QCArchive Documentation

Release 1.0

Daniel G. A. Smith

Aug 20, 2019

BASIC EXAMPLES

These examples demonstrate how to use QCPortal to access data on the QCArchive in a variety of situations.

All examples are available in your browser through Binder. Alternatively, you may run these examples locally. To do so, please install QCPortal with either `conda` or `pip`.

CHAPTER

ONE

LOCAL INSTALLATION

Install `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.

1.1 First Steps

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.

[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')
```

1.1.1 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 corresponds to a single molecule, with one or more QM methods applied to that molecule.
- `ReactionDataset` - A dataset where each record is a combination of molecules (e.g. interaction and reaction energies). Each record contains data from one or more QM methods.

- `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).

[2]:
```
client.list_collections().head()
```

[2]:
```
                                                                                ␣
↪   tagline
collection          name
OpenFFWorkflow      chemper_rdkit                                               ␣
↪       None
OptimizationDataset OpenFF Optimization Set 1                                   ␣
↪       None
ReactionDataset     A21                         Equilibrium complexes from A24␣
↪database of sma...
                    A24                         Interaction energies for small␣
↪bimolecular com...
                    ACONF                                          Conformation energies␣
↪for alkanes
```

Specific `Collection` types can be queried to limit the amount of collections to browse through:

[3]:
```
client.list_collections("reactiondataset").head()
```

[3]:
```
                                                                tagline
collection      name
ReactionDataset A21        Equilibrium complexes from A24 database of sma...
                A24        Interaction energies for small bimolecular com...
                ACONF                      Conformation energies for alkanes
                AlkBind12  Binding energies of saturated and unsaturated ...
                AlkIsod14     Isodesmic reaction energies for alkanes N=3--8
```

1.1.2 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:

[4]:
```
ds = client.get_collection("ReactionDataset", "S22")
print(ds)
```

```
ReactionDataset(name=`S22`, id='5c8159a4b6a2de3bd1e74306', client='https://api.
↪qcarchive.molssi.org:443/')
```

1.1.3 Statistics and Visualization

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

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

(continued from previous page)

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

1.1.4 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 for 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.

1.2 Reaction Datasets

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:

[2]:
```
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:

[2]:
```
client.list_collections("ReactionDataset").head()
```

[2]:

		tagline
collection	name	
ReactionDataset	A21	Equilibrium complexes from A24 database of sma...
	A24	Interaction energies for small bimolecular com...
	ACONF	Conformation energies for alkanes
	AlkBind12	Binding energies of saturated and unsaturated ...
	AlkIsod14	Isodesmic reaction energies for alkanes N=3--8

1.2.1 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:

[3]:
```
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 through QCArchive. Such data often come from experiments or very costly benchmarks taken from literature.

Datasets are based off of Pandas DataFrames; we can directly access the underlying DataFrame to see the data provided:

[4]:
```
ds.df.head()
```

[4]:

	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")
```

[5]:

					stoichiometry
driver	program	method	basis	keywords	
energy	psi4	b2plyp	def2-svp	scf_default	cp
				scf_default	default

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, specific to the details of the program or procedure.

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.

1.2.2 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()
```

[6]:
```
                      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
Uracil Dimer HB   -20.65 -20.69 -20.641          -25.623412

                    B3LYP-D3M/def2-tzvp
Ammonia Dimer                 -4.049052
Water Dimer                   -6.427460
Formic Acid Dimer            -20.668411
Formamide Dimer              -17.436781
Uracil Dimer HB              -21.922461
```

1.2.3 Statistics and Visualization

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

[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

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

[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

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

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

1.3 TorsionDrive Datasets

An individual TorsionDrive is 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")
```

[2]:
```
                                                                                ␣
↪        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
↪")
```

1.3.1 Exploring the Dataset

Each row of the dataset is comprised of a `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`.

[4]:
```
ds.df.head()
```

[4]:
```
Empty DataFrame
Columns: []
Index: [c1c[cH:1][c:2](cc1)[C:3](=[O:4])O, c1[cH:1][c:2](cnc1)[C:3](=[O:4])O,␣
↪[cH:1]1cncc[c:2]1[C:3](=[O:4])O, [cH:1]1cc(nc[c:2]1[C:3](=[O:4])O)[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.

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

[5]:
```
                                Description
Name
UFF        UFF gradient evaluation with RDKit
B3LYP-D3       B3LYP-D3 evaluation with Psi4
```

As we can see, 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:

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

[6]:
```
             UFF  B3LYP-D3
COMPLETE     165       226
INCOMPLETE    62         1
```

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

[7]:
```
                                          B3LYP-D3
c1c[cH:1][c:2](cc1)[C:3](=[O:4])O         COMPLETE
c1[cH:1][c:2](cnc1)[C:3](=[O:4])O         COMPLETE
[cH:1]1cncc[c:2]1[C:3](=[O:4])O           COMPLETE
[cH:1]1cc(nc[c:2]1[C:3](=[O:4])O)[O-]     COMPLETE
Cc1c[cH:1][c:2](cn1)[C:3](=[O:4])O        COMPLETE
```

1.3.2 Visualizing the TorsionDrives

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

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

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

1.3.3 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:

[9]:
```
ds.counts(["[CH3:4][O:3][c:2]1[cH:1]cccc1", "[CH3:4][O:3][c:2]1[cH:1]ccnc1"])
```

[9]:
```
                                 UFF  B3LYP-D3
[CH3:4][O:3][c:2]1[cH:1]cccc1     49        49
[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.

[10]:
```
ds.counts(["[CH3:4][O:3][c:2]1[cH:1]cccc1", "[CH3:4][O:3][c:2]1[cH:1]ccnc1"], count_
↪gradients=True)
```

[10]:
```
                                 UFF  B3LYP-D3
[CH3:4][O:3][c:2]1[cH:1]cccc1    606       601
[CH3:4][O:3][c:2]1[cH:1]ccnc1    538       680
```

1.3.4 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:

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

[11]:
```
                                                                                                                    ␣
↪UFF  \
c1c[cH:1][c:2](cc1)[C:3](=[O:4])O      TorsionDriveRecord(id='5c951108cc2095305535f47.
↪..
c1[cH:1][c:2](cnc1)[C:3](=[O:4])O      TorsionDriveRecord(id='5c951108cc2095305535f48.
↪..
[cH:1]1cncc[c:2]1[C:3](=[O:4])O        TorsionDriveRecord(id='5c951108cc2095305535f48.
↪..
[cH:1]1cc(nc[c:2]1[C:3](=[O:4])O)[O-]  TorsionDriveRecord(id='5c951108cc2095305535f49.
↪..
Cc1c[cH:1][c:2](cn1)[C:3](=[O:4])O     TorsionDriveRecord(id='5c951108cc2095305535f4a.
↪..

                                                                                                                B3LYP-
↪D3
c1c[cH:1][c:2](cc1)[C:3](=[O:4])O      TorsionDriveRecord(id='5c955144cc209530555d9af.
↪..
c1[cH:1][c:2](cnc1)[C:3](=[O:4])O      TorsionDriveRecord(id='5c955144cc209530555d9b1.
↪..
[cH:1]1cncc[c:2]1[C:3](=[O:4])O        TorsionDriveRecord(id='5c955144cc209530555d9b2.
↪..
[cH:1]1cc(nc[c:2]1[C:3](=[O:4])O)[O-]  TorsionDriveRecord(id='5c955144cc209530555d9b3.
↪..
Cc1c[cH:1][c:2](cn1)[C:3](=[O:4])O     TorsionDriveRecord(id='5c955144cc209530555d9b5.
↪..
```

We can then begin to explore an individual TorsionDriveRecord by first pulling it from the DataFrame:

[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:

[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
```

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

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

[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.

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

[15]:
```
'3-5-7-6 dihedral in degrees: 179.99999995886662'
```

1.3.5 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:

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

[16]:
```
<OptimizationRecord(id='5c988a95cc20953055316237' status='COMPLETE')>
```

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

[17]:
```
[-346.53147622614085,
 -346.53177915374135,
 -346.5318873591642,
 -346.53194287295247,
 -346.53197743434515,
 -346.5319973422999,
 -346.53199831291954,
 -346.53199826976714,
 -346.5319986074462]
```

1.3.6 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.

[18]:
```
result = opt.get_trajectory()[-1]
```

[19]:
```
print("Program:                   {}".format(result.program))
print("Number of Basis Functions: {}".format(result.properties.calcinfo_nbasis))
print("Total execution time:      {:.2f}s".format(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 2D array:

[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:

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

[21]:
```
1.2700940280530457
```

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

[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
                        6 Threads,  62259 MiB Core
         ---------------------------------------

  ==> Geometry <==

    Molecular
```