QC$_{J}SON_{S}chemaDocumentation$

QC$_{J}SON_{S}chema$

Aug 14, 2020

Contents

A JSON Schema for Quantum Chemistry

The purpose of this schema is to provide API-like access to existing workhorse quantum chemistry packages to enable more complex and unified workflows. Primary to this is avoiding parsing ASCII-based output files, instead placing output variables, vectors, matrices in a consistent format that can be easily read/loaded by humans or tools.

CHAPTER 1

High-Level Aspirations

In order to help define the overall scope and direction of the specification, several high-level goals will be pursued:

- Connecting QC to visualizers and GUIs
- Connecting QC to existing Workflows tools
- Transfer data between QC programs (Orbitals, Densities, etc.)
- Provide a rigorous record of computation for large-scale QC databases
- Provide a framework for QC API access

A concrete list of requirements for this schema can be found [here](Requirements.md).

Organizations:

- The Molecular Sciences Software Institute

Visualizers:

- Avogadro
- Molecular Design Toolkit
- VTK
- Jmol / JSmol

Quantum Chemistry Engines:

- GAMESS
- MPQC
- NWChem
- Q-Chem
- Psi4
- PySCF

Translators:

- cclib
- openbabel

Utilities:

- geomeTRIC

CHAPTER 2

Existing JSON Efforts

JSON or XML based input or output is a common abstraction with quantum chemistry. The idea is to pull from the wide and coalesce into a single specification to prevent duplication of effort.

- Autodesk JSON
- BAGEL JSON
- Chemical JSON
- MPQC JSON
- NWChem JSON
- Psi4 JSON
- PyQC Schema
- Molpro Database XML
- Chemical Markup Language

CHAPTER 3

Contents

3.1 Specification Components

A brief overview of the fields present in the QC Schema is contained below. It should be noted that a significant amount of customization can be added to each field, please see the Schema or Examples document section for further information.

3.1.1 Input Components

Topology

The closest representation to the real physical nature of the system. In practical terms, for molecular sciences, this is the coordinates (in some form) and the elements/Z-number at that coordinate. For both QM and MM, this is your molecule. This may include bonding information and unit cell, lattice parameters, etc, as well.

This is the foundation upon which you build the model basis of your calculation.

A water molecule example:

```
{
  "molecule": {
    "geometry": [
      0.0,  0.0000, -0.1294,
      0.0, -1.4941,  1.0274,
      0.0,  1.4941,  1.0274
    ],
    "symbols": ["O", "H", "H"]
  }
}
```

Driver

What are you looking to calculate: energy, gradient, Hessian, or property.

A energy call:

```
{
  "driver": "energy"
}
```

Model

The overall mathematical model we are using for our calculation. Another way to think about this is the largest superset that still obtains roughly the same result. In QM, this is the Hamiltonian (HF, DFT, . . .) combined with the overall basis of the calculation. An example in QM would be HF/STO-3G or B3LYP/6-311G**. Custom basis sets can be handled with custom keywords.

A example B3LYP call in the cc-pVDZ basis.

```
{
  "model": {
    "method": "B3LYP",
    "basis": "cc-pVDZ"
  }
}
```

Keywords

Various tunable parameters for the calculation. These vary widely, depending on the basis and model chemistry. These represent the keywords of individual programs currently.

Program specific keywords requesting a density-fitting SCF call and a specific energy convergence tolerance:

```
{
  "keywords": {
    "scf_type": "df",
    "e_congerence": 1.e-7
  }
}
```

3.1.2 Output Components

Input Components

The input components are duplicated in the output so that the result is a complete trace of the requested computation from input specification to results.

Success

A description if the computation was successful or not. For unsuccessful computations standard errors will be placed in the output such as convergence, IO errors, etc.

A successful example:

```
{
  "success": true,
{
```

An unsuccessful example:

```
{
  "success": false,
  "error": {
    "error_type": "convergence_error",
    "errorm_message": "SCF failed to converge after 50 iterations"
{
```

Returned Result

The "primary" return of a given computation. For energy, gradient, and Hessian quantities these are either single numbers or a array representing the derivative quantity.

A simple "energy" driver example:

```
{
   "return_result": -76.4187620271478
{
```

Provenance

A brief description of the program, version, and routine used to generate the output. Can include more detailed information such as computation time, processor information, and host location.

```
{
  "provenance": {
    "creator": "QM Program",
    "version": "1.1",
    "routine": "module.json.run_json"
  },
}
```

Properties

A set of intermediate values produced by the QM program such as the one-elecron and two-electron energies in SCF. In addition, this will include such values such as the number of atomic orbitals and the number of alpha and beta electrons.

An example properties from a water HF/cc-pVDZ computation:

```
{
  "properties": {
    "calcinfo_nbasis": 24,
    "calcinfo_nmo": 24,
    "calcinfo_nalpha": 5,
    "calcinfo_nbeta": 5,
    "scf_one_electron_energy": -122.44534536383044,
    "scf_two_electron_energy": 37.622464940400654,
```

(continues on next page)

(continued from previous page)

```
    "nuclear_repulsion_energy": 8.80146205625184,
    "scf_total_energy": -76.02141836717794
  }
}
```

Know variable lists include:

- IUPAC Goldbook
- Units/Constants
- Electron Density
- IUPAC recommendations for computational chemistry
- IUPAC recommendations are product of IUPAC Projects
- IUPAC InChI related activities beyond organics
- CCLibVars
- PsiVars
- Codessa

Basis Quantities

The schema supports the export of basis quantities such as the overlap matrix or the orbitals. TBD

3.2 Frequently Asked Questions

3.2.1 Will the json be validated before it reaches my software?

This is a question for the producer and consumers of the QC Schema. It is certainly recommended to validate the schema and validation can be accomplished in a variety of langauges found at the JSON-Schema website.

3.2.2 Does the schema accept arbitrary extra fields if my software piece needs internal extensions?

Yes, we are currently discussing which fields are reserved and where the best place for arbitrary fields would be.

3.2.3 Are there libraries for writing the schema in [programming-language]?

JSON is agnostic to the underlying programming language and is well supported in a variety of languages (C++/Python/JS/etc). We will provide examples on how to write JSON in other languages where JSON is not as well supported (Fortran).

3.2.4 Why not use XML?

The ability to hand write and tweak a given input has been a sought after property. In addition, the overall structure of JSON is viewed as simpler and more intuitive than XML. As the schema is fully specified it should be possible for the validator to take in a JSON input and return an XML output.

3.2.5 What style will be used for indexing and case?

We will support zero-indexing for arrays and snake_case for keys. Discussion is underway if we will follow the Google JSON Style Guide.

3.2.6 Will the schema be versioned?

Yes, the schema will have version flags so that the Schema can evolve over time.

3.3 Technical Discussion

This document contains various technical considerations that are both open and those which have been discussed and closed.

3.3.1 Open Questions

How do we reference other objects?

JSON does not directly support object references. This makes it non-trivial to, say, maintain a list of bonds between atoms. Some solutions are:

1) by array index (e.g., `residue.atom_indices=[3,4,5,6]`)
2) by JSON path reference (see, e.g., https://tools.ietf.org/html/draft-pbryan-zyp-json-ref-03)
3) JSON-LD allows some flexibility of referencing. Also gives flexibility to break one document or one JSON object into pieces that can be referenced against.
4) by a unique key. (e.g., `residue.id='a83nd83'`, `residue.atoms=['a9n3d9', '31di3']`)

Array index is probably the best option - although they are a little fragile, they're no more fragile than path references, and require far less overhead than unique keys.

We need to look at this beyond atoms and bonds. Especially in workflows we can reuse pieces of data from previous tasks in the workflow. Instead of repeating we can use referencing.

See also: http://stackoverflow.com/q/4001474/1958900

How do we uniquely specify physical units?

For instance, velocity might be "angstrom/fs" Alternatives:

1) Require units in the form `{unit_name:exponent}`, e.g. `atom.velocity.units={'angstrom':1, 'fs':-1}`
2) Allow strings of the form `atom.velocity.units="angstrom/fs"`, but require that units be chosen from a specific list of specifications
3) Allow strings of the form `atom.velocity.units="angstrom/fs"`, and require file parsers to parse the units according to a specified syntax

Note: There are multiple standards specifications for units, and conversions. If done right in a schema, you can use JSON-LD to link to the actual standards definition. Some examples in CML:

```
"orbitalEnergy": {"units": "Hartree", "value": 0.935524}
"shieldingAnisotropy": {"units": "ppm","value": 17.5292}
```

JSON and HDF5

The object specifications in this document are tailored to JSON, but can be easily stored in an HDF5 file as well. HDF5 is, like JSON, hierarchical and self-describing. These similarities make it easy to perform 1-to-1 transformations between well-formed JSON and a corresponding HDF5 representation.

Unlike JSON, HDF5 is binary and requires custom libraries to read, but has far better performance and storage characteristics for numerical data. We will provide tools to easily interconvert files between JSON and HDF5. Applications that support this format should always provide JSON support; ones that require high performance should also support the HDF5 variant.

3.3.2 Closed Questions

Store large collections of objects

There exists multiple ways to arrange data which represents objects. These expressions come down to two primary categories:

The "big" approach where each field is a flat (1D) array for each category:

```
{
    "symbols": ["C", "C", ...],
    "geometry": [0.000, 1.396, 0.000, 1.209, 0.698, 0.000, ...],
    "masses": [12.017, 12.017, ...]
}
```

The "small" approach which has a closer object-base mapping:

```
{
    "fields": ["symbols", "geometry", "masses"],
    "table": [
        ["C", [0.000, 1.396, 0.000], 12.017],
        ["C", [1.209, 0.698, 0.000], 12.017],
        ...
    ]
}
```

For the QC Schema it was decided to follow the big approach as it has the following benefits:

- Serialization/deserialization is much faster due to the smaller number of objects generated.
- The "small" approach can lead to a complex hierachy of fields.
- It is generally thought the "big" approach is more straightfoward to program due to its flatter structure.

3.4 Examples

Several examples of completed schema. As the input is duplicated in the output the corresponding input of these schema are the input fields alone. Effectively, this is all keys above the "provenance" field. For clarify all array-based values have been truncated to four decimal places.

3.4.1 Water MP2 Energy

```
{
  "schema_name": "qc_schema_output",
  "schema_version": 1,
  "molecule": {
    "geometry": [
      0.0,  0.0,    -0.1294,
      0.0, -1.4941,  1.0274,
      0.0,  1.4941,  1.0274
    ],
    "symbols": ["O", "H", "H"]
  },
  "driver": "energy",
  "model": {
    "method": "MP2",
    "basis": "cc-pVDZ"
  },
  "keywords": {},
  "provenance": {
    "creator": "QM Program",
    "version": "1.1",
    "routine": "module.json.run_json"
  },
  "return_result": -76.22836742810021,
  "success": true,
  "properties": {
    "calcinfo_nbasis": 24,
    "calcinfo_nmo": 24,
    "calcinfo_nalpha": 5,
    "calcinfo_nbeta": 5,
    "calcinfo_natom": 3,
    "return_energy": -76.22836742810021,
    "scf_one_electron_energy": -122.44534536383037,
    "scf_two_electron_energy": 37.62246494040059,
    "nuclear_repulsion_energy": 8.80146205625184,
    "scf_dipole_moment": [0.0, 0.0, 2.0954],
    "scf_iterations": 10,
    "scf_total_energy": -76.02141836717794,
    "mp2_same_spin_correlation_energy": -0.051980792916251864,
    "mp2_opposite_spin_correlation_energy": -0.15496826800602342,
    "mp2_singles_energy": 0.0,
    "mp2_doubles_energy": -0.20694906092226972,
    "mp2_total_correlation_energy": -0.20694906092226972,
    "mp2_total_energy": -76.22836742810021
  }
}
```

3.4.2 Water HF Gradient

```
{
  "schema_name": "qc_schema_output",
  "schema_version": 1,
  "molecule": {
    "geometry": [
```

(continues on next page)

(continued from previous page)

```
      0.0,  0.0,    -0.1294,
      0.0, -1.4941,  1.0274,
      0.0,  1.4941,  1.0274
    ],
    "symbols": ["O", "H", "H"]
  },
  "driver": "gradient",
  "model": {
    "method": "HF",
    "basis": "cc-pVDZ"
  },
  "keywords": {},
  "provenance": {
    "creator": "QM Program",
    "version": "1.1",
    "routine": "module.json.run_json"
  },
  "return_result": [
    0.0,  0.0,    -0.0595,
    0.0, -0.0430,  0.0297,
    0.0,  0.0430,  0.0297
  ],
  "success": true,
  "properties": {
    "calcinfo_nbasis": 24,
    "calcinfo_nmo": 24,
    "calcinfo_nalpha": 5,
    "calcinfo_nbeta": 5,
    "calcinfo_natom": 3,
    "return_energy": -76.02141836717794,
    "scf_one_electron_energy": -122.44534536383044,
    "scf_two_electron_energy": 37.622464940400654,
    "nuclear_repulsion_energy": 8.80146205625184,
    "scf_dipole_moment": [0.0, 0.0, 2.0954],
    "scf_iterations": 10,
    "scf_total_energy": -76.02141836717794
  }
}
```

3.5 Topology Schema

A full description of the overall molecule its geometry, fragments, and charges.

3.5.1 Required Keys

The following properties are required for a topology.

Key Name	Description	Field Type
symbols	(nat,) atom symbols in title case.	array[string]
geometry	(3 * nat,) vector of XYZ coordinates [a0] of the atoms.	array[number]
schema_name	No description provided.	string
schema_version	No description provided.	integer

3.5.2 Optional Keys

The following keys are optional for the topology specification.

Key Name	Description	Field Type
molecular_charge	The overall charge of the molecule.	number
comment	Any additional comment one would attach to the molecule.	string
fragments	(nfr, <varies>) list of indices (0-indexed) grouping atoms into molecular fragments within the topology.	array[array]
fragment_multiplicities	(nfr,) list of multiplicities associated with each fragment tuple.	array[number]
connectivity	A list describing bonds within a molecule. Each element is a (atom1, atom2, order) tuple.	array[array]
fix_com	Whether translation of geometry is allowed (fix F) or disallowed (fix T).	boolean
molecular_multiplicity	The overall multiplicity of the molecule.	number
fix_symmetry	Maximal point group symmetry at which *geometry* should be treated. Lowercase.	string
name	The name of the molecule.	string
fragment_charges	(nfr,) list of charges associated with each fragment tuple.	array[number]
fix_orientation	Whether rotation of geometry is allowed (fix F) or disallowed (fix T).	boolean
provenance	#/definitions/provenance	object
atomic_numbers	(nat,) atomic numbers, nuclear charge for atoms. Ghostedness should be indicated through 'real' field, not zeros here.	array[number]
masses	(nat,) atom masses [u]; canonical weights assumed if not given.	array[number]
atom_labels	(nat,) atom labels with any user tagging information.	array[string]
real	(nat,) list describing if atoms are real (T) or ghost (F).	array[boolean]
mass_numbers	(nat,) mass numbers for atoms, if known isotope, else -1.	array[number]

3.6 Properties Schema

A list of valid quantum chemistry properties tracked by the schema.

3.6.1 Calculation Information

A list of fields that involve basic information of the requested computation.

Key Name	Description	Field Type
cal-cinfo_nbasis	The number of basis functions for the computation.	number
cal-cinfo_nmo	The number of molecular orbitals for the computation.	number
cal-cinfo_nalpha	The number of alpha electrons in the computation.	number
cal-cinfo_nbeta	The number of beta electrons in the computation.	number
cal-cinfo_natom	The number of atoms in the computation.	number
re-turn_energy	The energy of the requested method, identical to *return_value* for energy computations.	number

3.6.2 Self-Consistent Field

A list of fields added at the self-consistent field (SCF) level. This includes both Hartree–Fock and Density Functional Theory.

Key Name	Description	Field Type
scf_one_electron_energy	The one-electron (core Hamiltonian) energy contribution to the total SCF energy.	number
scf_two_electron_energy	The two-electron energy contribution to the total SCF energy.	number
nu-clear_repulsion_energy	The nuclear repulsion energy contribution to the total SCF energy.	number
scf_vv10_energy	The VV10 functional energy contribution to the total SCF energy.	number
scf_xc_energy	The functional energy contribution to the total SCF energy.	number
scf_dispersion_correction_energy	The dispersion correction appended to an underlying functional when a DFT-D method is requested.	number
scf_dipole_moment	The X, Y, and Z dipole components.	ar-ray[number]
scf_total_energy	The total electronic energy of the SCF stage of the calculation. This is represented as the sum of the … quantities.	number
scf_iterations	The number of SCF iterations taken before convergence.	number

3.6.3 Moller-Plesset

A list of fields added at the Moller–Plesset (MP) level.

Key Name	Description	Field Type
mp2_same_spin_correlation_energy	The portion of MP2 doubles correlation energy from same-spin (i.e. triplet) correlations, without any user scaling.	number
mp2_opposite_spin_correlation_energy	The portion of MP2 doubles correlation energy from opposite-spin (i.e. singlet) correlations, without any user scaling.	number
mp2_singles_energy	The singles portion of the MP2 correlation energy. Zero except in ROHF.	number
mp2_doubles_energy	The doubles portion of the MP2 correlation energy including same-spin and opposite-spin correlations.	number
mp2_correlation_energy	The MP2 correlation energy.	number
mp2_total_energy	The total MP2 energy (MP2 correlation energy + HF energy).	number
mp2_dipole_moment	The MP2 X, Y, and Z dipole components.	array[number]

3.6.4 Coupled Cluster

A list of fields added at the Coupled Cluster (CC) level.

Key Name	Description	Field Type
ccsd_same_spin_correlation_energy	The portion of CCSD doubles correlation energy from same-spin (i.e. triplet) correlations, without any user scaling.	number
ccsd_opposite_spin_correlation_energy	The portion of CCSD doubles correlation energy from opposite-spin (i.e. singlet) correlations, without any user scaling.	number
ccsd_singles_energy	The singles portion of the CCSD correlation energy. Zero except in ROHF.	number
ccsd_doubles_energy	The doubles portion of the CCSD correlation energy including same-spin and opposite-spin correlations.	number
ccsd_correlation_energy	The CCSD correlation energy.	number
ccsd_total_energy	The total CCSD energy (CCSD correlation energy + HF energy).	number
ccsd_prt_pr_correlation_energy	The CCSD(T) correlation energy.	number
ccsd_prt_pr_total_energy	The total CCSD(T) energy (CCSD(T) correlation energy + HF energy).	number
ccsdt_correlation_energy	The CCSDT correlation energy.	number
ccsdt_total_energy	The total CCSDT energy (CCSDT correlation energy + HF energy).	number
ccsdtq_correlation_energy	The CCSDTQ correlation energy.	number
ccsdtq_total_energy	The total CCSDTQ energy (CCSDTQ correlation energy + HF energy).	number
ccsd_dipole_moment	The CCSD X, Y, and Z dipole components.	array[number]
ccsd_prt_pr_dipole_moment	The CCSD(T) X, Y, and Z dipole components.	array[number]
ccsdt_dipole_moment	The CCSDT X, Y, and Z dipole components.	array[number]
ccsdtq_dipole_moment	The CCSDTQ X, Y, and Z dipole components.	array[number]
ccsd_iterations	The number of CCSD iterations taken before convergence.	number
ccsdt_iterations	The number of CCSDT iterations taken before convergence.	number
ccsdtq_iterations	The number of CCSDTQ iterations taken before convergence.	number

3.7 Wavefunction Schema

A list of valid quantum chemistry wavefunction properties tracked by the schema. Matrices are in column-major order. AO basis functions are ordered according to the CCA standard as implemented in libint.

3.7.1 Basis Set

One-electron AO basis set. See *Basis Set Schema*.

3.7.2 Result

A list of fields comprising the primary result information. e.g. SCF quantities for a DFT calculation and MP2 quantities for an MP2 calculation. Result fields contain the names of other fields in the wavefunction schema.

Key Name	Description	Field Type
orbitals_a	Alpha-spin orbitals in the AO basis of the primary return.	string
orbitals_b	Beta-spin orbitals in the AO basis of the primary return.	string
density_a	Alpha-spin density in the AO basis of the primary return.	string
density_b	Beta-spin density in the AO basis of the primary return.	string
fock_a	Alpha-spin Fock matrix in the AO basis of the primary return.	string
fock_b	Beta-spin Fock matrix in the AO basis of the primary return.	string
eigenvalues_a	Alpha-spin orbital eigenvalues of the primary return.	string
eigenvalues_b	Beta-spin orbital eigenvalues of the primary return.	string
occupations_a	Alpha-spin orbital occupations of the primary return.	string
occupations_b	Beta-spin orbital occupations of the primary return.	string

3.7.3 Self-Consistent Field

A list of fields added at the self-consistent field (SCF) level. This includes both Hartree–Fock and Density Functional Theory.

Key Name	Description	Field Type
scf_orbitals_a	SCF alpha-spin orbitals in the AO basis.	array[number]
scf_orbitals_b	SCF beta-spin orbitals in the AO basis.	array[number]
scf_density_a	SCF alpha-spin density in the AO basis.	array[number]
scf_density_b	SCF beta-spin density in the AO basis.	array[number]
scf_fock_a	SCF alpha-spin Fock matrix in the AO basis.	array[number]
scf_fock_b	SCF beta-spin Fock matrix in the AO basis.	array[number]
scf_coulomb_a	SCF alpha-spin Coulomb matrix in the AO basis.	array[number]
scf_coulomb_b	SCF beta-spin Coulomb matrix in the AO basis.	array[number]
scf_exchange_a	SCF alpha-spin exchange matrix in the AO basis.	array[number]
scf_exchange_b	SCF beta-spin exchange matrix in the AO basis.	array[number]
scf_eigenvalues_a	SCF alpha-spin orbital eigenvalues.	array[number]
scf_eigenvalues_b	SCF beta-spin orbital eigenvalues.	array[number]
scf_occupations_a	SCF alpha-spin orbital occupations.	array[number]
scf_occupations_b	SCF beta-spin orbital occupations.	array[number]

3.7.4 Localized Orbitals

A list of fields added at by orbital localization. Full MO matrices are stored even if only a subset of MOs are localized.

Key Name	Description	Field Type
localized_orbitals_a	Localized alpha-spin orbitals in the AO basis. All nmo orbitals are included, even if only a subset were localized.	array[number]
localized_orbitals_b	Localized beta-spin orbitals in the AO basis. All nmo orbitals are included, even if only a subset were localized.	array[number]
localized_fock_a	Alpha-spin Fock matrix in the localized molecular orbital basis. All nmo orbitals are included, even if only a subset were localized.	array[number]
localized_fock_b	Beta-spin Fock matrix in the localized molecular orbital basis. All nmo orbitals are included, even if only a subset were localized.	array[number]

3.7.5 Core Hamiltonian

A list of fields associated with (effective) one-electron (AKA) core Hamiltonians.

Key Name	Description	Field Type
h_core_a	Alpha-spin core (one-electron) Hamiltonian in the AO basis.	array[number]
h_core_b	Beta-spin core (one-electron) Hamiltonian in the AO basis.	array[number]
h_effective_a	Alpha-spin effective core (one-electron) Hamiltonian in the AO basis.	array[number]
h_effective_b	Beta-spin effective core (one-electron) Hamiltonian in the AO basis.	array[number]

3.8 Basis Set Schema

A full description of the basis set.

3.8.1 Required Keys

The following properties are required for a basis set.

Key Name	Description	Field Type
center_data	Shared basis data for all atoms/centers in the molecule	object
atom_map	Mapping of all atoms/centers in the molecule to data in center_data	array[string]
name	Name of the basis set	string

3.8.2 Optional Keys

The following keys are optional for the basis set specification.

Key Name	Description	Field Type
schema_name	No description provided.	string
description	Brief description of the basis set	string
schema_version	No description provided.	integer