Manual for `tools-for-g16.bash` (0.2.0_alpha2, 2019-03-08)

Martin C Schwarzer

March 12, 2019

Contents

Preamble

The following document is currently little more than the already existing reference sheet. As documentation often goes, it takes a bit more time to do it properly. I apologise for the inconvenience.

1 Introduction

The essence of quantum chemistry is performing calculations with a variety of different methods and programs. Quite a few of these programs use low level input and output systems. In principle you are usually dealing with text-based files.

It is in the nature of the problem, that some of the regular tasks are quite repetitive. In order to ease these tasks, but more importantly also secure a certain level of consistency, some steps can and should be automated. This is where this repository, tools-for-g16.bash, comes into play. Within it are contained a few scripts written in bash, which are targeted to aid the use of the quantum chemistry software package Gaussian 16. They cover automatic generation of input files, submission of these to a queueing system, and post-processing of output files to archive and extract results.

Please understand that this project started out to help me with my everyday work. I am happy to hear about suggestions and bugs. I am also fairly certain, that it is and will be a work in progress for quite some time. Be aware that it could be in constant flux.

This 'software' comes with absolutely no warrenty. None. Nada. If you decide to use any of the scripts, it is entirely your resonsibility.

1.1 Preliminary notes on the usage of this manual

To make the manual easier to understand a few abreviations and common notations are used: Anything set in brackets `[ ]` indicates optional arguments or inputs. Arguments in angles `< >` require human input, and a vertical bar `|` indicates alternatives between the inputs or options.

The following abbreviations will be used throughout:

`opt`	Short for option(s)
`ARG`	String type argument
`INT`	Positive integer (including zero)
`NUM`	Signed whole number (including zero, the plus sign can be omitted)
`FLT`	Signed floating point number (the plus sign can be omitted)
`DUR`	Duration in format `[[HH:]MM:]SS`

2 Installation & Configuration

The scripts of this repository are not self-contained, they each need access to the resources directory and it has to be called like that.

The easiest way to install the script is to clone the git repository from github.com/polyluxus/tools-for-g16.bash. Alternatively you can download the tarball archive of the latest release from github.com/polyluxus/tools-for-g16.bash/releases/latest.

After unpacking it only needs to be configured. There are a two kinds of file names recognised: `g16.tools.rc` and the hidden `.g16.toolsrc`. The repository comes with an example of the former, therefore updating the repository will (probably) overwrite the file. The latter file is excluded from this process, and therefore has always precedence, so it is generally a safer option to configure.

The scripts will search for these configuration settings in the following order of directories:

1. the path to the script itself
2. the user's home directory
3. the `.config` directory in the user's home directory
4. the current working directory, i.e. from where the script is called.

Only the last found file will be applied.

The repository comes with a configuration script in the `configure` directory. It produces a formatted file like `g16.tools.rc` from old or the default settings. While you can assign an arbitrary filename, I recommend to store as `.g16.toolsrc` in the root directory of the repository.

To make the scripts accessible from anywhere, the directory where they have been stored must be found in the `PATH` variable. Alternatively, you can create softlinks to them in a directory, which is already recognised by

PATH. A common setting for this is the local $HOME/bin. The configure script will, if desired, try to create these softlinks (omitting the sh suffix).

3 Utilities

There are essentially three different kinds of scripts contained in thins repository: input generation scripts, execution/ submission scripts, and post-processing scripts.

The input generation scripts will, as the description may hint, parse a given file and write an input file for Gaussian 16. These scripts are designed to be chained, so that a generated input file may act as a template for a new input file. This of course makes them dependending on each other, more precisely the newly generated input relies on the successful creation of a checkpoint file of the previous job. The mechanism employed by the scripts was introduced in Gaussian 09 Rev. E.01, with the %OldChk directive, which essentially copies the checkpoint file of a completed calculation to the new location. The contents of which can then be read, or accessed, with the geom/ guess keywords. That actually means, that the input files of a calculation chain can be prepared in advance without knowing the actual outcome, then submitted with these dependencies.

In Gaussian there is a mechanism to do this with one input file, so called compound jobs, but I (personally) discourage everyone from using this. There are a few reasons for this, but the most important is that non-standard route commands will not be copied to a second job, do depending on what was done, the level of theory may actually vary.[1] Second most important is that any of these steps may fail; if they do, you losse the fallback and with it results and calculation time. Another inconvenience is that some visualisation programs are not able to parse these kind of output files.

Essentially the scripts emulate this process, but split it up and externalise the dependencies.

The second gourp of scripts are execution or submission scripts. Working with a queueing system often comes with some wating time. Adjusting the parameters of the calculations to reasonable values can lead to significant reductions of these times. To further avoid failing in early stages, it is reasonable to check the route section for syntax errors before submission. Nothing is worse than waiting for a job to start, only to realise that a keyword is misspelt.

In this category one crucial example is yet missing: A multipurpose wrapper (or shortcut) script, to actually call the utilities without having them permanently accessible in the environment. The whole repository was intended to be based on this, however, it is still the one thing that is missing.[2]

The last category are post-processing scripts, which help extract data, or transform data files into file formats, which are more suitable for archiving.

This chapter serves as an overview for the scripts and the options with which they come.

3.1 Input preparation: g16.prepare.sh

This tool reads in a file containing a set of cartesian coordinates and writes a Gaussian inputfile with predefined keywords. The script interfaces to Xmol format as it is understood by most molecular editors, but it understands the Turbomole (or in extension GFN-xTB) coord format, too, if Open Babel is installed on the system.

The implementation actually only looks for lines containing (at most) two letters and four floating point numbers separated by spaces (or tabs). It is therefore suitable to also parse already generated input files. This was necessary due to the incapabilities of GaussView to produce actual coordinate files; it rather produces Gaussian input data files (*.gjf) containing structural information.

This tool is incapable of parsing/ producing z-matrix style input.

Usage:

g16.prepare.sh [opt] <file>

Options:

-T <FLT>	Temperature (kelvin). This switch adds the temperature keyword to the route section.
-P <FLT>	Pressure (atmosphere). This switch adds the pressure keyword to the route section.
-r <ARG>	Add ARG to route section. This keyword may be used to add any keyword to the route section.
-R <ARG>	Specific route section ARG. With this option, you can specify a complete route section. It will be checked for syntax errors, and it can further be modified with -r <ARG>.
-l <INT>	Load predefined route section. The basic configuration (and the default values) comes with a couple predefined route sections (as examples). They can be changed in the configuration files and eith the configuration script. I still have not gotten around to write a route section building script, so this has to be done by hand.

[1] See the Gaussian website http://gaussian.com/input/?tabid=1 for details about multistep jobs.

[2] For comparison, the wrapper script existed in the now deprecated github.com/polyluxus/tools-for-g09.bash repository; and it may be implemented some time in the future.

`-l list`	Show all predefined route sections. This is a special argument to the above.
`-t <ARG>`	Adds `ARG` to end of file. Sometimes it is necessary to add additional lines to the end of an input file. This option can be used to do that. Fair warning though: this is still somewhat experimental and not very convenient to use. To include an empty line, the argument has to be set explicitly empty: `-t ''`.
`-C <ARG>`	Specify caption/title of job. Gaussian requires a title section, which must not be empty. Replacements: `%F` input filename; `%f` input filename without `.xyz`; `%s` like `%f`, also filtering `start`; `%j` jobname; `%c` charge (with indicator `chrg`); `%M` multiplicity (with indicator `mult`); `%U` unpaired electrons (with indicator `uhf`) The script defaults to using `Calculation: %j; %c; %M; %U; (date)`.
`-j <ARG>`	Jobname of the calculation. The script selects this on the basis of the given molecular structure file name (`<file>`) and derives all other file names from it. This switch can be used to explicitly ovewrite this.
`-j %f`	Jobname is `<file>` filtering the extension `.xyz`
`-j %s`	Jobname is `<file>` filtering the extension `start.xyz`
`-f <ARG>`	Filename of generated input. This is normally derived from the jobname, but ocasionally it is worthwhile to set it explicitly.
`-c <NUM>`	Charge (default: 0).
`-M <INT>`	Multiplicity (default: 1; ≥ 1).
`-U <INT>`	Unpaired electrons (unset; ≥ 0). This option will overwrite a multipliity and vice versa.
`-m <INT>`	Memory (megabyte). This will add `%Mem=INTMB` to the input file.
`-p <INT>`	Processors. This will add `%NProcShared=INT` to the input file.
`-d <INT>`	Disksize (megabyte). This will add the `MaxDisk` keyword to the input file.
`--`	Explicitly close reading script options.
`-s`	Silence script (incremental) to supress messages, warnings, errors.
`-h`	Print a small help file.

3.2 Pre-processing: g16.testroute.sh

This tool parses a Gaussian 16 input file and extracts the route section, which it then checks for syntax errors with the Gaussian 16 utility `testrt`. Unfortunately the original Gaussian utility requires a string as input, which is quite inconvenient if you want to test the inputfile itself. The input file is parsed by the same routine as used in the other scripts, and can be used to validate the created (or to be submitted) files.

Usage:

```
g16.testroute.sh [opt] <file>
```

Options:

`--`	Explicitly close reading script options.
`-s`	Silence script (incremental) to supress messages, warnings, errors.
`-h`	Print a small help file.

3.3 Input preparation: g16.dissolve.sh

This tool reads in a Gaussian 16 input file to extract the current route section and adds relevant keywords for solvent corrections and produce a new input file. Since this again utilises the `%OldChk` directive in conjunction with the `geom`/ `guess` keywords, the calculation it is based on should be completed.

The list of options regarding the `scrf` keyword is long and for specifics it is referred to the Gaussian online manual.

If nothing but a file name is specified, the script defaults to writing a single point calculation input file with the `scrf(pcm,solvent=water)` keyword and options.

Usage:

```
g16.dissolve.sh [opt] <file>
```

Options:

`-o <ARG>`	Adds option `ARG` to the `SCRF` keyword. This may be used to specify the method (like `PCM`, `SMD`, `dipole`, etc.). The default is PCM. The solevent should no be set with this switch, but with `-S <ARG>` instead. This option may be specified multiple times, the input stack will be collated and written to file.
`-S <ARG>`	Adds option `solvent=ARG` to the `SCRF` option list. The default solvent is water. This option will cause an error if specified more than once.
`-O`	Runs an optimisation (preserves or adds `OPT`). The default of this script is to run a singlepoint calculation. Rather than adding the `opt` keyword anew (with `-r opt(<ARG>)`), this switch prevents deleting it in the first place.

`-r <ARG>`	Add `ARG` to route section. This keyword may be used to add any keyword to the route section; it can be specified multiple times.
`-t <ARG>`	Adds `ARG` to end of file. This switch functions the same way as in the prepare script.
`-f <ARG>`	Filename of generated input.
`-m <INT>`	Memory (megabyte). This will add `%Mem=INTMB` to the input file.
`-p <INT>`	Processors. This will add `%NProcShared=INT` to the input file.
`-d <INT>`	Disksize (megabyte). This will add the `MaxDisk` keyword to the input file.
`--`	Explicitly close reading script options.
`-s`	Silence script (incremental) to supress messages, warnings, errors.
`-h`	Print a small help file.

3.4 Input preparation: g16.freqinput.sh

This tool reads in a Gaussian 16 inputfile to extract the current route section and adds relevant keywords for a frequency calculation and produce a new input file. Since this again utilises the `%OldChk` directive in conjunction with the `geom/ guess` keywords, the calculation it is based on should be completed.

There are some options regarding the `freq` keyword, some common ones are mentioned below, but for specifics it is referred to the Gaussian online manual. If nothing further is specified, no additional options will be activated.

Usage:

```
g16.freqinput.sh [opt] <file>
```

Options:

`-o <ARG>`	Adds option `ARG` to the `freq` keyword. This option may be specifie multiple times, the input stack will be collated and written to file. Example options are `NoRaman`, `VCD`, `Anharmonic`, etc..
`-R`	Writes a property run input file to redo a frequency calculation. This adds the option `ReadFC` to the `freq` option list, which is most comonly used to repeat the statistical thermodynamics at different temperatures and pressures (see below). Note that this behaves similar to `opt(RCFC)`, but is spelt differently.
`-T <FLT>`	Temperature (kelvin). This switch adds the `temperature` keyword to the route section.
`-P <FLT>`	Pressure (atmosphere). This switch adds the `pressure` keyword to the route section.
`-r <ARG>`	Add `ARG` to route section. This keyword may be used to add any keyword to the route section; it can be specified multiple times.
`-t <ARG>`	Adds `ARG` to end of file. This switch functions the same way as in the prepare script.
`-f <ARG>`	Filename of generated input.
`-m <INT>`	Memory (megabyte). This will add `%Mem=INTMB` to the input file.
`-p <INT>`	Processors. This will add `%NProcShared=INT` to the input file.
`-d <INT>`	Disksize (megabyte). This will add the `MaxDisk` keyword to the input file.
`--`	Explicitly close reading script options.
`-s`	Silence script (incremental) to supress messages, warnings, errors.
`-h`	Print a small help file.

3.5 Input preparation: g16.ircinput.sh

This tool parses a Gaussian 16 inputfile from a previous frequency calculation and adds relevant keywords to produce two input files for separate IRC calculations, i.e. the forward and the reverse direction. Since this again utilises the `%OldChk` directive in conjunction with the `geom/ guess` keywords, the calculation it is based on should be completed. The script will assume it is based on a successful frequency calculation, and will therefore write a file that will retrieve calculated Cartesian force constants from the checkpoint file. This means that `RCFC` will always be included as an option to the `irc` keyword. If there is no `freq` keyword in the input stream, the script will issue a warning, and the created input files should be checked manually.

Therefore the script defaults to adding `irc(RCFC,forward/reverse)` to the respective route sections.

There are some more options regarding the `irc` keyword, some common ones are mentioned below, but for specifics it is referred to the Gaussian online manual.

Usage:

```
g16.ircinput.sh [opt] <file>
```

Options:

-o <ARG>	Adds option ARG to the irc keyword. This option may be specifie multiple times, the input stack will be collated and written to file. Example options are MaxPoints=10, StepSize=10, etc.. Please note that, as described above, the option RCFC will alway be written; therefore specifying CalcFC will lead to an error in the calculations.
-r <ARG>	Add ARG to route section. This keyword may be used to add any keyword to the route section; it can be specified multiple times.
-t <ARG>	Adds ARG to end of file. This switch functions the same way as in the prepare script.
-f <ARG>	Filenametemplate of generated input files; If <ARG>=jobname.suffix then it produces jobname.fwd.suffix and jobname.rev.suffix.
-m <INT>	Memory (megabyte). This will add %Mem=INTMB to the input file.
-p <INT>	Processors. This will add %NProcShared=INT to the input file.
-d <INT>	Disksize (megabyte). This will add the MaxDisk keyword to the input file.
--	Explicitly close reading script options.
-s	Silence script (incremental) to supress messages, warnings, errors.
-h	Print a small help file.

3.6 Input preparation: g16.optinput.sh

This tool parses a Gaussian 16 inputfile and writes a new inputfile for a structure optimisation. Originally this was desingend to be chained onto a previously completed IRC calculation to converge the structure into a local minimum. This has since changed and the utility of this script is apparently wider. It can also, for example, used to write an input file to remove imaginary modes that were discovered during a frequency calculation. Another use would be chaining it onto a successful single point calculation, e.g. after applying solvent corrections. Since this again utilises the %OldChk directive in conjunction with the geom/ guess keywords, the calculation it is based on should be completed. Another usage is performing scans, or unfreezing scans, etc..

There are many options for the opt keyword, some common ones are mentioned below, but for specifics it is referred to the Gaussian online manual.

If no options are specified, this just defaults to adding the opt keyword.

Usage:

g16.optinput.sh [opt] <file>

Options:

-o <ARG>	Adds option ARG to the opt keyword. This option may be specifie multiple times, the input stack will be collated and written to file. Example options are TS, AddGIC/ModRedundant, MaxStep=10, MaxCycles=300, etc..
-r <ARG>	Add ARG to route section. This keyword may be used to add any keyword to the route section; it can be specified multiple times.
-t <ARG>	Adds ARG to end of file. This switch functions the same way as in the prepare script.
-f <ARG>	Filename of generated input.
-m <INT>	Memory (megabyte). This will add %Mem=INTMB to the input file.
-p <INT>	Processors. This will add %NProcShared=INT to the input file.
-d <INT>	Disksize (megabyte). This will add the MaxDisk keyword to the input file.
--	Explicitly close reading script options.
-s	Silence script (incremental) to supress messages, warnings, errors.
-h	Print a small help file.

3.7 Input preparation: g16.spinput.sh

This tool is the counterpart to g16.optinput.sh, and it creates a new inputfile for a subsequent single point calculation from a specified Gaussian 16 inputfile. If nothing is specified as options, i.e. a new route section or additional keywords, it will simply remove the opt keyword along with its options.

It is possible to overwrite the existing route section to e.g. use a different level of theory. It is still based on the %OldChk directive add the geom/guess keywords. This means that data from a previous calculation will be retrieved.

Usage:

g16.spinput.sh [opt] <file>

Options:

-r <ARG>	Add ARG to route section. This keyword may be used to add any keyword to the route section; it can be specified multiple times.

`-R <ARG>`	Specific route section `ARG`. With this option, you can specify a complete route section. It will be checked for syntax errors, and it can further be modified with `-r <ARG>`.
`-t <ARG>`	Adds `ARG` to end of file. This switch functions the same way as in the prepare script.
`-f <ARG>`	Filename of generated input.
`-m <INT>`	Memory (megabyte). This will add `%Mem=INTMB` to the input file.
`-p <INT>`	Processors. This will add `%NProcShared=INT` to the input file.
`-d <INT>`	Disksize (megabyte). This will add the `MaxDisk` keyword to the input file.
`--`	Explicitly close reading script options.
`-s`	Silence script (incremental) to supress messages, warnings, errors.
`-h`	Print a small help file.

3.8 Input submission: `g16.submit.sh`

This tool parses and then submits a Gaussian 16 inputfile to a queueing system.

More specifically it will modify the supplied input file to match the requested resources for the queueing system. It will create a bash script to be executed as a batch job, i.e. setting up the correct environment and executing Gaussian 16.

Currently there are three queueing systems supported: PBS, SLURM, LSF (or BSUB). Each come with a generic set-up, and the latter two with some RWTH specific settings. (Note that the RWTH cluster running the LSF queueing system will be decommissioned in April 2019, therefore the developement of this feature will probably be halted at an as-is status.)

If the input files to be submitted are prepared with the tools of this repository, a few warnings are to be expected (specifically the memory and priocessors settings). These basically only serve as a reminder.

Usage:

`g16.submit.sh [opt] <file>`

Options:

`-m <INT>`	Memory (megabyte). This will add `%Mem=INTMB` to the Gaussian input file, it will also set the memory flag for the queuing system. The both settings will not be equal, because some overhead for Gaussian is included.
`-p <INT>`	Processors. This will add `%NProcShared=INT` to the input file, it will also set the appropriate flags for the numbere of threads/CPUs for the queueing system.
`-d <INT>`	Disksize (megabyte). This will add the `MaxDisk` keyword to the input file, but it has no effect in the submitted script.
`-w <DUR>`	Walltime limit. This sets the appropriate flag for the maximum wall time the job will be allowed to run.
`-e <ARG>`	Specify an environment variable `ARG` in format `<VAR=value>`. Occasionally it is necessary to set specific environment variables for Gaussian 16 (see manual), this flag causes them to be written to the submitted script before Gaussian is loaded.
`-j <INT>`	Wait for job with ID `INT`. Only numeric arguments are supported (even though some of the available queueing systems would allow names, too.) This option is useful to chain calculations that depend on each other. Dependencies may stretch over multiple jobs, so this option can also be specified multiple times.
`-H`	Submit with status hold (PBS, SLURM) or `PSUSP` (BSUB). Jobs submitted with this status have to be released later, this could be useful to test submisson. However, RWTH users should be aware that the tools to release a job are not available for ordinary users.
`-k`	Only create (keep) the jobscript, do not submit it. This option is useful to see what the script looks like before commiting it to the queuing system, and possibly ammend /modify it. (This was initially born by the idea to delete the script after successful submission, hence the name of the option.)
`-Q <ARG>`	Queue for which job script should be created `<queue>-<special>` (`<queue>`: `pbs`, `slurm`, `bsub`; `<special>`: `gen` [generic], `rwth`; see above for more details).
`-P <ARG>`	Account to project (BSUB) or account (SLURM); if `ARG` is `default|0|''` presets are overwritten. If accounting is activated for BSUB or SLURM, this option will cause the appropriate flag to be set in the submission script. (I have no experience for the PBS queueing system with that, so this is not implemented right now.)

`-M <ARG>` Specify a machine type (only `bsub-rwth`); if `ARG` is `default|0|''` presets are overwritten. This is a very specific setting, and is probably soon be deprecated. Specifically, this will cause `#BSUB -m ARG` to be added to the submit script.

`-u <ARG>` set user email address (SLURM, BSUB); if `ARG` is `default|0|''` presets are overwritten. (I have no experience for the PBS queueing system with that, so this is not implemented right now.)

`--` Explicitly close reading script options.

`-s` Silence script (incremental) to supress messages, warnings, errors.

`-h` Print a small help file.

3.9 Post-processing: g16.getenergy.sh

This tool finds energy statements from the output files of Gaussian 16 calculations. It is basically an elaborate interface to the grep command, which can be used to create summaries of the output files.

If no files are given as argument, the script will operate on all output files in the current directory. The default operating mode of the script will look for input files with the configured suffix (default `*.com`), then match the appropriate output suffix (default: `*.log`), and extract energies from these files.

As such, it does not matter where `<file>` is an input or output file.

Usage:

```
g16.getenergy.sh [opt] [<file(s)>]
```

Options:

`-i <ARG>` Specify the input suffix to be used for finding matching output files when processing a directory.

`-o <ARG>` Specify the output suffix to be used for finding the files when processing a directory.

`-L` Print the full file and path name of the parsed output file (seperated by newline). The default behaviour is printing only the filename without the suffix.

`--` Explicitly close reading script options.

`-s` Silence script (incremental) to supress messages, warnings, errors.

`-h` Print a small help file.

3.10 Post-processing: g16.getfreq.sh

This tool summarises the output of a Gaussian 16 frequency calculation and extracts and reformats the thermochemistry data. This is similarly to `g16.getenergy.sh` an elaborate interface to the grep command, but it (depending on the level of verbosity) extracts all corrections necessary to compute thermochemistry, see also the Gaussian White Paper on thermochemistry.

Usage:

```
g16.getfreq.sh [opt] <file(s)>
```

Options:

`-v` Incrementally increase verbosity of the script.

`-V <INT>` Set level of verbosity directly, (`0|1|2|3|4`). The different levels are as follows: (0; default) display a single line with only the filename `file`, electronic energy E_{el}, zero-point correction E_{ZPE}, enthalpy correction H_{corr}, and correction to the Gibbs energy G_{corr}; (1) display a single line of most values (equal to `-v`), which also prints the method, temperature T, pressure p, internal energy correction U_{corr}, entropy S_{tot}, constant volume heat capacity $C_{V,\mathrm{corr}}$; (2) display a short table of most values (like above, equal to `-vv`); (3) like 2, also repeats the route section (equal to `-vvv`); (4) like 3, also includes the decomposition of the entropy, thermal energy E_{tot} and heat capacity into electronic, translational, rotational, and vibrational contributions (equal to `-vvvv`). If this option is found, `-v` will be ignored.

`-c` Separate values by comma (`-V0|-V1`).

`-f <ARG>` Write the summary to file `file` instead of screen.

`--` Explicitly close reading script options.

`-s` Silence script (incremental) to supress messages, warnings, errors.

`-h` Print a small help file.

3.11 Post-processing: g16.chk2xyz.sh

This tool is intended to create files for archiving a Gaussian 16 calculation. Since normal Gaussian checkpoint files are binary files, and hence platform dependent, it interfaces to the `formchk` utility to produce a formatted

checkpoint file (a text file). It further uses Open Babel to write a molecular structure file in Xmol format (`*.xyz`).

It can either be applied to checkpoint files or the current directory. In the latter case it assumes `chk` as the suffix of the checkpoint files.

Usage:

```
g16.chk2xyz.sh [-s] -h | -a | <chk-file(s)>
```

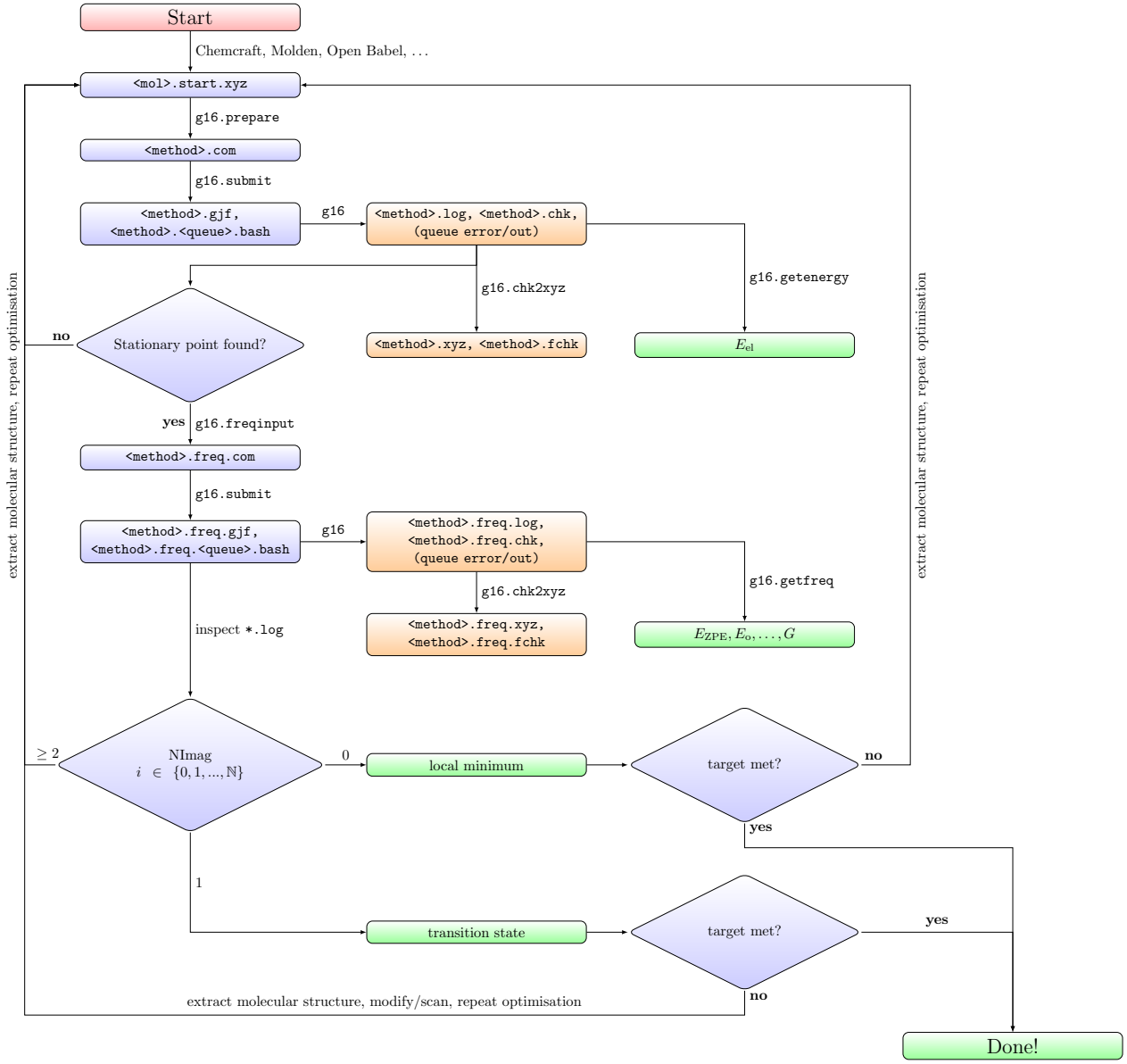
Options:

`-a`	Formats all checkpointfiles that are found in the current directory.
`--`	Explicitly close reading script options.
`-s`	Silence script (incremental) to supress messages, warnings, errors.
`-h`	Print a small help file.

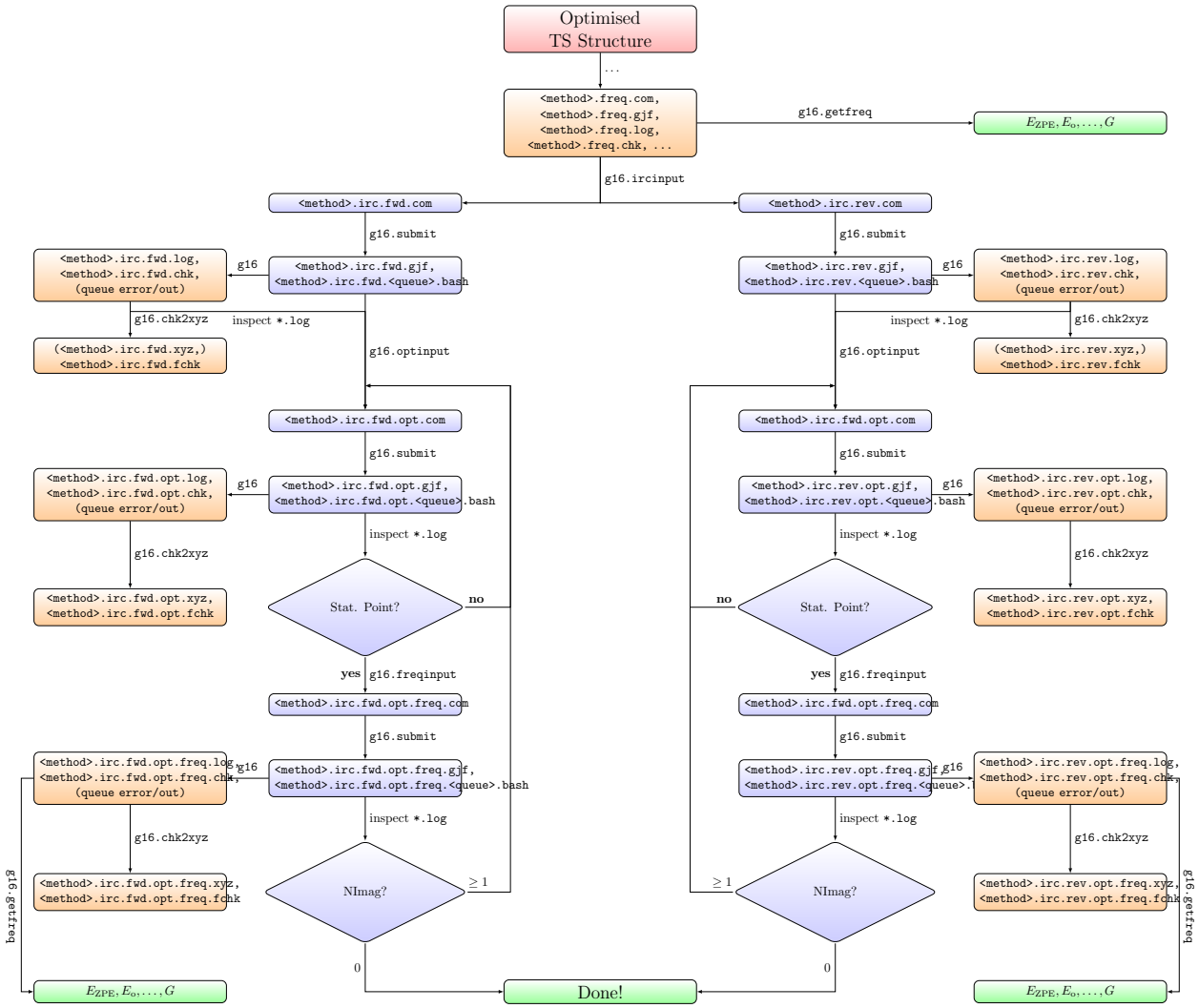
4 Usage examples

This chapter is still in heavy developement.

The following chart presents a basic diagramme of how the tools may interact with each other, or can be chained after one another.



The following chart attempts to do the same for the steps after a successful transition state optimisation.



5 Author, Bugs, and the Rest

This repository was created and is maintained by Martin C Schwarzer (Martin-マーチン polyluxus).

If you find any bugs, have questions, want features implemented, please do it via the GitHub Issue tracker, or fork it and send a pull request.

The help of my colleagues for testing, finding bugs, extending the documentation is greatly appreciated.