# Make a template file

The template file specifies the geometry of the molecule. The atom labels have the elements as the first characters, the word ATOM and then the atom number afterwards.

You can also put “PAR#” with # starting at 1, anywhere in the file. This allows you to substituted values into this position in the file from inside a matlab script.

CATOM1

OATOM2 1 PAR1

HATOM3 1 1.11 2 122.2

HATOM4 1 1.11 2 122.2 3 - 180.0

!ENV

The file should be names \*.tpl.

For a general basis, the format must be: (No newline between !ENV and the basis set, and the end must have 2 blank lines.

CATOM1

OATOM2 1 1.23

HATOM3 1 1.11 2 122.2

HATOM4 1 1.11 2 122.2 3 -180.0

!ENV

-H 0

S 3 PAR1

3.42525091 0.15432897

0.62391373 0.53532814

0.16885540 0.44463454

\*\*\*\*

-C 0

S 3 1.00

71.6168370 0.15432897

13.0450960 0.53532814

3.5305122 0.44463454

S 3 PAR2

2.9412494 -0.09996723

0.6834831 0.39951283

0.2222899 0.70011547

P 3 PAR3

2.9412494 0.15591627

0.6834831 0.60768372

0.2222899 0.39195739

\*\*\*\*

-O 0

S 3 1.00

130.7093200 0.15432897

23.8088610 0.53532814

6.4436083 0.44463454

S 3 PAR4

5.0331513 -0.09996723

1.1695961 0.39951283

0.3803890 0.70011547

P 3 PAR5

5.0331513 0.15591627

1.1695961 0.60768372

0.3803890 0.39195739

\*\*\*\*

# Create a fragment

## From matlab

The matlab command:

frag = Fragment('datapath', config);

creates a fragment, with the template and data stored in the ‘datapath’ directory. The config variable holds the specification for the calculation. The method Fragment.defaultConfig() returns a config structure with default values:

template: 'template' % template file will be template.tpl

basisSet: 'STO-3G' % basis set keyword

charge: 0

spin: 1

par: [] % list of values to substitute for par1 par2 etc in the tpl file.

You then over-ride these anyway you want, i.e.

config = Fragment.defaultConfig();

config.template = 'fhyde';

config.basisSet = 'sto-3g ';

config.par = [1.0 4.0];

frag = Fragment('c:\dave\apoly\msqc\data4', config);

TODO: par should be a cell array, so substitution can be more flexible

## What happens underneath

The datapath directory is searched to see if this calculation has already been done. If it has, the result is loaded. If not, the following is done to generate data. (See end of this section for how this is stored and checked.)

Template file is read in and stored in *templateText* (properties will be italicized).

*natom* and *npar* are initialized by counting # of times ATOM and PAR appear in the template

initializeData() is then called. This routine:

Defines *header* to the part of a Gaussian input file that includes everything down to title.

Full calculation (a normal Gaussian calculation on the entire molecule)

Replaces ATOM# with a space, for # = 1..natom

Replaces PAR# with the values in the config.par array

Defines *gaussianFile* as this input file, adding the keyword “charge” (for use by addEnv)

Writes out the file ‘full.gjf’

Calls Gaussian on full.gjf, producing files full.out, temp.fch, fort.32

Calls formchk on temp.chk, producing temp.fch (a formatted checkpoint file)

Calls Fragment.readfchk() to read data from temp.fch

If this fails, the error ‘failed during fchk read’ is generated

Calls Fragment.readpolyatom() to read data from fort.32

If this fails, the error ‘failed during polyatom read’ is generated

Files are cleaned up by deleting fort.32, full.gjf, full.out, temp.chk and temp.fch

One-nucleus calculations are done in order to get the H1 matrix elements corresponding to interaction with just one nucleus. (Gaussian allows you to specify an element as X-Bq to cause the basis set for that element to be included in the calculation, without having the nucleus present. We set all nuclei except 1 to be a –Bq element, and then H1 gives the interaction of the entire basis set with the nucleus that is not a –Bq)

Loop over iatom and do the following:

Replace all atoms except iatom with a –Bq element

To keep the molecule a singlet, decrease the charge by 1 if iatom has an odd Z

(this means adding an electron to the system to keep even # electrons)

Write this Gaussian input file as atom#.gjf

Run Gaussian: produces atom#.out, temp.fch, fort.32

Call Fragment.readpolyatom() to read H1 from fort.32

Files are cleaned up by deleting fort.32, full.gjf, full.out, temp.chk and temp.fch

Two files are then written:

templateName#\_cfg.mat holds the *config* data structure

templateName#\_calc.mat holds the entire *Fragment* class

where templateName is equal to *config.template* and # is the next number in a sequence of files stored in the datapath.

Two calculations are considered the same if their config structures are the same, i.e. if they are based on the same \*.tpl file and have the same basis set, spin, charge and parameter array. This will work fine, as long as the \*.tpl file is not edited. If \*.tpl is changed, all data based on the template becomes invalid and should be deleted.

## Adding environments

Results for environments are stored in matlab arrays, and pre-dimensioning these to the correct size is a good idea.

frag.setEnvSize(n) % dimensions the arrays to size n

Then you do:

frag.addEnv( env )

The code replaces the word ‘!ENV’ with the external charges

Writes the file env.gjf to the dataPath directory

Calls Gaussian on env.gjf: generates env.out, temp.chk and fort.32

Calls formchk on temp.chk to generate temp.fch

Reads in temp.fch

If this fails, error ‘failed during env fchk read’ is generated

Reads in fort.32

If this fails, error ‘failed during env polyatom read’ is generated