Note

The content of this guide has not been updated since 2019-09-10. Since then a lot has happened. For example, xtb went open source and is now available on GitHub. This is also true for enso, which can also be obtained from GitHub. Furthermore, I am not maintaining any installation on the RWTH cluster, hence I cannot recommend the setup for the outdated bundle given below. This guide would have to be adjusted for any of these changes.

A more general guide on how to set-up xtb and crest for the use with runxtb can be found here. This guide is maintained only as a starting point for computations using enso, although the instructions in the xtb manual are by far better resources.

(Martin; 2020-04-09; wrapper version 0.4.0)

How to use enso (and crest, and xtb)

Setup xtb, crest, and enso

If you have access to the rwth0425 computation project on CLAIX18, then you can skip most of this part, as this installation is ready to go after initialising the software from this project (see section at the end). If you want to know more, then read on.

Bundle the programs

The most recent versions this guide is based on are xtb 6.2 RC2, crest 2.7.1, and enso 1.2. I have combined these into one loadable module in the following way:

1. Obtain the distributed archives from the Mulliken Center for Theoretical Chemistry. Currently this should be the following three files: `xtb-190806.tar.xz`, `crest_2.7.1.zip`, `enso.tgz`.
2. Unpack the xtb archive. This will yield the following directories: `bin`, `include`, `lib`, `man`, `python`. There are two configuration files, too, which you can ignore for the moment: `Config_xtb_env.bash`, `Config_xtb_env.csh`.
3. Unpack the crest archive, which only contains crest, and copy it to the `bin` directory. Make sure the executable bit is set.
4. In the root directory, I created the folder `scripts` for enso. This is basically how the setup used to be with xtb, so I kept that part. In principle you could also use a different name or the `bin` directory. Extract the enso archive to this folder. For easier access I have created a symbolic link for enso:

   ```
   ln -s enso.py enso
   ```

5. Now you have to make the software accessible to run. If you are using runxtb.bash (from this repository) follow the installation guide there.
   If you want to run it only interactively, **source** the appropriate configuration file `Config_xtb_env.*sh` from your respective shell configuration. This will set or append the environment variables `PATH`, `XTBPATH`, `MANPATH`, `LD_LIBRARY_PATH`, `PYTHONPATH`. You need to edit this file to make sure to also include the directory created in step 4 in `PATH`.
   If you are using modules, then you probably know how to create one yourself and how to load it, but examples are available with this repository.

Additional notes on enso

1. You will need the `.ensorc` file with standard settings in your home (`$HOME`) directory. Alternatively you can put it in your working directory. (If you are using the bundled versions of me, see above, you will find it in `$XTBPATH/scripts`.) You will need to edit this with settings appropriate for your system, especially the paths to the executables and external programs.
2. You need a working installation of either ORCA or Turbomole for enso. If you use runxtb.bash with modules, you can simply add these to be loaded at runtime. (Note: I have not managed to get it to work with turbomole correctly.)

3. In case you use Turbomole, you also need the `.cefinerc` file in your home/working directory. (If you are using the bundled versions of me, you will find it in `$XTBPATH/scripts`.) You also need to set `PARA_ARCH=SMP` and `PARNODES=<INT>` and export them, for example, include the following line in `.bashrc`:

   ```
   PARA_ARCH=SMP; PARNODES=1; export PARA_ARCH PARNODES
   ```

Automatically generate NMR Spectra (run enso)

This How-to focusses on the use with this repository, but it can easily be adjusted to interactive use. It is based on a guide from an older version of enso, and it worked with what I have installed. However, there is absolutely no guarantee for correctness.

0. Generate a structure, for example butan-2-one with Open Babel:

   ```
   obabel -:'CC(=O)CC' --gen3d -oxyz -Ostart.xyz
   ```

1. Optimise the structure with xtb at the same settings as you intend to use crest with. This is recommended as crest uses this structure for sanity checks, but not strictly necessary.
   Note that the double dashes `--` divide the options for `runxtb` from the options send to `xtb` (or `crest`, or `enso`).

   ```
   runxtb -- <XYZ> --opt --gbsa <SOLVENT>
   ```

2. Set up to generate a conformer-rotamer-ensemble (CRE) with crest. This will only take a `<XYZ>` (if found `xtbopt.xyz`), convert it to a `coord`, and puts it into a directory called `crest`.

   ```
   crest.prepare
   cd crest
   ```

3. (optional) Check for tautomers (requires crest 2.7).

   ```
   runxtb -S -C crest -- -tautomerize -gbsa <SOLVENT> -chrg <INT> -uhf <INT>
   ```

   Since the job name is derived from the executed program (or a file, if specified), the output of this part will be written to `crest.runxtb.out`. This step is still quite fast and submission is (probably) not necessary. If you find some tautomers, then you may want to calculate a spectrum for each of them.

4. Generate CRE.

   ```
   runxtb -S -C crest -- -nmr -gbsa <SOLVENT> -chrg <INT> -uhf <INT>
   ```

   The output of this step will also write to `crest.runxtb.out`. If you looked for tautomers (step 3), then you might want to rename that output file, but it will be backed up in any case.

5. Run the enso script to get your settings.
   You will need either `$HOME/.ensorc` or `./.ensorc` for this.

   ```
   runxtb -C enso -o stdout
   ```

   This is very fast and doesn't need submission. The output should go to the terminal, so you can check it easily.

6. Most of the times the default settings, which you chose them in rc file, are already what you want. If you want to deviate from the defaults you have set, e.g. change a basis set, or turn off a part, you can now edit the `flags.dat` to make these changes.

7. Run enso. This will take time. Multiple calculations will be performed with Orca or Turbomole. This step also needs (much) more memory than any of the others. There is a small caveat with using Orca: In the enso script the value for `maxcore` value is hard-coded to be 8000 MB (see also the Orca Input Library). In case of the rwth0425 installation, I have changed it to 4000 MB, which should suffice for most applications. Given sufficient overhead for Orca, I recommend at least 5 GB *per cpu* (better would be 5.5 GB), which you can set with the `-m` switch to the runxtb script. The value for `maxcore` in case of Turbomole can be set in the `.cefinerc`. Depending on the number of conformers to be treated, you should adjust the requested walltime (`-w` switch).
   I am using the Orca interface, and it worked well from the command line. However, since Orca uses

MPI, the job script created by runxtb must be edited (see below). I tried Turbomole, but couldn't make that work because of solvation in step 3.
This is a workaround for Orca (for now). Create a script, but do not submit it (`-s` switch):

```
runxtb -s -m <INT> -w <HH:MM:SS> -C enso -- -run
```

The output of this part will be written to `enso.runxtb.out`.
Now switch the values for tasks and cpus:

```
--ntasks=<INT>           (was)   --ntasks=1
--cpus-per-task=1        (was)   --cpus-per-task=<INT>
```

Remove the `srun` wrapper before the `enso` command; the line should be:

```
enso -run > "enso.runxtb.out" || exit_status=1
```

I don't know the reason, but the orca input generated with enso will use the `omp` value for the `%pal` block, so this needs to be set to a reasonable number. I believe (but am not sure at all) that `omp` multiplied by `maxthreads` should not exceed the number of CPU you have available. I had quite successful runs with `omp: <INT>` (CPU I request) and `maxthreads: 1` in `flags.dat`.
Submit the script to slurm:

```
sbatch enso.runxtb.slurm.bash
```

(Alternatively to all of the above on CLAIX18, get an interactive session with `salloc` and run everything interactively.)

8. Run the NMR program. You probably have to specify the measuring frequency (`-mf <INT>`). This should also be very quick; there is no need for batch submission.

   ```
   runxtb -C anmr -- -mf <INT>
   ```

   The output will be written to `anmr.runxtb.out`, and the program should create the file `anmr.dat` for the next step.

9. Plot the spectrum, which is also quick.

   ```
   runxtb -o stdout -C nmrplot.py -- -i anmr.dat -start 0 -end 11
   ```

   If you are unsure about the options, get help:

   ```
   runxtb -o stdout -C nmrplot.py -- -h
   ```

10. Look at the spectrum. Be happy.

Notes for rwth0425:

Source the paths for the modules from the project (e.g. from your `bashrc`):

```
. ~rwth0425/initialise/init_rwth0425.sh
```

If you are using the bashrc from rwth-tools, you can also create a symbolic link to the above script ind the initialisation script directory.

```
cd ~/local/bash_profile.d/
ln -s ~rwth0425/initialise/init_rwth0425.sh init_rwth0425.bash
```

The above mentioned runxtb script is pre-installed and configured, if you load the bin module.

```
module load rwth0425-bin
```

Get a copy of a modified (for this set-up) `.ensorc` to your home,

```
cp ~rwth0425/local/xtb/examples/ensorc "$HOME"/.ensorc
```

or to your current directory (the directory with the CRE):

```
cp ~rwth0425/local/xtb/examples/ensorc ./.ensorc
```

(Same applies to `cefinerc`.) Make modifications as appropriate.

This guide was last updated: 2019-09-10.