955144cc2095305555df9af.
   \.. c1[ch:1][c:2](cnc1)[C:3]([O:4])O TorsionDriveRecord(id='5c955144cc2095305555d9b1.
   \..  [ch:1]1cncc[c:2]1[C:3]([O:4])O TorsionDriveRecord(id='5c955144cc2095305555d9b2.
   \..  [ch:1]1cc(nc[c:2]1[C:3]([O:4])O)O[-] TorsionDriveRecord(id='5c955144cc2095305555d9b3.
```

(continues on next page)
We can then begin to explore an individual TorsionDriveRecord by first pulling it from the DataFrame:

```python
[12]: td = ds.df.loc["[CH3:4][O:3][c:2]1[cH:1]cccc1", "B3LYP-D3"]
```

We can then request a variety of attributes off this TorsionDriveRecord:

```python
[13]: print("Torsion of interest : \{}".format(td.keywords.dihedrals))
print("Final optimization energy in hartree: \{}".format(td.get_final_energies(180)))
```

```
Torsion of interest : \[(3, 5, 7, 6)\]
Final optimization energy in hartree: -346.5319986074462
```

```python
[14]: td.get_final_molecules(90)
```

```
Data type cannot be displayed: application/3dmoljs_load.v0, text/html
```

```python
[14]: <Molecule(name='C7H8O' formula='C7H8O' hash='a6601a8')>
```

The Molecule object has a measure attribute so that we check the dihedral angle is in fact 180 degrees.

```python
[15]: "3-5-7-6 dihedral in degrees: \{}".format(td.get_final_molecules(180).measure([3, 5, 7, 6]))
```

```
3-5-7-6 dihedral in degrees: 179.99999995886662
```

### 3.5 Exploring connection optimizations

If desired, we can pull each geometry optimization belonging to the torsiondrive with the `get_history` function. In this case, we will pull the lowest energy optimization for the 180 degree dihedral:

```python
[16]: opt = td.get_history(180, minimum=True)
opt
```

```
<OptimizationRecord(id='5c988a95cc20953055316237' status='COMPLETE')>
```

```python
[17]: opt.energies
```

```
```
3.6 Exploring individual gradient evaluations

We can go even deeper in the calculations to look at each gradient calculation of the Optimization calculation if we so choose and see even more details about how the TorsionDrive object was constructed.

```python
[18]: result = opt.get_trajectory()[−1]
[19]: print("Program: {}\nNumber of Basis Functions: {}\nTotal execution time: {:.2f}s").format(result.program, result.properties.calcinfo_nbasis, result.provenance.wall_time)
```

Program: psi4
Number of Basis Functions: 152
Total execution time: 12.15s

This example also contains the Wiberg-Lowdin indices. As this is specific to Psi4 this data resides inside the extras tag rather than general properties. This data is not yet well curated and currently exists as a 1D list. We will first pull this data and transform it to a 2-D array:

```python
[20]: import numpy as np
wiberg = np.array(result.extras["qcvars"]['WIBERG_LOWDIN_INDICES']).reshape(-1, 16)
```

As this particular example is exploring the 3-5-7-6 dihedral we would find the most use in the 5-7 bond. This can be acquired as follows:

```python
[21]: wiberg[5, 7]
[21]: 1.2700940280530457
```

We can continue to explore these results and even obtain the standard Psi4 logging information!

```python
[22]: print(result.get_stdout()[:1000])
```

Memory set to 60.800 GiB by Python driver.
gradient() will perform analytic gradient computation.

*** tstart() called on dt039
*** at Mon Mar 25 04:02:23 2019
=> Loading Basis Set <=
  Name: DEF2-SVP
  Role: ORBITAL
  Keyword: BASIS
  atoms 1-7 entry C line 90 file /home/lnaden/miniconda3/envs/qca/
  →share/psi4/basis/def2-svp.gbs
  atoms 8 entry O line 130 file /home/lnaden/miniconda3/envs/qca/
  →share/psi4/basis/def2-svp.gbs
  atoms 9-16 entry H line 15 file /home/lnaden/miniconda3/envs/qca/
  →share/psi4/basis/def2-svp.gbs

-------------------------------
SCF
by Justin Turney, Rob Parrish, Andy Simmonett
and Daniel G. A. Smith
RKS Reference

(continues on next page)
6 Threads, 62259 MiB Core
--------------------------------------

==> Geometry <==

Molecular