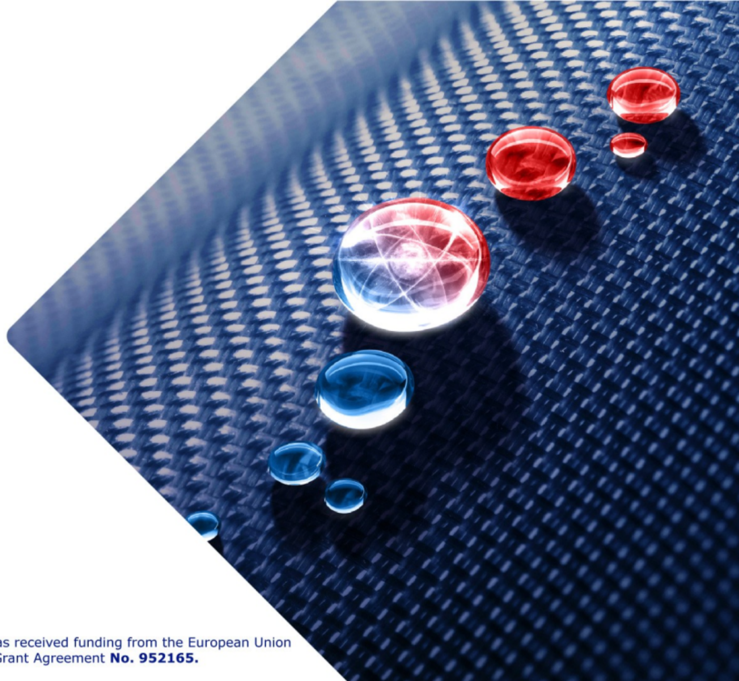
Targeting Real chemical accuracy at the EXascale

TREXIO

Evgeny Posenitskiy, Anthony Scemama

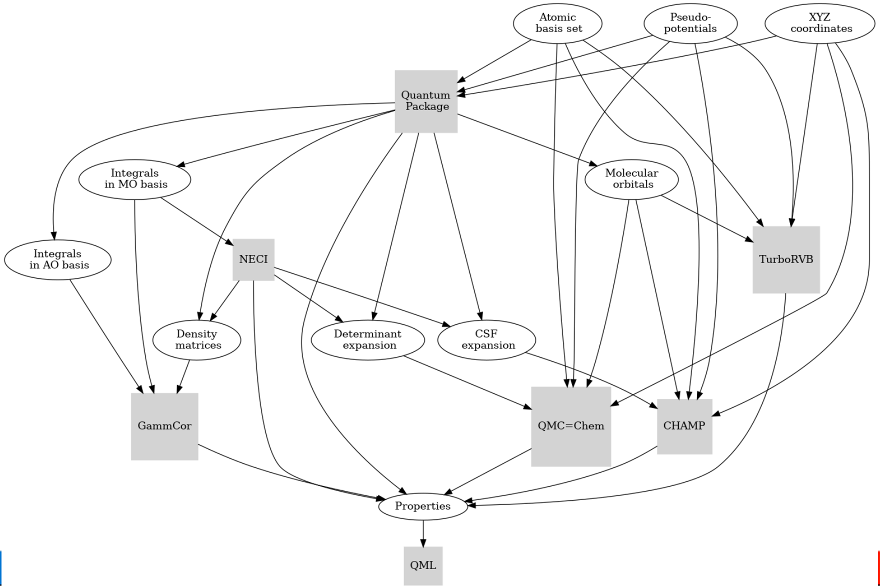
February 28, 2022

CNRS | LCPQ



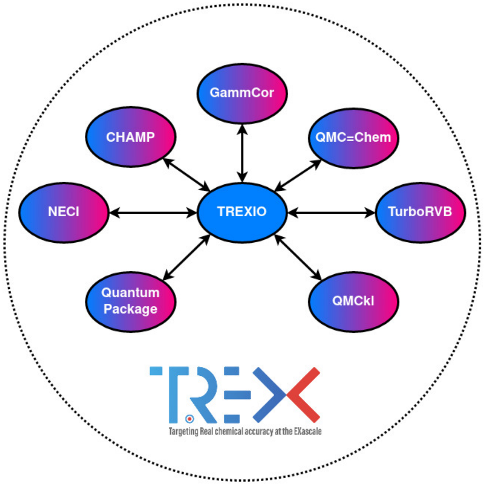
Targeting Real Chemical Accuracy at the Exascale project has received funding from the European Union Horizoon 2020 research and innovation programme under Grant Agreement **No. 952165.**

Motivation


Atomic basis set
Pseudo-potentials
XYZ coordinates
Quantum Package
Integrals in MO basis
Molecular orbitals
Integrals in AO basis
NECI
TurboRVB
Density matrices
Determinant expansion
CSF expansion
GammCor
QMC=Chem
CHAMP
Properties
QML

Motivation

TREX codes can efficiently exchange data by using a common format.



TREXIO: The TREX I/O library

TREXIO library

Front end

- User–friendly and extensible API with error handling
- Source code in pure **C** for the best performance and portability
- Binding in **Fortran** (using `ISO_C_BINDING`)
- Binding in **Python** (using `SWIG`)

Back ends

- **TREXIO_HDF5**: efficient I/O, requires installation of the HDF5 library
- **TREXIO_TEXT**: debugging, fallback when HDF5 cannot be installed

The **TREXIO source code and documentation**[1] **are automatically generated** from the `Emacs` org-mode files.

[1] `https://trex-coe.github.io/trexio`

Naming convention

- trexio_open
- trexio_write_<group>_<data>[_<precision>]
- trexio_read_<group>_<data>[_<precision>]
- trexio_has_<group>_<data>
- trexio_close

The <**precision**> = **32/64** suffix is optional.
It can be provided for numerical (integer and float) data types.

Naming convention

- trexio_open
- trexio_write[_safe]_<group>_<data>[_<precision>]
- trexio_read[_safe]_<group>_<data>[_<precision>]
- trexio_has_<group>_<data>
- trexio_close
- **trexio_delete_<group> ('u' mode in v.2.2)**

The **safe** suffix is optional.
Safe functions contain one additional argument indicating the expected number of array elements to read/write.

Naming convention

- trexio_open
- trexio_write[_safe]_<group>_<data>[_<precision>]
- trexio_read[_safe]_<group>_<data>[_<precision>]
- trexio_has_<group>_<data>
- trexio_close
- **trexio_delete_<group> ('u' mode in v.2.2)**

Example

<group>

- nucleus

  <data>

  - num
  - charge
  - coord
  - label
  - point_group
  - repulsion

TREXIO configuration file (trex.json)

group:

```
      data              :  [ data type  ,  [ list of dimensions ]        ]
```

```
"nucleus": {
    "num"         : [ "dim"   , []                          ],
    "charge"      : [ "float" , ["nucleus.num"]             ],
    "coord"       : [ "float" , ["nucleus.num", "3" ]       ],
    "label"       : [ "str"   , ["nucleus.num"]             ],
    "point_group" : [ "str"   , []                          ],
    "repulsion"   : [ "float" , []                          ]
    }
```

Order of dimensions   **trex.org:** column-major (Fortran)   **trex.json:** row-major (C)

Currently supported data types

- dim
- int
- index
- float
- float sparse
- str

Values of **index** type are internally shifted by one depending on the used binding. This is required since arrays are 1-based in Fortran and 0-based in C/Python.

Values of **dim** type are strictly positive integers (dim stands for dimensioning).

Sparse data

Sparse tensors are stored using **coordinate list**[2] representation, e.g.

```
    (index1, index2, index3, index4, value)
```

for 4–index sparse tensors like two-electron integrals.

```
"eri":["float sparse",["mo.num","mo.num","mo.num","mo.num"]]
```

Indices and values are provided by the TREXIO API separately, namely one has to call

```
trexio_read_mo_2e_int_eri_index(...)
trexio_read_mo_2e_int_eri_value(...)
```

to obtain `(index1, index2, index3, index4)` and `values`, respectively.

The **mo.num** value is used to compress the storage of indices in the file, e.g. if `mo.num < UINT8_MAX (255)` then indices can be stored as unsigned 8-bit integers.

[2] https://en.wikipedia.org/wiki/Sparse_matrix

Take-home messages

1. TREXIO files are immutable, i.e. one cannot overwrite existing data. Provide **unsafe ('u')** mode to `trexio_open` as a workaround (discouraged)
2. All functions (except for `trexio_open`) in C/Fortran return `trexio_exit_code`

```
use trexio
integer(trexio_exit_code) :: rc
integer(trexio_t)         :: fhandle

fhandle = trexio_open(file_name, 'w', TREXIO_HDF5, rc)
call trexio_assert(rc, TREXIO_SUCCESS)
rc = trexio_write_nucleus_num(fhandle, 12)
call trexio_assert(rc, TREXIO_SUCCESS)
rc = trexio_close(fhandle)
call trexio_assert(rc, TREXIO_SUCCESS)
```

Take-home messages

1. TREXIO files are immutable, i.e. one cannot overwrite existing data. Provide **unsafe ('u')** mode to `trexio_open` as a workaround (discouraged)
2. All functions (except for `trexio_open`) in C/Fortran return `trexio_exit_code`

```
import trexio

fhandle = trexio.File(file_name, 'w', trexio.TREXIO_HDF5)
assert fhandle.exists

trexio.write_nucleus_num(fhandle, 12)
assert trexio.has_nucleus_num(fhandle)
```

Take-home messages

1. TREXIO files are immutable, i.e. one cannot overwrite existing data. Provide **unsafe ('u')** mode to `trexio_open` as a workaround (discouraged)
2. All functions (except for `trexio_open`) in C/Fortran return `trexio_exit_code`
3. Dimensioning (dim) variables should be written **before** arrays associated with them. Otherwise, `trexio_write_<array>` will fail
4. Sparse data I/O can be done block-wise when it is too big to fit into memory. See `offset_file` and `buffer_size` arguments of the corresponding functions

TREXIO: The TREX I/O format for electronic wave functions

Currently supported groups (from the trex.org file)

- electron
- nucleus
- ecp
- basis
- rdm
- metadata
- ao
- ao_1e_int
- ao_2e_int
- mo
- mo_1e_int
- mo_2e_int

More details in the TREXIO documentation.[3]

[3] https://trex-coe.github.io/trexio/trex.html

Some enhancements compared to existing wave function formats

- Fully self-consistent, i.e. no external (code-specific) knowledge is required
- Exhaustive list of normalization parameters to cover existing ambiguities
- AOs support for both Cartesian (alphabetical ordering) and spherical ($0,\ldots,\pm m$) representations
- Reduced density matrices and 2-electron integrals are stored in sparse data format (coordinate list) similar to FCIDUMP

More details in the TREXIO documentation.[4]

[4] `https://trex-coe.github.io/trexio/trex.html`

Effective Core Potentials (ECP)

Challenging to define general representation for TREXIO:

- different number of ECP functions per L;
- different number of L per atom.

Example: ECP for C atom in GAMESS input format

```
C-ccECP GEN 2 1
3
4.00000      1      8.35974
33.43895     3      4.48362
-19.17537    2      3.93831
1
22.55164     2      5.02992
```

Effective Core Potentials (ECP)

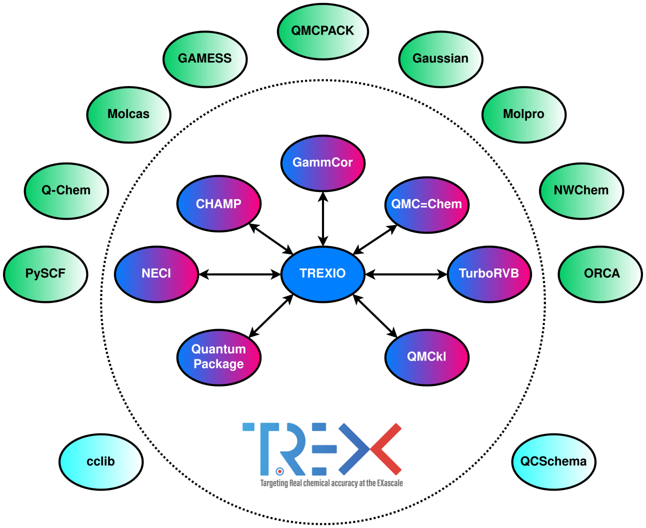
Task: write ECP for C_2 in TREXIO file
Solution: use flat arrays and mappings

```
C-ccECP GEN 2 1
3
4.00000       1       8.35974
33.43895      3       4.48362
-19.17537     2       3.93831
1
22.55164      2       5.02992
```

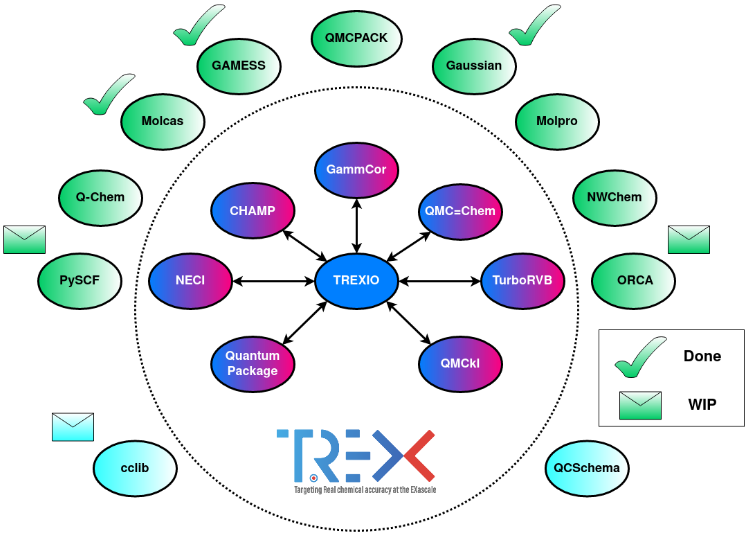
```
ecp_num = 8
ecp_max_ang_mom_plus_1 =
[ 1, 1 ]
ecp_nucleus_index =
[ 0, 0, 0, 0,
  1, 1, 1, 1 ]
ecp_ang_mom =
[ 1, 1, 1, 0,
  1, 1, 1, 0 ]
ecp_coefficient =
[ 4.00, 33.44, -19.18, 22.55,
  4.00, 33.44, -19.18, 22.55 ]
```

TREXIO tools: collaborative effort

TREXIO tools



TREXIO tools



TREXIO tools: live demo

Conclusion

Perspectives

- Storage of the multi-configurational wave functions (CI determinants/CSF)
- More converters for external codes in `trexio_tools`
- Advanced compression of the HDF5 files (collaboration with UVSQ)
- Packaging (conda, Spack, Guix)

TREXIO links

- GitHub repository : `https://github.com/trex-coe/trexio`
- Stable release v.2.1.0 : `https://github.com/TREX-CoE/trexio/releases/tag/v2.1.0`
- Source code documentation : `https://trex-coe.github.io/trexio`

Thank you for your attention!