xtb Python API Documentation

xtb

Jan 31, 2022

Contents

1 Installation **3**
- 1.1 Installation with Conda . . . 4
- 1.2 Building from Source . . . 4
  - 1.2.1 Building the Extension Module . . . 4
  - 1.2.2 Building without Upstream Dependency . . . 6

2 API Documentation **7**
- 2.1 Calculation Environment . . . 7
- 2.2 Molecular Structure Data . . . 8
- 2.3 Single Point Calculator . . . 9
- 2.4 Calculation Results . . . 11
- 2.5 Available Calculation Methods . . . 14

3 Atomic Simulation Environment **17**

4 QCSchema Integration **19**

Python Module Index **21**

Index **23**

This is the documentation of the Python API for the extended tight binding program (`xtb`). The project is hosted at GitHub.

```
>>> from xtb.interface import Calculator
>>> from xtb.utils import get_method
>>> import numpy as np
>>> numbers = np.array([8, 1, 1])
>>> positions = np.array([
... [ 0.00000000000000, 0.00000000000000,-0.73578586109551],
... [ 1.44183152868459, 0.00000000000000, 0.36789293054775],
... [-1.44183152868459, 0.00000000000000, 0.36789293054775]])
...
>>> calc = Calculator(get_method("GFN2-xTB"), numbers, positions)
>>> res = calc.singlepoint()  # energy printed is only the electronic part
   1     -5.1027888 -0.510279E+01  0.421E+00   14.83       0.0  T
   2     -5.1040645 -0.127572E-02  0.242E+00   14.55       1.0  T
   3     -5.1042978 -0.233350E-03  0.381E-01   14.33       1.0  T
   4     -5.1043581 -0.602769E-04  0.885E-02   14.48       1.0  T
   5     -5.1043609 -0.280751E-05  0.566E-02   14.43       1.0  T
   6     -5.1043628 -0.188160E-05  0.131E-03   14.45      44.1  T
   7     -5.1043628 -0.455326E-09  0.978E-04   14.45      59.1  T
   8     -5.1043628 -0.572169E-09  0.192E-05   14.45    3009.1  T
     SCC iter.                  ...        0 min,  0.022 sec
     gradient                   ...        0 min,  0.000 sec
>>> res.get_energy()
-5.070451354836705
>>> res.get_gradient()
array([[ 6.24500451e-17 -3.47909735e-17 -5.07156941e-03]
       [-1.24839222e-03  2.43536791e-17  2.53578470e-03]
       [ 1.24839222e-03  1.04372944e-17  2.53578470e-03]])
>>> res.get_charges()
array([-0.56317912  0.28158956  0.28158956])
```

CHAPTER 1

Installation

Depending on what you plan to do with `xtb-python` there are two recommended ways to install.

If you plan to use this project in your workflows, proceed with the *Installation with Conda*. If you plan to develop on this project, proceed with *Building from Source*.

Contents

- *Installation*
  - *Installation with Conda*
  - *Building from Source*
    - *Building the Extension Module*
      - *Meson cannot find xtb dependency*
      - *Dealing with Several Versions of Python*
      - *Installing in Development Mode*
      - *Helpful Tools*
    - *Building without Upstream Dependency*

For the basic functionalities the `xtb-python` project requires following packages:

```
cffi
numpy
```

Additionally the project provides a calculator implementation for ASE (see *Atomic Simulation Environment*) which becomes available if the `ase` package is installed. For integration with the QCArchive infrastructure (see *QCSchema Integration*) the `qcelemental` package is required.

Of course, the package depends on the extended tight binding program package as well, directly or indirectly. Depending on how `xtb-python` was packaged it requires an installation of `xtb` or it will be able to provide its own. For more details on the `xtb` API dependency see *Building from Source*.

1.1 Installation with Conda

For details on how to setup conda look up the conda documentation.

Installing `xtb-python` from the conda-forge channel can be achieved by adding conda-forge to your channels with:

```
conda config --add channels conda-forge
```

Once the conda-forge channel has been enabled, `xtb-python` can be installed with:

```
conda install xtb-python
```

It is possible to list all of the versions of `xtb-python` available on your platform with:

```
conda search xtb-python --channel conda-forge
```

To install the additional dependencies for ASE and QCArchive integration use

```
conda install qcelemental ase
```

1.2 Building from Source

To install `xtb-python` from source clone the repository from GitHub with

```
git clone https://github.com/grimme-lab/xtb-python
cd xtb-python
git submodule update --init
```

This will ensure that you have access to the `xtb-python` and the parent `xtb` repository, with the latter to be found in `subprojects/xtb`.

1.2.1 Building the Extension Module

To work with `xtb-python` it is necessary to build the extension to the `xtb` API first, this is accomplised by using meson and the C foreign function interface (CFFI). Following modules should be available to build this project:

```
cffi
numpy
meson  # build only
```

To install the meson build system first check your package manager for an up-to-date meson version, usually this will also install ninja as dependency. Alternatively, you can install the latest version of meson and ninja with `pip` (or `pip3` depending on your system):

```
pip install cffi numpy meson ninja
```

If you prefer `conda` as a package manage you can install meson and ninja from the conda-forge channel. Make sure to select the conda-forge channel for searching packages.

```
conda config --add channels conda-forge
conda install cffi numpy meson ninja
```

Now, setup the project by building the CFFI extension module from the `xtb` API with:

```
meson setup build --prefix=$PWD --default-library=shared
ninja -C build install
```

This step will create the CFFI extension `_libxtb` and place it in the `xtb` directory.

Meson cannot find xtb dependency

If meson cannot find your `xtb` installation check if you have `pkg-config` installed and that `xtb` can be found using

```
pkg-config xtb --print-errors
```

In case this fails ensure that the `xtb.pc` file is in a directory in the `PKG_CONFIG_PATH` and retry. For the official release tarball you possible have to edit the first line of `xtb.pc` to point to the location where you installed `xtb`:

```
--- a/lib/pkgconfig/xtb.pc
+++ b/lib/pkgconfig/xtb.pc
@@ -1,4 +1,4 @@
-prefix=/
+prefix=/absolute/path/to/xtb
 libdir=${prefix}/lib
 includedir=${prefix}/include/xtb
```

Note: Installs from conda-forge should work out-of-box.

Dealing with Several Versions of Python

If you have several versions of Python installed you can point meson with the `-Dpy=<version>` option to the correct one. Depending on your setup you have to export your compilers (`CC` and `FC`) first and set the `-Dla_backend=<name>` and `-Dopenmp=<bool>` option accordingly.

Installing in Development Mode

After creating the `_libxtb` extension, the Python module can be installed as usual with

```
pip install -e .
```

Now you are set to start using `xtb-python`. You can test your setup by opening a new Python interpreter and try to import the interface module

```
>>> import xtb.interface
```

If you also want to use extensions install with

```
pip install -e '.[ase,qcschema]'
```

Now you can test your installation with

```
pytest --pyargs xtb
```

Helpful Tools

We aim for a high quality code base and encourage substainable development models.

Please, install a linter like `flake8` or `pylint` to catch errors before they become bugs. Also, typehints are mandatory in this project, you should typecheck locally with `mypy`. A consistent coding style is enforced by using `black`, every source file should be reformatted using `black`, the only exceptions are tests.

1.2.2 Building without Upstream Dependency

For convenience we also offer a mode to work without an upstream `xtb` dependency, this can be quite handy if you also want to work on the `xtb` API itself or want to create a failsafe package that cannot break due to ABI or API incompatibilities.

Note: It is highly recommend to make yourself familiar with building `xtb` first.

For this approach we follow the same scheme as with the normal extension build. You will need the following packages installed

```
cffi
numpy
meson  # build only
```

Additionally you will need a development version of Python, for the Python headers, a Fortran and a C compiler (GCC 7 or newer or Intel 17 or newer) and a linear algebra backend (providing LAPACK and BLAS API).

We closely follow the approach from before, but we change the configuration of the extension build to

```
meson setup build --prefix=$PWD --default-library=static
ninja -C build install
```

Depending on how you acquired the project mesons wrap-tool will first need to download the `xtb` source code. Instead of dynamically depending on `xtb` the complete project will be build and included as a whole into the CFFI extension module, making your `xtb-python` effectively independent of `xtb`.

You can pass the `-Dopenmp=<bool>` and `-Dla_backend=<netlib|openblas|mkl>` in the configuration step to configure the `xtb` build. To change the compiler used export them in the environment variables `CC` and `FC`.

Tip: For more information on the build with meson, follow the guide in the `xtb` repository here.

From here you can proceed with *Installing in Development Mode*.

CHAPTER 2

API Documentation

Important: All properties exchanged with the `xtb` API are given in atomic units. For integrations with other frameworks the unit conventions might differ and require conversion.

Contents

- *API Documentation*
  - *Calculation Environment*
  - *Molecular Structure Data*
  - *Single Point Calculator*
  - *Calculation Results*
  - *Available Calculation Methods*

2.1 Calculation Environment

class `xtb.interface.`**Environment**

Wraps an API object representing a TEnvironment class in `xtb`. The API object is constructed automatically and deconstructed on garbage collection, it stores the IO configuration and the error log of the API.

All API calls require an environment object, usually this is done automatically as all other classes inherent from the calculation environment.

Example

```
>>> from xtb.libxtb import VERBOSITY_FULL
>>> from xtb.interface import Environment
>>> env = Environment()
>>> env.set_output("error.log")
>>> env.set_verbosity(VERBOSITY_FULL)
>>> if env.check != 0:
...     env.show("Error message")
...
>>> env.release_output()
```

check() → int
Check current status of calculation environment

Example

```
>>> if env.check() != 0:
...     raise XTBException("Error occured in the API")
```

get_error(*message: Optional[str] = None*) → str
Check for error messages

Example

```
>>> if env.check() != 0:
...     raise XTBException(env.get_error())
```

release_output() → None
Release output unit from this environment

set_output(*filename: str*) → None
Bind output from this environment

set_verbosity(*verbosity: int*) → None
Set verbosity of calculation output

show(*message: str*) → None
Show and empty error stack

2.2 Molecular Structure Data

class `xtb.interface.`**`Molecule`**(*numbers: numpy.ndarray, positions: numpy.ndarray, charge: Optional[float] = None, uhf: Optional[int] = None, lattice: Optional[numpy.ndarray] = None, periodic: Optional[numpy.ndarray] = None*)

Represents a wrapped TMolecule API object in `xtb`. The molecular structure data object has a fixed number of atoms and immutable atomic identifiers.

Example

```
>>> from xtb.interface import Molecule
>>> import numpy as np
>>> numbers = np.array([8, 1, 1])
>>> positions = np.array([
... [ 0.00000000000000, 0.00000000000000,-0.73578586109551],
... [ 1.44183152868459, 0.00000000000000, 0.36789293054775],
... [-1.44183152868459, 0.00000000000000, 0.36789293054775]])
...
>>> mol = Molecule(numbers, positions)
>>> len(mol)
3
>>> mol.update(np.zeros((len(mol), 3)))  # will fail nuclear fusion check
xtb.interface.XTBException: Update of molecular structure failed:
-1- xtb_api_updateMolecule: Could not update molecular structure
>>> mol.update(positions)
```

Raises

- `ValueError` – on invalid input on the Python side of the API
- `XTBException` – on errors returned from the API

`update`(*positions: numpy.ndarray*, *lattice: Optional[numpy.ndarray] = None*) → None

Update coordinates and lattice parameters, both provided in atomic units (Bohr). The lattice update is optional also for periodic structures.

Generally, only the cartesian coordinates and the lattice parameters can be updated, every other modification, regarding total charge, total spin, boundary condition, atomic types or number of atoms requires the complete reconstruction of the object.

Raises

- `ValueError` – on invalid input on the Python side of the API
- `XTBException` – on errors returned from the API, usually from nuclear fusion check

2.3 Single Point Calculator

class `xtb.interface.`**`Calculator`**(*param: xtb.interface.Param, numbers: List[int], positions: List[float], charge: Optional[float] = None, uhf: Optional[int] = None, lattice: Optional[List[float]] = None, periodic: Optional[List[bool]] = None*)

This calculator represents a calculator object in the `xtb` API and provides access to all methods implemented with a unified interface. The API object must be loaded with a parametrisation before it can be used in any other API request.

The parametrisation loading is included in the initialization in this class, which has the advantage that all API functionality is readily available, the downside is that a calculator object on the Python side can only carry one distinct parametrisation, which is not allowed to change.

Examples

```
>>> from xtb.libxtb import VERBOSITY_MINIMAL
>>> from xtb.interface import Calculator, Param
>>> import numpy as np
>>> numbers = np.array([8, 1, 1])
>>> positions = np.array([
... [ 0.00000000000000, 0.00000000000000,-0.73578586109551],
... [ 1.44183152868459, 0.00000000000000, 0.36789293054775],
... [-1.44183152868459, 0.00000000000000, 0.36789293054775]])
...
>>> calc = Calculator(Param.GFN2xTB, numbers, positions)
>>> calc.set_verbosity(VERBOSITY_MINIMAL)
>>> res = calc.singlepoint()  # energy printed is only the electronic part
   1     -5.1027888 -0.510279E+01  0.421E+00   14.83       0.0  T
   2     -5.1040645 -0.127572E-02  0.242E+00   14.55       1.0  T
   3     -5.1042978 -0.233350E-03  0.381E-01   14.33       1.0  T
   4     -5.1043581 -0.602769E-04  0.885E-02   14.48       1.0  T
   5     -5.1043609 -0.280751E-05  0.566E-02   14.43       1.0  T
   6     -5.1043628 -0.188160E-05  0.131E-03   14.45      44.1  T
   7     -5.1043628 -0.455326E-09  0.978E-04   14.45      59.1  T
   8     -5.1043628 -0.572169E-09  0.192E-05   14.45    3009.1  T
     SCC iter.                  ...        0 min,  0.022 sec
     gradient                   ...        0 min,  0.000 sec
>>> res.get_energy()
-5.070451354836705
>>> res.get_gradient()
[[ 6.24500451e-17 -3.47909735e-17 -5.07156941e-03]
 [-1.24839222e-03  2.43536791e-17  2.53578470e-03]
 [ 1.24839222e-03  1.04372944e-17  2.53578470e-03]]
```

Raises `XTBException` – on errors encountered in API or while performing calculations

`release_external_charges``()` → None
Unset external point charge field

`set_accuracy``(`*accuracy: float*`)` → None
Set numerical accuracy for calculation, ranges from 1000 to 0.0001, values outside this range will be cutted with warning placed in the error log, which can be retrieved by get_error() but will not trigger check().

Example

```
>>> calc.set_accuracy(1.0)
```

`set_electronic_temperature``(`*etemp: int*`)` → None
Set electronic temperature in K for tight binding Hamiltonians, values smaller or equal to zero will be silently ignored by the API.

Example

```
>>> calc.set_electronic_temperature(300.0)
```

`set_external_charges``(`*numbers: numpy.ndarray*, *charges: numpy.ndarray*, *positions: numpy.ndarray*`)` → None
Set an external point charge field

set_max_iterations(*maxiter: int*) → None

Set maximum number of iterations for self-consistent charge methods, values smaller than one will be silently ignored by the API. Failing to converge in a given number of cycles is not necessarily reported as an error by the API.

Example

```
>>> calc.set_max_iterations(100)
```

set_solvent(*solvent: Optional[xtb.interface.Solvent] = None*) → None

Add/Remove a solvation model to/from calculator

Example

```
>>> from xtb.utils import get_solvent, Solvent
...
>>> calc.set_solvent(Solvent.h2o)  # Set solvent to water with enumerator
>>> calc.set_solvent()  # Release solvent again
>>> calc.set_solvent(get_solvent("CHCl3"))  # Find correct enumerator
```

singlepoint(*res: Optional[xtb.interface.Results] = None, copy: bool = False*) → xtb.interface.Results

Perform singlepoint calculation, note that the a previous result is overwritten by default.

Example

```
>>> res = calc.singlepoint()
>>> res = calc.singlepoint(res)
>>> calc.singlepoint(res)  # equivalent to the above
>>> new = calc.singlepoint(res, copy=True)
```

2.4 Calculation Results

class xtb.interface.**Results**(*res: Union[xtb.interface.Molecule, Results]*)

Holds `xtb` API object containing results from a single point calculation. It can be queried for indiviual properties or used to restart calculations. Note that results from different methods are generally incompatible, the API tries to be as clever as possible about this and will usually automatically reallocate missmatched results objects as necessary.

The results objects is connected to its own, independent environment, giving it its own error stack and IO infrastructure.

Example

```
>>> from xtb.libxtb import VERBOSITY_MINIMAL
>>> from xtb.interface import Calculator, Param
>>> import numpy as np
>>> numbers = np.array([8, 1, 1])
>>> positions = np.array([
```

(continues on next page)

(continued from previous page)

```
... [ 0.00000000000000, 0.00000000000000,-0.73578586109551],
... [ 1.44183152868459, 0.00000000000000, 0.36789293054775],
... [-1.44183152868459, 0.00000000000000, 0.36789293054775]])
...
>>> calc = Calculator(Param.GFN2xTB, numbers, positions)
>>> calc.set_verbosity(VERBOSITY_MINIMAL)
>>> res = calc.singlepoint()  # energy printed is only the electronic part
   1     -5.1027888 -0.510279E+01  0.421E+00   14.83       0.0  T
   2     -5.1040645 -0.127572E-02  0.242E+00   14.55       1.0  T
   3     -5.1042978 -0.233350E-03  0.381E-01   14.33       1.0  T
   4     -5.1043581 -0.602769E-04  0.885E-02   14.48       1.0  T
   5     -5.1043609 -0.280751E-05  0.566E-02   14.43       1.0  T
   6     -5.1043628 -0.188160E-05  0.131E-03   14.45      44.1  T
   7     -5.1043628 -0.455326E-09  0.978E-04   14.45      59.1  T
   8     -5.1043628 -0.572169E-09  0.192E-05   14.45    3009.1  T
     SCC iter.                  ...        0 min,  0.022 sec
     gradient                   ...        0 min,  0.000 sec
>>> res.get_energy()
-5.070451354836705
>>> res.get_gradient()
[[ 6.24500451e-17 -3.47909735e-17 -5.07156941e-03]
 [-1.24839222e-03  2.43536791e-17  2.53578470e-03]
 [ 1.24839222e-03  1.04372944e-17  2.53578470e-03]]
>>> res = calc.singlepoint(res)
   1     -5.1043628 -0.510436E+01  0.898E-08   14.45       0.0  T
   2     -5.1043628 -0.266454E-14  0.436E-08   14.45  100000.0  T
   3     -5.1043628  0.177636E-14  0.137E-08   14.45  100000.0  T
     SCC iter.                  ...        0 min,  0.001 sec
     gradient                   ...        0 min,  0.000 sec
>>> res.get_charges()
[-0.56317912  0.28158956  0.28158956]
```

Raises `XTBException` – in case the requested property is not present in the results object

get_bond_orders() → numpy.ndarray

Query singlepoint results object for bond orders

Example

```
>>> res.get_bond_orders()
[[0.00000000e+00 9.20433501e-01 9.20433501e-01]
 [9.20433501e-01 0.00000000e+00 2.74039053e-04]
 [9.20433501e-01 2.74039053e-04 0.00000000e+00]]
```

get_charges() → numpy.ndarray

Query singlepoint results object for partial charges in e

Example

```
>>> get_charges()
[-0.56317913  0.28158957  0.28158957]
```

get_dipole() → numpy.ndarray

Query singlepoint results object for dipole in e·Bohr

Example

```
>>> get_dipole()
[-4.44089210e-16  1.44419023e-16  8.89047667e-01]
```

get_energy() → float

Query singlepoint results object for energy in Hartree

Example

```
>>> res.get_energy()
-5.070451354836705
```

get_gradient() → numpy.ndarray

Query singlepoint results object for gradient in Hartree/Bohr

Example

```
>>> res.get_gradient()
[[ 6.24500451e-17 -3.47909735e-17 -5.07156941e-03]
 [-1.24839222e-03  2.43536791e-17  2.53578470e-03]
 [ 1.24839222e-03  1.04372944e-17  2.53578470e-03]]
```

get_number_of_orbitals() → int

Query singlepoint results object for the number of basis functions

Example

```
>>> res.get_number_of_orbitals()
6
```

get_orbital_coefficients() → numpy.ndarray

Query singlepoint results object for orbital coefficients

Example

```
>>> res.get_orbital_coefficients()
array([[-7.94626768e-01,  6.38378239e-16,  4.52990407e-01,
        -6.38746369e-16, -8.35495085e-01, -4.44089210e-16],
       [ 2.77555756e-17, -6.97332245e-01,  7.49400542e-16,
         1.88136491e-17,  7.21644966e-16, -9.60006511e-01],
       [ 2.17336312e-16, -1.08051945e-16, -1.11598977e-15,
        -1.00000000e+00,  5.74153329e-17,  3.30330107e-17],
       [-8.67578876e-02, -9.71445147e-16, -8.05763104e-01,
         7.71702239e-16, -7.18690020e-01, -4.71844785e-16],
       [-1.84540457e-01, -3.54572323e-01, -2.39090946e-01,
         2.87533552e-16,  7.68757806e-01,  9.02845514e-01],
```

(continues on next page)

(continued from previous page)

```
        [-1.84540457e-01,  3.54572323e-01, -2.39090946e-01,
          2.01021058e-16,  7.68757806e-01, -9.02845514e-01]])
```

get_orbital_eigenvalues() → numpy.ndarray

Query singlepoint results object for orbital energies in Hartree

Example

```
>>> res.get_orbital_eigenvalues()
array([-0.68087967, -0.56667693, -0.51373083, -0.44710101,  0.08394016,
        0.24142397])
```

get_orbital_occupations() → numpy.ndarray

Query singlepoint results object for occupation numbers

Example

```
>>> res.get_orbital_occupations()
array([2., 2., 2., 2., 0., 0.])
```

get_virial() → numpy.ndarray

Query singlepoint results object for virial given in Hartree

Example

```
>>> res.get_virial()
[[ 1.43012837e-02  3.43893209e-17 -1.86809511e-16]
 [ 0.00000000e+00  0.00000000e+00  0.00000000e+00]
 [ 1.02348685e-16  1.46994821e-17  3.82414977e-02]]
```

2.5 Available Calculation Methods

class xtb.interface.**Param**

Possible parametrisations for the Calculator class

GFN0xTB = 3

Experimental non-self-consistent extended tight binding Hamiltonian using classical electronegativity equilibration electrostatics and extended Hückel Hamiltonian.

Geometry, frequency and non-covalent interactions parametrisation for elements up to Z=86.

Requires the param_gfn0-xtb.txt parameter file in the `XTBPATH` environment variable to load!

See: P. Pracht, E. Caldeweyher, S. Ehlert, S. Grimme, ChemRxiv, 2019, preprint. DOI: 10.26434/chemrxiv.8326202.v1

GFN1xTB = 2

Self-consistent extended tight binding Hamiltonian with isotropic second order electrostatic contributions and third order on-site contributions.

Geometry, frequency and non-covalent interactions parametrisation for elements up to Z=86.

Cite as: S. Grimme, C. Bannwarth, P. Shushkov, *J. Chem. Theory Comput.*, 2017, 13, 1989-2009. DOI: 10.1021/acs.jctc.7b00118

GFN2xTB = 1

Self-consistent extended tight binding Hamiltonian with anisotropic second order electrostatic contributions, third order on-site contributions and self-consistent D4 dispersion.

Geometry, frequency and non-covalent interactions parametrisation for elements up to Z=86.

Cite as: C. Bannwarth, S. Ehlert and S. Grimme., J. Chem. Theory Comput., 2019, 15, 1652-1671. DOI: 10.1021/acs.jctc.8b01176

GFNFF = 4

General force field parametrized for geometry, frequency and non-covalent interactions up to Z=86.

`xtb` API support is currently experimental.

Cite as: S. Spicher and S. Grimme, Angew. Chem. Int. Ed., 2020, 59, 15665–15673. DOI: 10.1002/anie.202004239

IPEAxTB = 5

Special parametrisation for the GFN1-xTB Hamiltonian to improve the description of vertical ionisation potentials and electron affinities. Uses additional diffuse s-functions on light main group elements. Parametrised up to Z=86.

Cite as: V. Asgeirsson, C. Bauer and S. Grimme, Chem. Sci., 2017, 8, 4879. DOI: **‘10.1039/c7sc00601b <https://dx.doi.org/10.1039/c7sc00601b‘_**

`utils.`**`get_method`**`()` → Optional[xtb.interface.Param]

Return the correct parameter enumerator for a string input.

Example

```
>>> get_method('GFN2-xTB')
<Param.GFN2xTB: 1>
>>> get_method('gfn2xtb')
<Param.GFN2xTB: 1>
>>> get_method('GFN-xTB') is None
True
>>> get_method('GFN1-xTB') is None
False
```

CHAPTER 3

Atomic Simulation Environment

ASE calculator implementation for the `xtb` program.

This module provides the basic single point calculator implementation to integrate the `xtb` API into existing ASE workflows.

Supported properties by this calculator are:

- energy (free_energy)
- forces
- stress (GFN0-xTB only)
- dipole
- charges

Example

```
>>> from ase.build import molecule
>>> from xtb.ase.calculator import XTB
>>> atoms = molecule('H2O')
>>> atoms.calc = XTB(method="GFN2-xTB")
>>> atoms.get_potential_energy()
-137.9677758730299
>>> atoms.get_forces()
[[ 1.30837706e-16  1.07043680e-15 -7.49514699e-01]
 [-1.05862195e-16 -1.53501989e-01  3.74757349e-01]
 [-2.49755108e-17  1.53501989e-01  3.74757349e-01]]
```

Supported keywords are

Keyword	Default	Description
method	“GFN2-xTB”	Underlying method for energy and forces
accuracy	1.0	Numerical accuracy of the calculation
electronic_temperature	300.0	Electronic temperatur for TB methods
max_iterations	250	Iterations for self-consistent evaluation
solvent	“none”	GBSA implicit solvent model
cache_api	True	Reuse generate API objects (recommended)

CHAPTER 4

QCSchema Integration

Integration with the QCArchive infrastructure.

This module provides a way to translate QCSchema or QCElemental Atomic Input into a format understandable by the `xtb` API which in turn provides the calculation results in a QCSchema compatible format.

The `xtb` model supports any method accepted by `xtb.utils.get_method`.

Supported keywords are

Keyword	Default	Description
accuracy	1.0	Numerical accuracy of the calculation
electronic_temperature	300.0	Electronic temperatur for TB methods
max_iterations	250	Iterations for self-consistent evaluation
solvent	“none”	GBSA implicit solvent model

`xtb.qcschema.harness.`**`run_qcschema`**(*input_data: Union[dict, qcelemental.models.results.AtomicInput]*) → qcelemental.models.results.AtomicResult

Perform a calculation based on an atomic input model.

Example

```
>>> from xtb.qcschema.harness import run_qcschema
>>> import qcelemental as qcel
>>> atomic_input = qcel.models.AtomicInput(
...     molecule = qcel.models.Molecule(
...         symbols = ["O", "H", "H"],
...         geometry = [
...             0.00000000000000,  0.00000000000000, -0.73578586109551,
...             1.44183152868459,  0.00000000000000,  0.36789293054775,
...            -1.44183152868459,  0.00000000000000,  0.36789293054775
...         ],
```

(continues on next page)

(continued from previous page)

```
...     ),
...     driver = "energy",
...     model = {
...         "method": "GFN2-xTB",
...     },
...     keywords = {
...         "accuracy": 1.0,
...         "max_iterations": 50,
...     },
... )
...
>>> atomic_result = run_qcschema(atomic_input)
>>> atomic_result.return_result
-5.070451354848316
```

Python Module Index

x

`xtb.ase`, 15
`xtb.ase.calculator`, 17
`xtb.interface`, 6
`xtb.qcschema.harness`, 19

Index

C

`Calculator` (*class in xtb.interface*), 9
`check()` (*xtb.interface.Environment method*), 8

E

`Environment` (*class in xtb.interface*), 7

G

`get_bond_orders()` (*xtb.interface.Results method*), 12
`get_charges()` (*xtb.interface.Results method*), 12
`get_dipole()` (*xtb.interface.Results method*), 12
`get_energy()` (*xtb.interface.Results method*), 13
`get_error()` (*xtb.interface.Environment method*), 8
`get_gradient()` (*xtb.interface.Results method*), 13
`get_method()` (*xtb.utils method*), 15
`get_number_of_orbitals()` (*xtb.interface.Results method*), 13
`get_orbital_coefficients()` (*xtb.interface.Results method*), 13
`get_orbital_eigenvalues()` (*xtb.interface.Results method*), 14
`get_orbital_occupations()` (*xtb.interface.Results method*), 14
`get_virial()` (*xtb.interface.Results method*), 14
`GFN0xTB` (*xtb.interface.Param attribute*), 14
`GFN1xTB` (*xtb.interface.Param attribute*), 14
`GFN2xTB` (*xtb.interface.Param attribute*), 15
`GFNFF` (*xtb.interface.Param attribute*), 15

I

`IPEAxTB` (*xtb.interface.Param attribute*), 15

M

`Molecule` (*class in xtb.interface*), 8

P

`Param` (*class in xtb.interface*), 14

R

`release_external_charges()` (*xtb.interface.Calculator method*), 10
`release_output()` (*xtb.interface.Environment method*), 8
`Results` (*class in xtb.interface*), 11
`run_qcschema()` (*in module xtb.qcschema.harness*), 19

S

`set_accuracy()` (*xtb.interface.Calculator method*), 10
`set_electronic_temperature()` (*xtb.interface.Calculator method*), 10
`set_external_charges()` (*xtb.interface.Calculator method*), 10
`set_max_iterations()` (*xtb.interface.Calculator method*), 11
`set_output()` (*xtb.interface.Environment method*), 8
`set_solvent()` (*xtb.interface.Calculator method*), 11
`set_verbosity()` (*xtb.interface.Environment method*), 8
`show()` (*xtb.interface.Environment method*), 8
`singlepoint()` (*xtb.interface.Calculator method*), 11

U

`update()` (*xtb.interface.Molecule method*), 9

X

`xtb.ase` (*module*), 15
`xtb.ase.calculator` (*module*), 17
`xtb.interface` (*module*), 6
`xtb.qcschema.harness` (*module*), 19