Manual for `tools-for-g16.bash` (0.3.0, 2019-07-xx)

Martin C Schwarzer

July 9, 2019

Contents

Preamble

The following document is a more verbose version of the already existing reference sheet. As documentation often goes, it takes some time to reflect all changes properly. Some sections (expecially scetion refsec:usage-examples) may still be incomplete, a work in progress. 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 specified in colon separated format `[[HH:]MM:]SS` *or* with suffixes `INT[D|H|M]` (days/hours/minutes)

1.2 General notes on the processed files

The scripts in this repository use *input* files as templates to write new input files. This will be further explaines in section 3. Route sections of these input files will only be recognised, if they contain a start pattern of the form `#/#N/#P/#T` *followed by a space*, even though a valid Gaussian input does not necessitate this.

2 Installation & Configuration

2.1 Installation of the main scripts

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: github.com/polyluxus/tools-for-g16.bash. Alternatively you can download the tarball archive of the latest

release from there, too:
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.

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`.

2.2 The configuration scripts

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.

The script will prompt for options and will take user input where necessary.

Usage:

```
configure.sh [opt]
```

Options:

`-R`	Route mode. Load the default configuration file and skip directly to the route section editor. This will replace the loaded configuration file.
`-O`	Overwrite mode. Load a (specified) configuration file, run the configuration routine to update settings. This will replace the loaded configuration file.
`-T`	Translate mode. Load the default cofiguration file (possibly from an earlier version), and translate the settings to the most recent version and use default settings for everything unspecified. Depending on the configuration file loaded, this may or may not work well. This will replace the loaded default configuration file.
`--`	Explicitly close reading script options.
`-h`	Print a small help file.

The configure script will also, if desired, try to create softlinks to the scripts (omitting the `sh` suffix), in the `$HOME/bin` directory.

In the `configure` directory is a second script, `routebuilder.sh`, which is not designed to be executed on its own. Instead it interfaces to the `configure.sh` script to update the route sections within the configuration files. However, it can be run by itself, creating an overwiew for the predefined routes and also manipulate them. If no target file is specified, then they will be written to a temporary file.

Usage:

```
routebuilder.sh [opt]
```

Options:

`-r <ARG>`	Raw mode. Do not load the default configuration file, load `<ARG>` instead as input.
`-o <ARG>`	Specify the output file for the created/ stored route sections.
`-m <ARG>`	Specify a custom message `<ARG>` to include into the menu build.
`--`	Explicitly close reading script options.
`-h`	Print a small help file.

2.3 Dependencies

For displaying the help files, the command line utility `grep` is required.

Some scripts need access to an istallation of Open Babel;[1] it is mainly used to interface to the TurboMole `coord` file format, and the post-processing scripts.

It should be no surprise that an installation of Gaussian 16 is necessary for some of the scripts, currently used are the utilities `testrt` and `formchk`. They can either be specified via paths to the respective binaries or directories, or as modules which may be loaed at runtime via the `module` command. A wrapper to set a suitable environment is included within this repository (see next chapter).

For the RWTH clusters, which have a modular environment installed, there is a separate repository with some more scripts, which can also interface to this repository. github.com/polyluxus/rwth-tools.

3 Utilities

There are essentially three different kinds of scripts contained in this 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. For example, an input file created with `g16.prepare` (3.1) can be used with `g16.freqinput` (3.3), or with `g16.dissolve` (3.2). In the next chapter the interplay of the utilities is further explained with example usages. This of course makes these files 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, depending on what was done, the level of theory may actually vary.[2] Second most important is that any of these steps may fail; if they do, you loose 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 group of scripts are execution or submission scripts. Working with a queueing system often comes with some waiting time. Adjusting the parameters of the calculations to reasonable values can lead to significant reductions of these times (see `g16.submit`, 3.8). To further avoid failing in early stages, it is reasonable to check the route section for syntax errors before submission (see `g16.testroute`, 3.7). 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

[1] For more information see http://openbabel.org/.
[2] See the Gaussian website http://gaussian.com/input/ for details about multistep jobs.

intended to be based on this, however, it is still the one thing that is missing.[3]

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

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 simple Xmol[4] 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.

Since the implementation actually only looks for lines containing a string (at least one letter) and four floating point numbers separated by spaces (or tabs), it is therefore also suitable to also parse already generated input files, which contain Cartesian coordinates. 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 with the configuration script. Each of the predefined route sections has a free text comment associated to it. Along with the configuration script comes a route section building script, which should simplify the process of defining them.
`-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)`.

[3] 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.

[4] The prepare script is currently limited to a very simple version of the Xmol file format, and currently only supports the four-field format. Each file therefore should contain a two line "header", with the number (integer) of atoms in the first, and a free form comment in the second. This header, however, will be discarded by g16.prepare.sh. The "body" of the file contains the positions of the atoms, one line each, in Cartesian coordinates: `Element-Symbol xx.xxx yy.yyy zz.zzz`, where the element symbol is a string and the other fields are floating point numbers. The five-field format, which includes an optional charge for the element, or the eight-field format, which assigns an optional vector associated with the element, as well as the seven-field format, which is the same as before, but without the charge, are not yet recognised. (If there were implemented, everything except the first four fields would be discarded.)

`-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 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.

3.3 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.

3.4 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.

3.5 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.

3.6 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.

3.7 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 input file 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.

3.8 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.

3.8.1 The extended (or extra) mail interface

This an internal setting for the submission script to include a routine to send more information as an email as part of the job. It cannot be activated with a commandline option, but has to be configured with the rc file(s).

It is essentially a command, which should be executed during the cleanup routine of the batch job. This could be in theory any executable file (or script), it doesn't have to be mailing program. However, the executed command will be analogous to the `mail` program, and therefore defaults to:

```
mail -s 'a subject line'
```

The used subject line is built dynamically from the job status, job id, and jobname.

Consequently, the system email of the queueing system can also be turned off.

The responsible setting for this extension are:

```
# Deliver the default queuing system email
# "1/yes/active" (default), or "0/no/disabled"
  qsys_email="active"
# Activate/deactivate sending extra mail
# "1/yes/active", or "0/no/disabled" (default)
  xmail_interface="disabled"
# Interface command (default: unset)
  xmail_cmd="<mymail.sh>"
```

3.9 Execution: g16.wrapper.sh

This tool is intended to provide a Gaussian environment at runtime. This can therefore be used to execute Gaussian utilities interactively, see also the Gaussian online manual for more information.

Usage:

```
g16.wrapper.sh [opt] -- <commands>
```

Options:

`-m <INT>`	Memory (megabyte). This will set the `GAUSS_MEMDEF` and `GAUSS_MDEF` environment variables to `INTMB`, which is equivalent to setting the `%mem=` directive in the input file.
`-p <INT>`	Processors. This will set the `GAUSS_PDEF` environment variable to `INT`, which is equivalent to adding `%NProcShared=INT` to the input file.
`--`	Explicitly close reading script options.
`-h`	Print a small help file.

Examples:

If you have an formatted checkpoint file (`<example.fchk>`), but you would like to recreate the binary (`<example.chk>`, optional), you can issue the following command:

```
g16.wrapper.sh unfchk example.fchk example.chk
```

3.10 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.

3.11 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.

3.12 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.
`-A`	Formats almost all checkpointfiles that are found in the current directory. Skips files in cases where it would overwrite them. This is synonymous with `-aS`.
`-B`	Create backup files where is would overwrite them (default). Only the last option amongst `-B`, `-F`, `-S` will take affact.
`-F`	Forces existing files to be removed before they are written, which is effectively equal to overwriting them. Only the last option amongst `-B`, `-F`, `-S` will take affact.
`-S`	Skips files in cases where it would overwrite them. Only the last option amongst `-B`, `-F`, `-S` will take affact.
`--`	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.

4.1 Basic molecular structure optimisations

The molecular structure is often also referred to as *geometry*, and hence the optimisation of the molecular structure is often referred to as *geometry optimisation*.

Here is a short checklist, flowchart, or list how to get from an initial (guessed) structure to a final structure.

1. Create a molecular structure file in cartesian coordinates (`*.xyz`) with your preferred molecular editor, command line tool, or even text editor.[5]

   It needs a bit training to click together a reasonable guess, Chemcraft has a reasonably large fragments database, Molden has a few standards, but is (technically) able to read in fragments, too. For simple molecules you can sometimes find structures online, maybe even crystal structures. Another easy way is to search for it on ChemSpider, PubChem, or draw it with Chemdraw (etc.) and generate a SMILES code. You can use Open Babel to create a guessed structure. For example, the SMILES for ethanol is `CCO`, see PubChem CID 702, then the command for Open Babel is:

   ```
   obabel -:'CCO' -oxyz --gen3d -Oopt.start.xyz
   ```

   The option `-:` takes a SMILES as an input, the `-o` selects the output format (here chosen as Xmol, Cartesian coordinates), `--gen3d` tells the program to generate 3D coordinates, and `-O` selects the output file to write on; see also the Open Babel Wiki.

   This process sometimes even works for more complex molecules; unfortunately not for organometallic (and

[5] Gaussview is unfortunately unable to write Xmol files, for whatever reason, this is not supported. Instead it can write a standard Gaussian input file, which you could set up ready-for-submission-complete with Gaussview. A lot of people do that, it is the GUI to Gaussian after all, the downside of this approach is that you tend to forget what you are doing.

Not everything in those input files is necessary, and some of it is a terrible choice. For example, it usually includes the absolute paths to checkpoint files. If you decide to change anything (and this is quite likely) and then rerun a calculation, it will very likely fail, or overwrite existing things. If you are using Gaussview locally, but submit a calculation to the cluster, it is also likely, that the specified location does not exist.

There are easy and not quite as easy workarounds:

- Use a different program.
- Edit the files on the cluster by hand, remove everything superflous with a text editor.
- Convert them with, e.g. Molden. (Unfortunately Open Babel cannot do that.)
- Use my prepare script. It will search the inputfile for lines of the pattern `XX +000.0000 -000.0000 +000.0000` (two letters followed by three signed real numbers).
- Alternatively, you can use a different program. (Seriously, it is worth mentioning it twice.)
- You can use Gaussian's `newzmat` utility to convert it.

similar) compounds, because SMILES does not describe ionic interactions.[6] Assume we did this, and we will call it `bp86svp.start.xyz`.

2. Create an input file. A *very simple* optimisation would probably use BP86/def2-SVP with mostly default parameters, or similar,[7] so you can just run the prepare script:

   ```
   g16.prepare -R'#P BP86/def2SVP/W06' -r'OPT(MaxCycles=100)' \
     -j'bp86svp.opt' bp86svp.start.xyz
   ```

   This will create the file `bp86svp.opt.com`.

3. Submit the input file to the queue. The following command will handle this:

   ```
   g16.submit -p12 -m4000 bp86svp.opt.com
   ```

   This will create a modified inputfile `bp86svp.opt.gjf`, with the correct settings for Gaussian according to what you requested in the submit routine. It will also create (assuming it is set to `slurm-gen`) a script for the queueing system, `bp86svp.opt.slurm.bash`, which will be executed remotely.

   What sensible parameters (processors, memory, walltime) are comes with experience. Obviously don't ask for more than you have (`-p12`, `-m4000` is reasonable for medium sized molecule optimisations).

4. Once the calculation is done (you'll probably get an email), you will have a few output files.

   - `bp86svp.opt.log`: the main output
   - `bp86svp.opt.chk`: the checkpoint file (useful for later)
   - `bp86svp.opt.slurm.bash.e012345`: the error file from the queueing system[8]
   - `bp86svp.opt.slurm.bash.o012345`: the standard output file from the queueing system

   Check the main output file for errors. Everything should be fine when you find *Normal termination* at the end (`tail bp86svp.opt.log`). You can (and should) also search for *stationary point found*: `grep 'Stationary point found' -B7 -A3 bp86svp.opt.log`.

   Don't forget to open it with a molecular viewer to make sure the structure is actually sensible.

5. Create a frequency calculation **in the same** directory:

   ```
   g16.freqinput bp86svp.opt.gjf
   ```

   This creates `bp86svp.opt.freq.com` which you can submit:

[6] Some may work okay-ish, e.g. sodium ethanolate (`CC[O-].[Na+]`), others not. You can try building the structure for the cation of tris(ethylenediamine)cobalt(III) chloride: `C(C[NH-])[NH-].C(C[NH-])[NH-].C(C[NH-])[NH-].[Co].[Cl-].[Cl-].[Cl-]`.

[7] Choosing an appropriate level of theory is not often trivial, one has to consider a couple of things, like accuracy, performance, and speed. I have previously written quite a long article about that: DFT Functional Selection Criteria.
The example command uses the DF-BP86/def2-SVP level of theory, but with something extra. Remember:
`g16.prepare -R'#P BP86/def2SVP/W06' -r'OPT(MaxCycles=100)' -j'bp86svp.opt' bp86svp.start.xyz`
The `-R` switch to `g16.prepare` sets the basic route section to `#P BP86/def2SVP/W06`, the different parts mean the following:

- The `#P` selects verbose printing (G16 manual), other options are `#N` (normal) and `#T` (terse).
- The method is selected as `BP86`, which selects the exchange functional `B` and the correlation functional `P86`. There are plenty of Functionals implemented.
- For the example I have chosen the def2-SVP basis set; please note that the keyword is without the dash. There are again many available in Gaussian, and additional ones can be defined; that is something for more advanced users (and a topic for another day).
- Additionally, this route section requests density fitting (sometimes also referred to as *resolution-of-the-identity*, or short RI, approximation) with the auxiliar basis set `W06` (see the Manual for more), which should speed up the calculation. This is also available via the `DensityFit/DenFit` keyword, and therefore commonly abbreviated with DF in the level of theory.

The `-r` switch to the script adds more keywords to the route section. In this specific case we'll request an optimisation (`OPT`) with 100 cycles at the most.
The `-j` switch selects a jobname for the calculation, and it will further be used to derive filenames for it.
The last argument is the molecular structure file, here in the Xmol format. There are some other formats recognised, but that is also something for another day.

[8] The number, here used as example `012345`, is the JobID assigned by the queueing system.

```
g16.submit -p12 -m24000 bp86svp.opt.freq.com
```

Frequency calculations are memory demanding, they will slow down if they do not have enough, about 2 GB per core should be okay.

6. You'll get similar output files like above. Open the `*.log` file with a molecular viewer, and look at the frequencies (or modes). If there are no imaginary modes, you have found a *local* minimum, if there is exactly one mode, you found a transition state. Repeat the above steps as necessary if you didn't meet your target.

7. Look for the energy values and put them in tables to calculate barriers and stuff. You can use the post-processing script for that, or look through the manual by yourself:

   ```
   g16.getfreq -V3 bp86svp.opt.freq.log
   ```

8. Finalise the calculation with creating a formatted checkpoint file and a coordinate file. The Gaussian checkpoint files are binary files and they are machine dependent (and fairly large). That usually doesn't pose any problem, but it is good practice to create a pure text file anyway. The formatted checkpointfile can be used as an input for further analysis by many programs, therefore it is good to have in any case. It can also be reverted to a binary checkpoint file, which means you can actually omit the checkpoint file while archiving.

   Writing an coordinate file with the optimised structure helps cutting down loading times; it also gives you an indicator that the calculation is done.

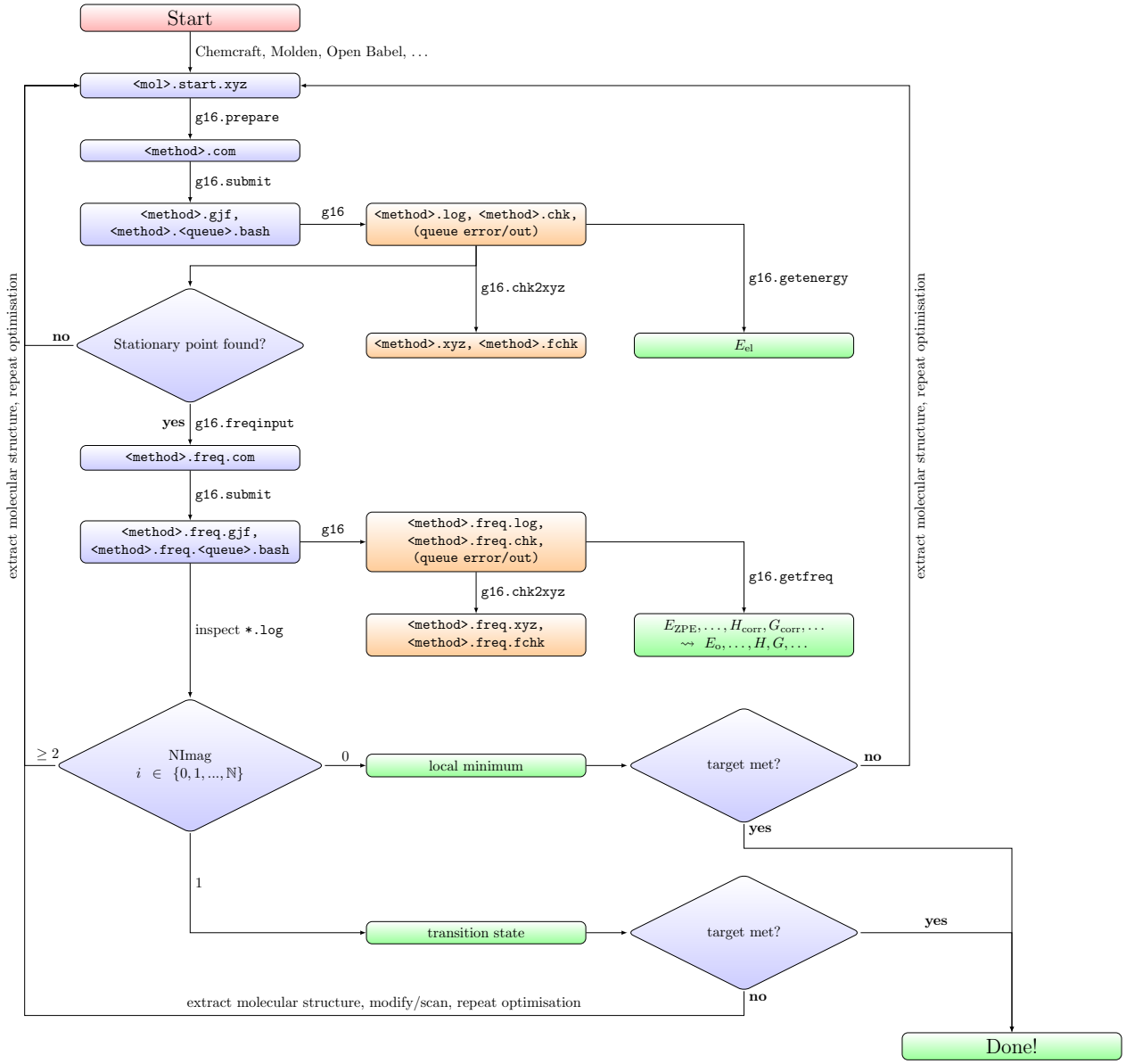
   You can do these two steps together with the following command:

   ```
   g16.chk2xyz bp86svp.opt.freq.chk
   ```

   The above command produces the formatted checkpoint file `bp86svp.opt.freq.fchk` and the coordinate file `bp86svp.opt.freq.xyz`. Alternatively you can use the script with the `-a` switch to do that for every `*.chk` file in the current directory.

Once you have done all the steps you can move on to interpreting the results, which is where often the real work lies.

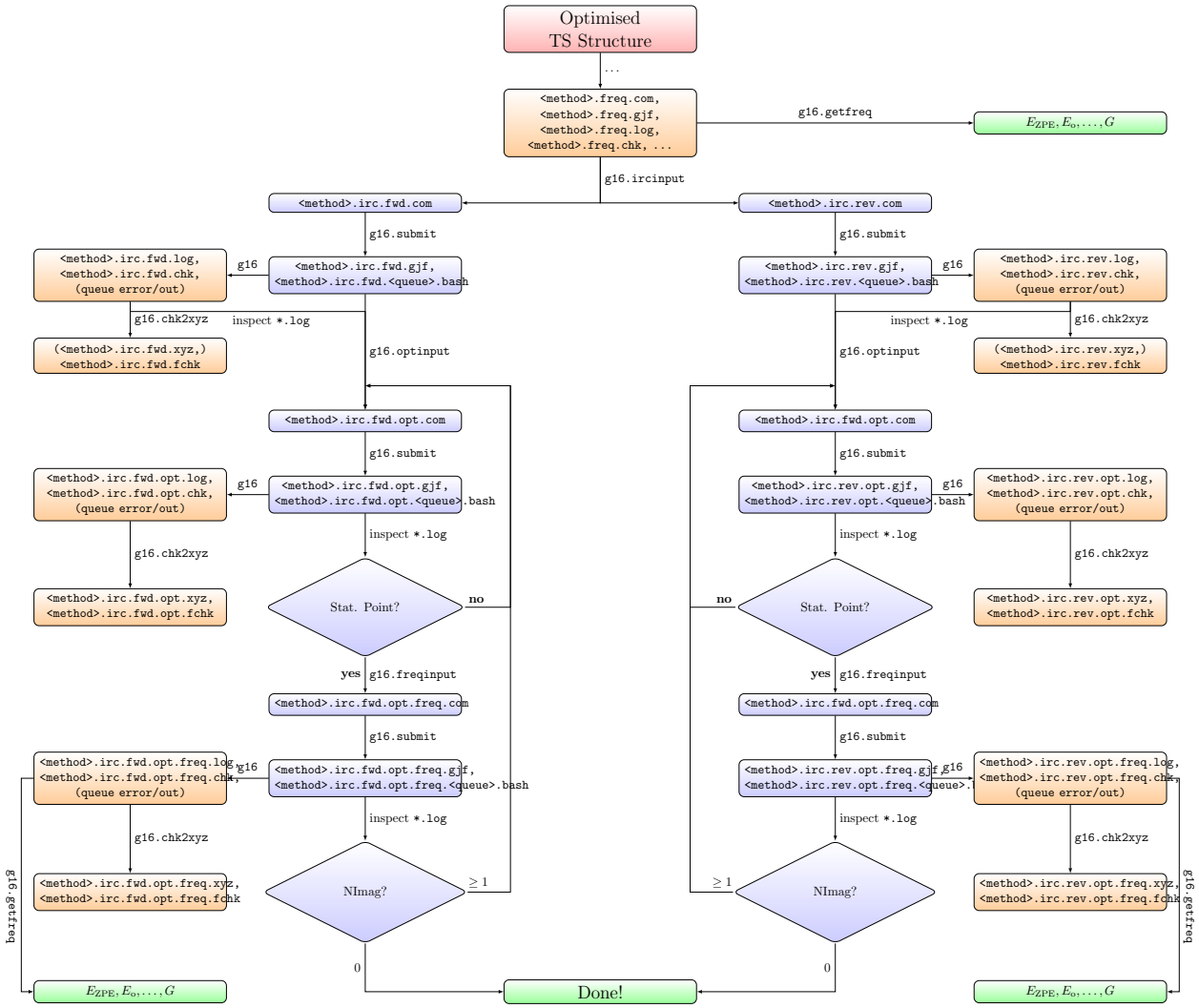
The following chart presents a basic diagramme of the above description, it shows how the tools may interact with each other, or can be chained after one another, respectively.



4.2 Transition state search

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

(This part of the documentation is still *very* much a work in progress.)



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.