Received: 23 August 2020 | Accepted: 6 October 2020

DOI: 10.1002/jcc.26468

SOFTWARE NEWS AND UPDATES

Journal of COMPUTATIONAL CHEMISTRY WILEY

IOData: A python library for reading, writing, and converting computational chemistry file formats and generating input files

Toon Verstraelen[1] | William Adams[2] | Leila Pujal[3] | Alireza Tehrani[3] | Braden D. Kelly[2] | Luis Macaya[4] | Fanwang Meng[2] | Michael Richer[2] | Raymundo Hernández-Esparza[2] | Xiaotian Derrick Yang[2,5] | Matthew Chan[2] | Taewon David Kim[2] | Maarten Cools-Ceuppens[1] | Valerii Chuiko[2,6] | Esteban Vöhringer-Martinez[4] | Paul W. Ayers[2] | Farnaz Heidar-Zadeh[3]

[1]Center for Molecular Modeling (CMM), Ghent University, Zwijnaarde, Belgium

[2]Department of Chemistry and Chemical Biology, McMaster University, Hamilton, Ontario, Canada

[3]Department of Chemistry, Queen's University, Kingston, Ontario, Canada

[4]Departamento de Físico Química, Facultad de Ciencias Químicas, Universidad de Concepción, Concepción, Chile

[5]Laboratoire de Chimie Théorique, Sorbonne Université, Paris, France

[6]Faculty of Physics, Taras Shevchenko National University of Kyiv, 01601, Kyiv, Ukraine

Correspondence
Toon Verstraelen, Center for Molecular Modeling (CMM), Ghent University, Technologiepark-Zwijnaarde 46, B-9052, Zwijnaarde, Belgium.

Esteban Vöhringer-Martinez, Departamento de Físico Química, Facultad de Ciencias Químicas, Universidad de Concepción, 4070371 Concepción, Chile.
Email: evohringer@udec.cl

Paul W. Ayers, Department of Chemistry and Chemical Biology, McMaster University, Hamilton, Ontario, L8S-4L8, Canada.
Email: ayers@mcmaster.ca

Farnaz Heidar-Zadeh, Department of Chemistry, Queen's University, Kingston, Ontario, K7L-3N6, Canada.
Email: farnaz.heidarzadeh@queensu.ca

Funding information
Fondo Nacional de Desarrollo Científico y Tecnológico; Fonds Wetenschappelijk Onderzoek; Natural Sciences and Engineering Research Council of Canada; Queen's University Research Initiation Grant; Canada Research Chairs, Compute Canada, and CANARIE; Max-Planck Society; PCI CONICYT Instituto Max Planck For Terrestrial Microbiology Marburg MPG190003; CONICYT/FONDECYT/REGULAR/FOLIO, Grant/Award Number: 1200369; Research Board of Ghent University (BOF)

Abstract

IOData is a free and open-source Python library for parsing, storing, and converting various file formats commonly used by quantum chemistry, molecular dynamics, and plane-wave density-functional-theory software programs. In addition, IOData supports a flexible framework for generating input files for various software packages. While designed and released for stand-alone use, its original purpose was to facilitate the interoperability of various modules in the HORTON and ChemTools software packages with external (third-party) molecular quantum chemistry and solid-state density-functional-theory packages. IOData is designed to be easy to use, maintain, and extend; this is why we wrote IOData in Python and adopted many principles of modern software development, including comprehensive documentation, extensive testing, continuous integration/delivery protocols, and package management. This article is the official release note of the IOData library.

KEYWORDS

basis set conversion, chemistry software development, computational chemistry, data parsing, file format conversion, input file generation, JSON schema, molecular mechanics, quantum chemistry, theoretical chemistry Python library

1 | WHAT IS IODATA?

IOData is a free and open-source Python 3 library for reading and writing formatted (non-binary) files generated by a broad array of molecular quantum chemistry, molecular dynamics, and plane-wave density-functional-theory programs. IOData is primarily intended as a utility for (a) parsing molecular/periodic geometry, wavefunction, and trajectory data from various file formats, (b) storing this data in a user-friendly object, so that it can be used in external Python modules, (c) dumping geometry, wavefunction, and trajectory data in a variety of standard file formats, hence (d) converting one standard file format to another, and (e) generating input files for various computational chemistry software packages. Section 5 details the file formats and input file generation currently supported by IOData; however, we designed the IOData framework to be not only easy to use but also easy to extend to new data formats and software.

2 | HISTORY

During the course of our scientific research, and especially our development of the HORTON[1] and ChemTools[2] software packages, we needed a utility that allowed us to parse the output files of various electronic structure theory programs and dump data from our software packages using standard file formats. Some of this functionality existed in HORTON 2.x,[1] but in the process of writing HORTON[3], we decided to split IOData (as well as much of the other functionality of HORTON) into stand-alone libraries to better support modularity, maintenance, and ease of use.

3 | ABOUT IODATA

IOData is, and always will be, a free and open-source library distributed under the GNU General Public License. The IOData source code is maintained on the GitHub platform; see https://github.com/theochem/iodata, and its documentation is hosted on Read the Docs; see https://iodata.readthedocs.io/en/latest/index.html. We strive to ensure that the IOData source code and website itself is comprehensively documented, including useful tests, scripts, and examples. As that documentation is maintained with the software, providing detailed (and eventually outdated) release notes here seems unwise. Instead, we will briefly list the distinguishing features and key capabilities of IOData in Section 5 and demonstrate them in Section 6.

4 | WHY IODATA?

Before starting to work on IOData, we considered using other free and open-source utilities for parsing and converting file formats like OpenBabel[3] (mostly geared toward chemoinformatics), RDKit[4] (limited wavefunction capabilities), cclib[5] (limited capabilities for dumping file formats), ASE[6] (limited features for molecular systems), MultiWFN[7] (difficult-to-extend Fortran 90 source code), MDTraj[8] and MDAnalysis[9,10] (both geared toward trajectory formats and analysis). However, none of these achieved our criteria of being easy to use and modify, readable, stand-alone, modular, and written in Python. Most critically, most other software packages are limited in their ability to parse molecular wavefunction information and extremely limited in their ability to write popular quantum chemistry output file formats.

Anticipating that other researchers may have similar needs that can be met by directly using IOData or by building on top of it, we decided to release IOData as a free and open-source library. Nonetheless, we wish to acknowledge the utility of these other programs, which we also use in our research. OpenBabel[3] is extremely useful for cheminformatics tasks; RDKit is an excellent tool for generating molecular structures and conformations; cclib has refined capabilities for parsing an abundance of information from quantum chemistry output/log file formats; ASE supports an abundance of solid-state electronic structure programs; MDTraj and MDAnalysis support an impressive number of trajectory formats; MultiWFN has comprehensive post-processing capabilities and supports many of the most important formats supported by IOData, but does not produce an internal data structure (e.g., a dictionary or a class) that is easily used in other programs. While IOData overlaps with these tools in some respects, its emphasis on reading and writing molecular quantum chemistry data file formats, generating input files, and usability as a library within other programs is unique.

In our view, the distinguishing characteristics of IOData are its modularity, its flexibility, and its ease of use. We achieve this by designing a unifying framework in which data from various file formats can be loaded, stored in a versatile container class, and dumped in a desired file format. We provide comprehensive documentation and rigorously adhere to high coding standards, including comprehensive testing of code correctness, quality, and readability. While IOData is primarily intended to be used as a Python library and its API was designed to facilitate interoperability with other Python packages, many operations can be performed through its command line tools.

5 | FEATURES OF IODATA

We display various features of IOData by first discussing file formats that it can currently analyze. Specifically,

- The file formats (and their commonly used extensions) that can be both loaded and dumped by IOData include: basic XYZ Cartesian coordinate files (*.xyz), extended XYZ files containing additional atomic properties (*.extxyz), Gaussian[11] formatted checkpoint files (*.fchk), wavefunction files (*.wfn), extended wavefunction files (*.wfx), MultiWFN files (*.mwfn), Molden[12] files (*.molden), Molekel files (*.mkl), Mol2 files (*.mol2), Molpro[13] 2012 FCIDUMP integral files (*.molpro), spatial data files (*.sdf), protein data bank files (*.pdb), QCSchema[14] files (*.json), and cube files (*.cube). Most often,

these file formats each contain information on a single chemical compound, and are commonly used in exchanging information between various software packages (e.g., for post-processing). Several of these file formats are often used to store information on multiple compounds in a single file (e.g., concatenation of a collection of molecules/frames in a single file with a single format). In this regard, `IOData` also supports load and dump functionality for concatenated *.xyz, *.extxyz, *.fchk, *.mol2, *.sdf, and *.pdb file formats.

- The file formats (and their commonly used extension) that can be only loaded by `IOData` include certain files from plane-wave density-functional theory codes including VASP (*.chgcar, *.locpot, *.poscar), molecular dynamics packages like Gromacs (*.gro) and CHARMM (*.crd) coordinate files, and `GAMESS`[15] punch files (*.dat). `IOData` can also partially parse and store data from log files of several quantum chemistry software packages including `Gaussian`[11] log files (*.log), `ORCA`[16] log files (*.out), `Q-Chem`[17] log files (*.qchemlog), and atomic `CP2K`[18] log files (*.cp2k.out). If the user is interested in a specific information printed in these log file that is not currently loaded by `IOData`, the existing parser can be easily extended.
- To support a broad range of external software, `IOData` supports and interconverts between arbitrary basis-set conventions, including basis-function order, sign (of the spherical harmonics), and normalization (L^1-normalized, L^2-normalized, and unnormalized). `IOData` also supports basis set conversion (e.g., Cartesian to/from spherical, primitive to/from contracted).
- `IOData` can be used to generate input files for various quantum chemistry and molecular dynamics software packages, including `Gaussian`[11] and `ORCA`.[16] Users have the option of providing an input template for specific use cases or using pre-defined basic templates. The input file writer can be easily extended for use with other software packages.

Considering the large number of file formats commonly used in computational chemistry, the `IOData` library is particularly useful as it provides a user-friendly platform to load the information contained in various file formats into an easy-to-use object and to convert one file format to the other. `IOData`'s file-conversion capability is especially useful because software packages usually work with specific file formats. Obviously, the XYZ and cube file formats cannot be used to generate file formats containing wavefunction information, however, any of the above mentioned formats can be loaded by `IOData` (like fchk) and converted to any other format (like XYZ, wfn, molden, etc.).

As depicted in Figure 1, the `IOData` library stores the loaded data as attributes of the `IOData` class. These are mostly numerical data, arrays, or strings; if available, the information related to molecular basis set and orbitals are stored as the "obasis" and "mo" attributes, which are instances of the `MolecularBasis` class and `MolecularOrbitals` class, respectively. Occasionally, file formats include information on 1- and 2-electron integrals, reduced density matrices, multipole moments, atomic charges, and/or force field

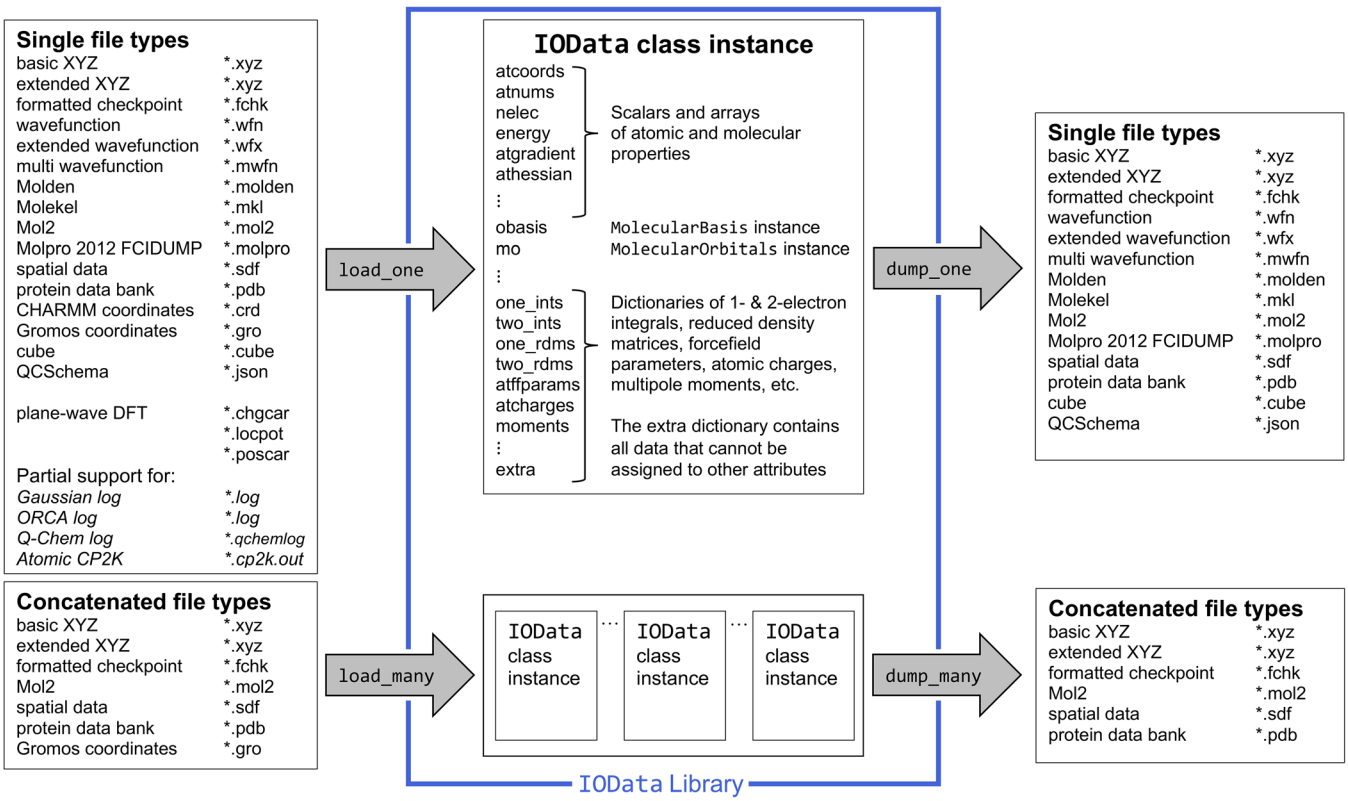

FIGURE 1 Schematic representation of `IOData` library

parameters; these are each stored as dictionaries. Any other additional information that does not fit into a pre-existing field is stored in the "extra" dictionary attribute of the `IOData` class; items in "extra" may become standard attributes in future releases.

We aim to ensure that file formats can be parsed robustly, so that even when popular software packages fail to follow the technical specifications of a file format (usually because the file format was not strictly or publicly defined) the error is automatically diagnosed and corrected if possible. For example, many programs read and generate *.wfn files that do not follow the original format statements from Richard Bader's `AIMPAC` program, and no formal file format specifications for *.wfn seems ever to have been published. To remedy this, we looked at four *.wfn formats from various open-source projects and wrote a *.wfn format specification that was as flexible as possible, yet still supported all four versions. The file format specification is provided as part of the `IOData` documentation, and hopefully will prevent rogue *.wfn file formats in the future. Another example is that different programs (`PSI4`,[19] `ORCA`,[16] `Turbomole`,[20] `Molden`,[12] `Molekel`, `Q-Chem`,[17] etc.) generate *.molden files with different basis-set normalization, sign, and ordering conventions. We automatically detect these conventions by evaluating the atomic-orbital overlap matrix, and then store the basis set in a flexible and universal way. When writing a *.molden file, we always use the same normalization convention that the Molden program itself uses.

Among the recently proposed file formats, `IOData` supports loading and dumping `QCSchema`,[14] a program-agnostic JSON schema that serves as both an input and output file format. Unfortunately, the implementation of the `QCSchema` in MolSSI projects such as `QCElemental`[21] has diverged from the original `QCSchema` specification.[14] The version implemented in `QCElemental` has transformed from a single, extended schema to four different sub-schema components: Molecule, Input, Output, and Basis. This reinvention of `QCSchema` is reasonable, but only exists as a Python-based partial implementation in `QCElemental`, not a true JSON schema specification usable by other potential adopters. `IOData` implements a backward-compatible and complete version of `QCSchema` by adopting many of the changes found in `QCElemental` and the development branch of `QCSchema` while also providing a genuine JSON schema and extensive documentation specifying each of the four sub-schema.

`IOData` uses an extensive automatic-testing framework to verify that the code behaves correctly, both when introducing code changes and when making new installations on various operating systems. Currently, `IOData` uses `pytest` framework and the unit test coverage feature of `codecov.io` to ensure a significant portion of our code is well tested; at present, 96% of the code base is tested. In addition, `IOData` uses an extensive set of quality-assurance tools (`pylint`, `autopep8`, `pycodestyle`, `pydocstyle`, etc.) to ensure that the code quality is not degraded by new contributions and the code remains readable and well-documented. The testing and quality-assurance workflows are automatically run by our continuous integration system every time a code change is proposed to our GitHub repository. Currently, we use `travis.io` for continuous integration, delivery, and deployment. If `travis.io` reports that the unit tests or quality-assurance tools is failed, the proposed changes are automatically rejected. When our code is ready for release, we Git tag a commit with a special keyword and `travis.io` builds and tests a release with the Conda and Pip package managers. This allows users to simply install `IOData` and all its binary dependencies in one command. Developers can also publish new releases to users in one command thanks to the automation provided by this workflow.

6 | EXAMPLES

For the most updated documentation and examples on how to use `IOData`, please refer to the `IOData` website. Here, we showcase several ways `IOData` can be used and incorporated into various workflows. Please note that these examples are based on version 1.0 of `IOData`, and the user might need to modify them if using future major releases of the `IOData` library. Within minor and bug-fix releases, backward compatibility is guaranteed.

6.1 | Using `IOData` as a Python library

Loading File Formats: The primary use case for `IOData` is to parse files of various formats and make the loaded data easily accessible to the user. This is mainly facilitated through the `load_one` and `load_many` functions, which return an `IOData` instance and a list of `IOData` instances, respectively, in which the loaded data is stored. For example,

```
# Loading Gaussian formatted checkpoint (fchk) file format
from iodata import load_one

data = load_one("water.fchk")       # instance of IOData class

# Examples of information that can be retrieved from the IOData instance
print(data.atnums)                  # atomic numbers
print(data.atcoords)                # atomic coordinates
print(data.nelec)                   # number of electrons
print(data.charge)                  # charge of molecule
print(data.obasis)                  # instance of MolecularBasis class
print(data.obasis.nbasis)           # number of basis functions
print(data.mo)                      # instance of MolecularOrbitals class
print(data.mo.energies)             # molecular orbital energies
print(data.atcharges["mulliken"])   # Mulliken atomic charges
```

The format of the file loaded is derived from its extension, however, the user can override the default file format detected using the optional `fmt` argument. The above example displays only a few of the `IOData` attributes; the data available from each file format is clearly tabulated on the `IOData` website. It is important to note that the wavefunction information is stored in an `IOData` instance in a manner designed to simplify calculations. For example, one can simply compute the spin-summed density matrix, for both restricted and unrestricted Slater determinants, using,

```
# Computing density matrix in the molecular orbital basis in 2 different ways
import numpy as np

dm = np.dot(data.mo.coeffs * data.mo.occs, data.mo.coeffs.T)              # using dot product
dm = np.einsum("ij,i,ij", data.mo.coeffs, data.mo.occs, data.mo.coeffs)   # using einsum
```

Similarly, the `load_many` function can be used to load concatenated file formats like databases (*.sdf, *.mol2, *.xyz, or *.extxyz) or successive conformations of molecules from molecular dynamics trajectories or reaction paths (*.pdb, *.fchk, *.xyz, or *.extxyz), stored in a single file. For example,

```
# Loading multiple molecules/frames contained in one XYZ file
from iodata import load_many

data = load_many("database.xyz")       # list of instances of IOData class

# Examples of information that can be retrieved
print(len(data))          # number of molecules/frames in the database
print(data[0])            # IOData instance of first molecule/frame
print(data[0].atcoords)   # atomic coordinates of first molecule/frame
print(data[-1].atnums)    # atomic numbers of last molecule/frame
```

Because many file formats are not strictly defined, the `IOData` library must be extremely flexible. For example, for the XYZ file format, `IOData` accepts atomic numbers, atomic symbols, or mixtures of both as atom specifications in the first column of the Coordinates section. `IOData` also allows the user to provide their own format specifications. For example, the many versions of the extended XYZ file format store various atomic properties by appending columns to the basic XYZ format. `IOData` can be instructed to load this additional data through the optional `atom_columns` argument of the `load_one` or `load_many` functions.

Writing File Formats: The information loaded from one file format can be used to write a file with a different file format, if the required data is available. This is made possible through the `dump_one` and `dump_many` functions, which take an `IOData` instance and sequence of `IOData` instances, respectively, and write out the specified file format. For example,

```
# Converting Gaussian formatted checkpoint (fchk) file format to Molden and XYZ file formats
from iodata import load_one, dump_one

data = load_one("water.fchk")                 # instance of IOData class
dump_one(data, "water.molden")                # write out a Molden file
dump_one(data, "water.xyz", fmt="xyz")        # write out an XYZ file
```

As mentioned previously, the format of a dumped file is derived from the filename extension; however, the user can override this default by specifying the optional `fmt` argument. Similarly, having a sequence of `IOData` instances, the `dump_many` function can be used to generate a single file containing the molecules/frames in the user-specified file format. Similarly, to generate an extended XYZ file, additional columns can be specified through the optional `atom_columns` argument of the `dump_one` or `dump_many` functions.

Storing Data: An instance of the `IOData` class can be constructed by the user, for example, in the process of performing a quantum chemistry calculation. This is helpful for both using the stored information internally (e.g., when using the `HORTON`[3] and `ChemTools`[2] software packages) or writing it out in a specific file format. For example,

```
# Storing information in IOData and dump file formats
import numpy as np
from iodata import IOData, dump_one

# Make an instance of IOData class with atomic numbers and coordinates (in Borh)
data = IOData(title="water")
data.atnums = np.array([8, 1, 1])
data.coordinates = np.array([[1.48123726, -0.93019123, 0.0], [0.0, 0.11720081, 0.0],
                             [-1.48123726, -0.93019123, 0.0]])

# Write out an XYZ file (with atomic coordinates in Angstrom)
dump_one(data, "water.xyz")
```

Writing Input Files: Input files for common software packages can be generated using either built-in or user-defined templates. The basic input writing functionality requires only an `IOData` instance with sufficient data for the input file (i.e., level of theory, basis set name, etc.). For example,

```
# Writing Gaussian input files from IOData instance
import numpy as np
from iodata import IOData, write_input

# Get an instance of IOData class by loading an XYZ file
data = load_one("water.xyz")
# Change the template's default level of theory (HF) & basis set (STO-3G)
data.lot = "B3LYP"
data.obasis_name = "cc-pVTZ"

# Generate Gaussian & ORCA input files using the default template, charge (0), & multiplicity (1)
write_input(data, "water.com", fmt="gaussian")
write_input(data, "water.in", fmt="orca", template=None)
```

The `write_input` function accepts a user-defined input `template` through template argument which is a text file containing Python string substitution markers and `IOData` attribute names, like ${lot} for level of theory, ${obasis_name} for basis set name, etc.

6.2 | Using `IOData` as a command-line tool

`iodata-convert` is the command-line file-conversion interface accessible upon installing the `IOData` library. For example,

```
# Convert Gaussian formatted checkpoint to a Molden file
iodata-convert water.fchk water.molden
```

The input and output file formats are derived from the file extensions, however, they can be specified through optional command-line arguments. To obtain more information on how to customize this utility, please refer to its help message,

```
iodata-convert --help
```

7 | FREQUENTLY ASKED QUESTIONS

Who is `IOData` for? We intend `IOData` to be primarily used by computational chemists and researchers developing quantum chemistry software, especially post-processing and visualization tools. Our goal is for `IOData` to be easy to use for novice programmers and computational chemists, yet capable of supporting advanced quantum chemistry workflows. Achieving these goals requires adherence to software design best practices, so extending or contributing to `IOData` requires intermediate-level programming ability (i.e. knowledge of decorators, type hints, automatic testing, GitHub, continuous integration, and automatic release). The command-line tools in `IOData` can be used by anyone who has passing familiarity with Linux shell commands.

What is the mission of `IOData`? We wish to serve the community of quantum chemistry users and developers by removing the barrier imposed by the plethora of (sometimes obscurely defined and/or documented) file formats. This is clearly a Sisyphean task, but we sincerely hope that others in the community will join our efforts and, if possible, eventually agree on a universal file format.

What does `IOData` do? As elaborated in Sections 5 and 6, `IOData` currently parses various file formats from third-party software packages and makes the data available to Python programs or writes out the data into a different format, and thus supports file format conversion. The latter can also be achieved through command-line functionality of `IOData`. In addition, `IOData` generates input files for various quantum chemistry or molecular dynamics software packages using basic or user-defined input file templates.

What is the future direction of `IOData`? We aim to support parsing and writing additional file formats, and also to extend our support for writing input files and enable direct integration with `QCEngine`.[21] Our current to-do list for `IOData` can be found on its GitHub's issue page, and we anticipate that most items will be resolved by the end of 2020. Within our software development efforts, the `IOData` will eventually serve as the input/output module of the `HORTON`[3] and `ChemTools`[2] software packages.

How do I install `IOData`? The `IOData` library can be installed from its source code available on GitHub or through *pip* and *conda* package-management systems. Maintaining the source code on GitHub allows users to access our latest development, even before it is officially released. For the most updated instructions on how to install `IOData`, please refer to the `IOData` website.

Can I contribute to `IOData`? Yes! We welcome and support new contributions in accordance with our Code of Conduct and Contributing Guidelines. For the most up-to-date instructions on how to contribute, please refer to the `IOData` website.

8 | SUMMARY

The purpose of this brief paper is to present the `IOData` module of `HORTON`[3], which parses, stores, and dumps a variety of file formats. The most unique capabilities of `IOData` are its robustness (e.g., its ability to read *.molden files with nonstandard basis-set conventions; its ability to read *.wfn files with strange fixed-formatting issues; the ability to read `QCSchema` with incompatible formats), its utilities to read and parse files for single and multiple structures (e.g., trajectories, reaction pathways, potential energy surface scans), its scope (including file formats typically used in Gaussian-basis-set-based molecular chemistry software, plane-wave density-functional-theory codes, and molecular dynamics calculations), its ability to output a variety of useful file formats, its meticulous documentation and testing protocols, its ease-of-use and extensibility, and its command-line tools. While we will continually improve `IOData`, we believe its present functionality already has significant benefits to the broader community of theoretical and computational chemists and we welcome their comments and contributions.

ACKNOWLEDGMENTS

We wish to acknowledge various refinements to the `IOData` library from Steven Vandenbrande, Jennifer Garner, Thomas Pigeon, Stijn Fias, Ali Malek, and the `HORTON` development team. T. V. acknowledges the Foundation of Scientific Research-Flanders (FWO) and the Research Board of Ghent University (BOF) for their financial support. L. M. and E. V. M. acknowledge financial support by CONICYT/FONDECYT/REGULAR/FOLIO 1200369, PCI CONICYT Instituto Max Planck For Terrestrial Microbiology Marburg MPG190003 and the Max-Planck Society. P. W. A. acknowledges Natural Sciences and Engineering Research Council (NSERC) of Canada, the Canada Research Chairs, Compute Canada, and CANARIE for financial and computational support. F. H. Z. acknowledges financial support from FWO, NSERC and Queen's University Research Initiation Grant.

CONFLICT OF INTEREST

There is no conflict of interest.

ORCID

Toon Verstraelen https://orcid.org/0000-0001-9288-5608
Esteban Vöhringer-Martinez https://orcid.org/0000-0003-1785-4558
Paul W. Ayers https://orcid.org/0000-0003-2605-3883
Farnaz Heidar-Zadeh https://orcid.org/0000-0002-2069-050X

REFERENCES

[1] T. Verstraelen, P. Tecmer, F. Heidar-Zadeh, C. E. González-Espinoza, M. Chan, T. D. Kim, et al. *HORTON 2.1.1*, **2017**, https://theochem.github.com/horton/.
[2] F. Heidar-Zadeh, M. Richer, S. Fias, R. A. Miranda-Quintana, M. Chan, M. Franco-Perez, et al., *Chem. Phys. Lett.* **2016**, *660*, 307.
[3] N. M. O'Boyle, M. Banck, C. A. James, C. Morley, T. Vandermeersch, G. R. Hutchison, *J. Cheminform.* **2011**, *3*(1), 33.
[4] Landrum G, *RDKit: Open-source cheminformatics*. https://www.rdkit.org/.
[5] N. M. O'Boyle, A. L. Tenderholt, K. M. Langner, *J. Comput. Chem.* **2008**, *29*(5), 839. https://doi.org/10.1002/jcc.20823.
[6] A. H. Larsen, J. J. Mortensen, J. Blomqvist, I. E. Castelli, R. Christensen, M. Dułak, et al., *J. Phys.: Condens. Matter* **2017**, *29*(27), 273002. http://stacks.iop.org/0953-8984/29/i=27/a=273002.
[7] T. Lu, F. Chen, *J. Comput. Chem.* **2012**, *33*(5), 580. https://doi.org/10.1002/jcc.22885.

[8] R. T. McGibbon, K. A. Beauchamp, M. P. Harrigan, C. Klein, J. M. Swails, C. X. Hernández, et al., *Biophys. J.* **2015**, *109*(8), 1528.
[9] N. Michaud-Agrawal, E. J. Denning, T. B. Woolf, O. Beckstein, *J. Comput. Chem.* **2011**, *32*(10), 2319.
[10] R. J. Gowers, M. Linke, J. Barnoud, T. J. E. Reddy, M. N. Melo, S. L. Seyler, et al. *Proceedings of the 15th Python in Science Conference* (Eds: S. Benthall, S. Rostrup), **2016**. p. 98.
[11] M. J. Frisch, G. W. Trucks, H. B. Schlegel, G. E. Scuseria, M. A. Robb, J. R. Cheeseman, et al., *Gaussian 16 Revision C.01*, Gaussian Inc, Wallingford, CT **2016**.
[12] G. Schaftenaar, E. Vlieg, G. Vriend, *J. Comput.-Aided Mol. Des.* **2017**, *31*(9), 789. https://doi.org/10.1007/s10822-017-0042-5.
[13] H. J. Werner, P. J. Knowles, G. Knizia, F. R. Manby, M. Schütz, *WIREs Comput. Mol. Sci.* **2012**, *2*, 242.
[14] *To learn about QCSchema, please refer to its website*. https://molssi-qc-schema.readthedocs.io/en/latest/index.html.
[15] G. M. J. Barca, C. Bertoni, L. Carrington, D. Datta, N. De Silva, J. E. Deustua, et al., *J. Chem. Phys.* **2020**, *152*(15), 154102. https://doi.org/10.1063/5.0005188.
[16] F. Neese, *WIREs Comput. Mol. Sci.* **2018**, *8*(1), e1327.
[17] Shao Y, Gan Z, Epifanovsky E, Gilbert ATB, Wormit M, Kussmann J, Lange AW, Behn A, Deng J, Feng X, Ghosh D, Goldey M, Horn PR, Jacobson LD, Kaliman I, Khaliullin RZ, Kuś T, Landau A, Liu J, Proynov EI, Rhee YM, Richard RM, Rohrdanz MA, Steele RP, Sundstrom EJ, Woodcock III HL, Zimmerman PM, Zuev D, Albrecht B, Alguire E, Austin B, Beran GJO, Bernard YA, Berquist E, Brandhorst K, Bravaya KB, Brown ST, Casanova D, Chang CM, Chen Y, Chien SH, Closser KD, Crittenden DL, Diedenhofen M, DiStasio Jr RA, Do H, Dutoi AD, Edgar RG, Fatehi S, Fusti-Molnar L, Ghysels A, Golubeva-Zadorozhnaya A, Gomes J, Hanson-Heine MWD, Harbach PHP, Hauser AW, Hohenstein EG, Holden ZC, Jagau TC, Ji H, Kaduk B, Khistyaev K, Kim J, Kim J, King RA, Klunzinger P, Kosenkov D, Kowalczyk T, Krauter CM, Lao KU, Laurent AD, Lawler KV, Levchenko SV, Lin CY, Liu F, Livshits E, Lochan RC, Luenser A, Manohar P, Manzer SF, Mao SP, Mardirossian N, Marenich AV, Maurer SA, Mayhall NJ, Neuscamman E, Oana CM, Olivares-Amaya R, O'Neill DP, Parkhill JA, Perrine TM, Peverati R, Prociuk A, Rehn DR, Rosta E, Russ NJ, Sharada SM, Sharma S, Small DW, Sodt A, Stein T, Stück D, Su YC, Thom AJW, Tsuchimochi T, Vanovschi V, Vogt L, Vydrov O, Wang T, Watson MA, Wenzel J, White A, Williams CF, Yang J, Yeganeh S, Yost SR, You ZQ, Zhang IY, Zhang X, Zhao Y, Brooks BR, Chan GKL, Chipman DM, Cramer CJ, Goddard III WA, Gordon MS, Hehre WJ, Klamt A, Schaefer III HF, Schmidt MW, Sherrill CD, Truhlar DG, Warshel A, Xu X, Aspuru-Guzik A, Baer R, Bell AT, Besley NA, Chai JD, Dreuw A, Dunietz BD, Furlani TR, Gwaltney SR, Hsu CP, Jung Y, Kong J, Lambrecht DS, Liang WZ, Ochsenfeld C, Rassolov VA, Slipchenko LV, Subotnik JE, van Voorhis T, Herbert JM, Krylov AI, Gill PMW, Head-Gordon M . *Mol. Phys.* **2015**;*113*:184–215.
[18] T. D. Kühne, M. Iannuzzi, M. Del Ben, V. V. Rybkin, P. Seewald, F. Stein, et al., *J. Chem. Phys.* **2020**, *152*(19), 194103. https://doi.org/10.1063/5.0007045.
[19] R. M. Parrish, L. A. Burns, D. G. A. Smith, A. C. Simmonett, A. E. DePrince, E. G. Hohenstein, et al., *J. Chem. Theory Comput.* **2017**, *13*(7), 3185. https://doi.org/10.1021/acs.jctc.7b00174.
[20] TURBOMOLE V7.4 2019, a development of University of Karlsruhe and Forschungszentrum Karlsruhe GmbH, 1989–2007, TURBOMOLE GmbH, since 2007, http://www.turbomole.com.
[21] D. G. A. Smith, D. Altarawy, L. A. Burns, M. Welborn, L. N. Naden, L. Ward, S. Ellis, B. P. Pritchard, T. D. Crawford. *WIREs Comput. Mol. Sci.*, e1491. https://doi.org/10.1002/wcms.1491.

How to cite this article: Verstraelen T, Adams W, Pujal L, et al. `IOData`: A python library for reading, writing, and converting computational chemistry file formats and generating input files. *J Comput Chem*. 2021;42:458–464. https://doi.org/10.1002/jcc.26468