DFTB$^+$

Snapshot 081217

U S E R   M A N U A L

Contents

Chapter 1

Introduction

The code DFTB$^+$ is the Fortran 95 successor of the old DFTB code, implementing the density functional based tight binding approach [9]. The code had been completely rewritten from scratch and extended with various features.

The most important features are:

- Non-scc and scc calculations (with expanded range of SCC accelerators)
  - Cluster/molecular systems
  - Periodic systems (arbitrary K-point sampling, band structure calc.)
- l-shell resolved calculations possible
- Spin polarised calculation (collinear spin)
- Geometry optimisation
  - Steepest descent
  - Conjugate gradient
- Geometry optimisation constraints (in xyz-coordinates)
- Molecular dynamics (NVE and NVT ensembles)
- Improved finite temperature calculations
- Dispersion correction (van der Waals interaction)
- Ability to treat f-electrons
- LDA+U extension
- QM/MM coupling with external point charges (smoothing possible)
- OpenMP parallelisation
- Automatic code validation (autotest system)
- New user friendly, extensible input format (HSD or XML)
- Dynamic memory allocation

- Additional tool for generating cube files for charge distribution, molecular orbitals, etc. (Waveplot)

Chapter 2

Input for DFTB$^+$

DFTB$^+$ can read two formats, either XML or the Human-friendly Structured Data format (HSD). If you are not familiar with HSD format yet, a detailed description is given in appendix A. The input file for DFTB$^+$ must be named `dftb_in.hsd` *or* `dftb_in.xml`. The input file must be present in the working directory. To prevent ambiguity, the parser refuses to read any input if both files are present. After processing the input, DFTB$^+$ creates a file of the parsed input, either `dftb_pin.hsd` or `dftb_pin.xml`. This contains the user input as well as any default values for unspecified options. All values are given in the default internal units. The file also contains the version of the current input parser. You should always keep this file, since if you want to exactly repeat your calculation with a later version of DFTB$^+$, it is recommended to use this file instead of the original input. (You must of course rename `dftb_pin.hsd` into `dftb_in.hsd` or `dftb_pin.xml` into `dftb_in.xml`.) This guarantees that you will obtain the same results, even if the defaults for some non specified options have been changed. The code can also produce `dftb_pin.xml` from `dftb_in.hsd` or vice versa if required (see section 2.6).

The following sections list properties and options that can be set in DFTB$^+$ input. The first column of each of the tables of options specifies the name of a property. The second column indicates the type of the expected value for that property. The letters "l", "i", "r", "s", "p", "m" stand for logical, integer, real, string, property list and method type, respectively. An optional prepended number specifies how often (if more than once) this type must occur. An appended "+" indicates arbitrary occurrence greater than zero, while "*" allows also for zero occurrence. Alternative types are separated by "|". Parentheses serve only to delimit groups of settings.

Sometimes a property is only interpreted if some condition(s) is met. If this is the case, the appropriate conditions are indicated in the third column. The fourth column contains the default value for the property. If no default value is specified ("-"), the user is required to assign a value to that property. The description of the properties immediately follows the table. If there is also a more detailed description available for a given keyword somewhere else, the appropriate page number appears in the last column.

Some properties are allowed to carry a modifier to alter the provided value (e.g. converting between units). The possible modifiers are listed between brackets ([]) in the detailed description of the property. If the modifier is a conversion factor for a physical unit, only the unit type is indicated (length, energy, force, time, etc.). A list of the allowed physical units can be found in appendix B.

2.1 Main input

The input file for DFTB$^+$ (dftb_in.hsd/dftb_in.xml) must contain the following property definitions:

Name	Type	Condition	Default	Page
Geometry	plm		-	5
Hamiltonian	m		-	10

Additionally optional definitions may be present:

Name	Type	Condition	Default	Page
Driver	m		{}	6
Options	p		{}	24
ParserOptions	p		{}	25

Geometry Specifies the geometry for the system to be calculated. See p. 5.

Hamiltonian Configures the Hamiltonian and its options. See p. 10.

Driver Specifies a geometry driver for your system. See p. 6.

Options Various global options for the run. See p. 24.

ParserOptions Various options affecting the parser only. See p. 25.

2.2 Geometry

The geometry can be specified either directly by passing the appropriate list of properties or by using the GenFormat{} method.

2.2.1 Explicit geometry specification

If the geometry is being specified explicitely, the following properties can be set:

Periodic	l		No
LatticeVectors	9r	Periodic = Yes	-
TypeNames	s+		-
TypesAndCoordinates	(1i3r)+		-

Periodic Specifies if the system is periodic in all 3 dimensions or is to be treated as a cluster. If set to Yes, property LatticeVectors{} must be also specified.

LatticeVectors [*length*] The x, y and z components of the three lattice vectors if the system is periodic.

TypeNames List of strings with the names of the elements, which appear in your geometry.

TypesAndCoordinates [relative|*length*] For every atom the index of its type in the TypeNames list and its coordinates. If for a periodic system (Periodic = Yes) the modifier relative is specified, the coordinates are interpreted in the coordinate system of the lattice vectors.

Example: Geometry of GaAs:

```
Geometry = {
  TypeNames = { "Ga" "As" }
  TypesAndCoordinates [Angstrom] = {
    1  0.000000    0.000000    0.000000
    2  1.356773    1.356773    1.356773
  }
  Periodic = Yes
  LatticeVectors [Angstrom] = {
     2.713546    2.713546    0.
     0.          2.713546    2.713546
     2.713546    0.          2.713546
  }
}
```

2.2.2 GenFormat{}

You can use the generic format to specify the geometry (see appendix C). The geometry specification for GaAs would be the following:

```
Geometry = GenFormat {
  2  S
  Ga  As
  1 1    0.000000    0.000000    0.000000
  2 2    1.356773    1.356773    1.356773
  0.000000    0.000000    0.000000
  2.713546    2.713546    0.
  0.          2.713546    2.713546
  2.713546    0.          2.713546
}
```

It is also possible to include the gen-formatted geometry from a file:

```
Geometry = GenFormat {
  <<< "geometry.gen"
}
```

2.3 Driver

The driver is responsible for changing the geometry of the input structure during the calculation. Currently the following methods are available:

{} Static calculation with the input geometry.

SteepestDescent{} Geometry optimisation by moving atoms along the acting forces. See p. 7.

CongjugateGradient{} Geometry optimisation using the conjugate gradient algorithm. See p. 8.

VelocityVerlet{} Molecular dynamics with the velocity Verlet algorithm. See p. 8.

2.3.1 SteepestDescent{}

MovedAtoms	i+lm	Range { 1 -1 }
MaxForceComponent	r	1e-4
MaxSteps	i	200
StepSize	r	100.0
OutputPrefix	s	"geo_end"
AppendGeometries	l	No
Constraints	(1i3r)*	{}

MovedAtoms Index of the atoms which should be moved. If the index range is continuous, the Range{} method can be used.

```
MovedAtoms = Range { 1 6 }
# equivalent with MovedAtoms = { 1 2 3 4 5 6 }
```

Negative indexes can be used to count backwards from the final atom:

```
MovedAtoms = Range { 1 -1 }   # Move all atoms including the last
```

MaxForceComponent [*force*] Optimisation is stopped, if the force component with the maximal absolute value goes below this threshold.

MaxSteps Maximum number of steps after which the optimisation should stop (unless already stopped by achieving convergence).

StepSize [*time*] Step size (δt) along the forces. The displacement δx_i along the ith coordinate is given for each atom as $\delta x_i = \frac{f_i}{2m}\delta t^2$, where f_i is the appropriate force component and m is the mass of the atom.

OutputPrefix Prefix of the geometry files containing the final structure.

AppendGeometries If set to Yes, the geometry file in the XYZ-format will contain all the geometries obtained during the optimisation (instead of containing only the last geometry).

Constraints Specifies geometry constraints. For every constraint the serial number of the atom is expected followed by the x, y, z components of a constraint vector. The specified atom is not allowed to move along the constraint vector. If two constraints are defined for the same atom, the atom will only by able to move normally to the the plane of the two constraining vectors.

Example:

```
Constraints = {
  # Atom one can only move along the z-axis
  1  1.0  0.0  0.0
  1  0.0  1.0  0.0
}
```

2.3.2 ConjugateGradient{}

MovedAtoms	i+lm	Range { 1 -1 }	
MaxForceComponent	r	1e-4	
MaxSteps	i	200	
OutputPrefix	s	"geo_end"	
AppendGeometries	l	No	
Constraints	(1i3r)*	{}	

See previous subsection for the description of the properties.

2.3.3 VelocityVerlet{}

MovedAtoms	i+lm	Range { 1 -1 }	
Steps	i	-	
TimeStep	r	-	
KeepStationary	r	Yes	
Thermostat	m	-	9
OutputPrefix	s	"geo_end"	
MDRestartFrequency	i	1	
Velocities	(3r)*	-	

MovedAtoms List of atoms to move during the MD. (See more detailed description on page 7.)

Steps Number of MD steps to perform. In the case of a thermostat using a TemperatureProfile{} the number of steps is calculated from the profile.

KeepStationary Remove translational motion from the system.

TimeStep [*time*] Time interval between two MD steps.

Thermostat Thermostating method for the MD simulation. See p. 9.

OutputPrefix Prefix of the geometry files containing the final structure.

MDRestartFrequency Interval that the current geometry and velocities are written to the XYZ format geometry file. In the case of SCC MD runs, the charge restart information is also written at this interval.

Velocities [*velocity*] Specified atomic velocities for all the atoms of the given structure (including “velocities” for any stationary atoms, which are silently ignored). This option can be used to restart an MD run, just make sure the geometry is consistent with the specified velocities. The easiest way to do this is to copy both from the same iteration of the XYZ file produced in the previous run (note the velocities printed in the XYZ files are specified in Å/ps, so this should be set in the input). If restarting an SCC MD run, we *strongly* suggest to use ReadInitialCharges, and in particular read charges for the geometry which you use to restart (iterations at which charges are writen to disc are marked in the XYZ file, with the most recent being present in charges.bin).

Thermostat

None{} No thermostating during the run, only the initial velocities are set according to either a given temperature or velocities, hence an NVE ensemble should be achieved for a reasonable time step.

InitialTemperature	r	-

InitialTemperature [*energy*] Starting velocities for the MD will be created according the Maxwell-Boltzmann distribution with the specified temperature. This is redundant in the case of specified initial velocities.

Andersen{} Andersen thermostat [2] sampling an NVT ensemble (note that Andersen thermostatting has a reputation for suppressing diffusion and also prevents accumulation of data for dynamical properties).

Temperature	rlm	-
ReselectProbability	r	-
ReselectIndividually	l	-
AdaptFillingTemp	l	No

Temperature [*energy*] Temperature of the thermostat. It can be either a real value, specifying a constant temperature through the entire run or the TemperatureProfile{} method specifying a changing temperature. (See p. 10.)

ReselectProbability Probability for reselecting velocity from the Maxwell-Boltzmann distribution.

ReselectIndividually If Yes, each atomic velocity is reselected individually with the specified probability. Otherwise all velocities are reselected at once with the specified probability.

AdaptFillingTemp If Yes, the temperature of the electron filling is always set to the current temperature of the thermostat. (The appropriate tag for the temperature of the electron filling is ignored.)

Berendsen{} Berendsen thermostat [4] samples something like an NVT ensemble (but without the correct micro-canonical fluctuations, be aware of the "flying icecube" problem before using this thermostat [12]).

Temperature	rlm	-
CouplingStrength	r	-
AdaptFillingTemp	l	No

Temperature [*energy*] Temperature of the thermostat. It can be either a real specifying a constant temperature through the entire run or the TemperatureProfile{} method specifying a changing temperature. (See p. 10.)

CouplingStrength Dimensionless coupling strength for the thermostat (given as $\Delta t/\tau_t$ in the original paper).

AdaptFillingTemp If Yes, the temperature of the electron filling is always set to the current temperature of the thermostat. (The appropriate tag for the temperature of the electron filling is ignored.)

TemperatureProfile{} Specifies a temperature profile during molecular dynamics. It takes as argument one or more lines containing the type of the annealing (string), its duration (integer), and the target temperature (real), which should be achieved at the end of the given period. For example:

```
Temperature [Kelvin] = TemperatureProfile {   # Temperatures in K
  constant        1    10.0   # Setting T=10 K for the 0th MD-step
  linear        500   600.0   # Linearly rising T in 500 steps up to T=600 K
  constant     2000   600.0   # Constant T through 2000 steps
  exponential   500    10.0   # Exponential decreasing in 500 steps to T=10 K
}
```

The annealing method can be constant, linear or exponential, with the duration of each stage of the anneal specified in steps of the driver containing the thermostat. The starting temperature for each annealing period is the target temperature of the previous one, with the last step of each stage being at the target temperature. Since the initial stage in the temperature profile has no previous step, the default starting temperature is set to 0, but no actual calculation for that temperature is made. In order to start the calculation from a finite temperature for the first annealing period, a constant profile temperature stage with the duration of only one step should be specified as first step (see the example). The temperatures of the stages are specified in atomic units, unless the Temperature keyword carries a modifier.

2.4 Hamiltonian

Currently only a DFTB Hamiltonian is implemented, so you must set Hamiltonian = DFTB{}. The DFTB{} method may contain the following properties:

SCC	l		No	
SCCTolerance	r	SCC = Yes	1e-5	
MaxSCCIterations	i	SCC = Yes	100	
OrbitalResolvedSCC	l	SCC = Yes	No	
Mixer	m	SCC = Yes	Broyden{}	14
MaxAngularMomentum	p		-	
Charge	r		0.0	
SpinPolarisation	m	SCC = Yes	{}	16
SpinConstants	p	SpinPolarisation = Colinear{}	-	
Eigensolver	m		DivideAndConquer{}	17
Filling	m		Fermi{}	17
IndependentKFilling	l	periodic system	No	
SlaterKosterFiles	plm		-	18
OldSKInterpolation	l		No	
PolynomialRepulsive	plm		{}	
KPointsAndWeights	(4r)+lm	periodic system	-	19
OrbitalPotential	m		{}	21
ReadInitialCharges	l		No	
ElectricField	p		{}	21
Dispersion	m		{}	22
DampHShortRange	l	SCC = Yes	No	

SCC If set to Yes, a charge self consistent (scc) calculation is made.

SCCTolerance Stopping criteria for the SCC. Specifies the tolerance for the maximum difference in any charge between two SCC cycles.

MaxSCCIterations Maximal number of SCC cycles to reach convergence. If convergence is not reached after the specified number of steps, the program stops.

OrbitalResolvedSCC If set to Yes, both (or all three/four) Hubbard U values for the different angular momenta are used, when calculating the SCC contributions. Otherwise, the value for the s-shell is used for all angular momenta. Please note, that the old standard DFTB code was *not* orbitally resolved, so that only the Hubbard U for the s-shell was used. Please check the documentation of the SK-files you intend to use as to whether they are compatible with an orbitally resolved SCC calculation (many of the biological files do not use orbitally resolved charges), before you switch this option to Yes. Even if the Hubbard U values for different shells are the same in the SK-files, this flag would still effect your results, since when it is set to Yes, any charge transfer between atomic shells will change the energy of the system compared to when it is set to set to No.

Mixer Mixer type for mixing the charges in an SCC calculation. See p. 14.

MaxAngularMomentum Specifies the highest angular momentum for each atom type. All orbitals up to that angular momentum will be included in the calculation. Several main-block elements require d-orbitals, check the documentation of the SK-files you are using to determine if this is necessary. Possible values for the angular momenta are s, p, d, f.

Example:

```
MaxAngularMomentum = {
  Ga = "p"     # You can omit the quotes around the
  As = "p"     # orbital name, if you want.
}
```

By using the SelectedShells method, there is also the possibility to pick shells with certain angular momenta from one or more species and treat them together as if they were shells of the same atom. The shells to be picked from a certain atom type should be listed without any separating characters. The list of shells of different atom types should be separated by whitespaces.

Example:

```
# Defining sps* basis for Si and C by combining sp and s shells from
# Si and Si2, and C and C2, resp.
MaxAngularMomentum = {
  Si = SelectedShells { "sp" "s" }    # Si atom with sps* basis
  C = SelectedShells { "sp" "s" }     # C atom with sps* basis
}

# Note, that you have to modify the Slater-Koster file definition accordingly
SlaterKosterFiles = {
  Si-Si = "Si-Si.skf" "Si-Si2.skf" "Si2-Si.skf" "Si2-Si2.skf"
  Si-C = "Si-C.skf" "Si-C2.skf" "Si2-C.skf" "Si2-C2.skf"
  C-Si = "C-Si.skf" "C-Si2.skf" "C2-Si.skf" "C2-Si2.skf"
  C-C = "C-C.skf" "C-C2.skf" "C2-C.skf" "C2-C2.skf"
```

If for a given atom type you pick orbitals from more than one species, you must specify an appropriate combinations of file names for the Slater-Koster tables in SlaterKosterFiles{}. For every atom type combination $n_{\mathrm{SK1}} \times n_{\mathrm{Sk2}}$ Slater-Koster files must be defined, where n_{SK1} and n_{SK2} are the number species combined to build up the shells of the two interacting atoms. The file names must be ordered with respect to the interacting species, so that the name for the second interacting specie is changed first. Above you see an example, where an extended basis with an s^*-orbital was generated by introducing the new species "Si2" and "C2", containing the appropriate s^*-orbital for Si and C, resp., as only orbitals.

In the case of many Slater-Koster files for a certain interaction, the repulsive data is read from the first specified file (e.g. Si-Si.skf, Si-C.skf, C-Si.skf and C-C.skf in the example above). The repulsive interactions in the other files are ignored. The mass for a certain species is read from the first SK-file for its homonuclear interaction.

Non-minimal basis Slater-Koster data may be directly defined in the SK-files in future.

Charge Total charge of the system. Negative value means electron excess.

SpinPolarisation Specifies if and how the system is spin polarised. See p. 16

SpinConstants Specifies the atom type specific constants needed for the spin polarised calculations, in units of Hartrees. For every atomic species in the calculation a property with the type name must be defined, containing the spin coupling constants for that atom. The constants must be ordered with respect to the two shells they couple, so that the index for the second shell must increase faster. For an spd-basis, that would yield the following order:

$$w_{ss}, w_{sp}, w_{sd}, \ldots, w_{ps}, w_{pp}, w_{pd}, \ldots, w_{ds}, w_{dp}, w_{dd}, \ldots$$

Example (GGA parameters for H_2O):

```
SpinConstants = {
  O = {
    # Wss  Wsp    Wps    Wpp
    -0.035 -0.030 -0.030 -0.028
  }
  H = {
    # Wss
    -0.072
  }
}
```

Several standard values of atomic spin constants are given in appendix D. Constants calculated with the same density functional as the SK-files should be used. This input block may be moved to the SK-data definition files in the future.

When using the SelectedShells method for the keyword MaxAngularMomentum, the spin constants are listed as an array of values running over SK1SK2... in the same order as listed for SlaterKosterFiles.

```
SpinConstants = { # not real values, only an example
  Si = {
    # Wss  Wsp    Wss*
    -0.035 -0.030 -0.01
    # Wps  Wpp    Wps*
    -0.030 -0.037 -0.02
    # Ws*s Ws*p   Ws*s*
    -0.01  -0.02  -0.01
  }
```

Eigensolver Specifies which eigensolver to use for diagonalising the Hamiltonian. See p. 17.

Filling Method for occupying the one electron levels with electrons. See p. 17.

SlaterKosterFiles Name of the Slater-Koster files for every atom type pair combination. See 18.

OldSKInterpolation If set to Yes (strongly discouraged), the lookup tables for the non-scc interactions are interpolated with the same algorithm as in the *old* DFTB code. Please note, that the new method uses a smoother function, is more systematic, and yields better derivatives than the old one. This option serves only compatibility purpose, and might be removed in the future.

PolynomialRepulsive Specifies for each interaction, if the polynomial repulsive function should be used. for every pairwise conbination of atoms it should contain a logical value, where Yes stands for the use of a polymial repulsive function and No for a spline. For non-specified interactions the default value No is used

Example:

```
# Use the polynomial repulsive function for Ga-Ga and As-As interactions
# in GaAs
PolynomialRepulsive = {
  Ga-Ga = Yes
  Ga-As = No
  # As-Ga unspecifed, therefore per default set to No
  As-As = Yes
}
```

If you want to apply the same setting for all species pairs, you can specify the appropriate logical value as argumen of the SetForAll keyword:

```
# Using polynomial repulsive functions for all interactions in GaAs
PolynomialRepulsive = SetForAll { Yes }
```

KPointsAndWeights [relative|absolute] Contains the special k-points to be used for the Brillouin-zone integration. See p. 19. For automatically generated k-point grids the modifier should not be set.

OrbitalPotential Specifies which (if any) orbitally dependant contributions should be added to the DFTB energy and band structure. See p. 21.

ReadInitialCharges If set to Yes the first Hamiltonian is constructed by using the charge information read from the file charges.bin.

ElectricField Specifies an external electric field. See p. 21.

Dispersion Specifies which kind of dispersion correction to apply. See p. 22.

DampHShortRange If set to Yes the short range contribution to the SCC interaction coming from atom A and atom B is damped by the factor

$$e^{-\left(\frac{U_{Al}+U_{Bl}}{2}\right)^4 r_{AB}^2}$$

provided that at least one of the pair of atoms is hydrogen. (U_{Al} and U_{Bl} are the Hubbard Us of the two atoms for the l-shell, r_{AB} is the distance between the atoms.) An atom is considered to be a hydrogen atom, if its type name is either "H" or "h".

2.4.1 Mixer

DFTB$^+$ offers currently the charge mixing methods Broyden{} (Broyden-mixer), Anderson{} (Anderson-mixer), DIIS{} (DIIS-accelerated simple mixer) and Simple{} (simple mixer).

Broyden{}

Minimises the error function

$$E = \omega_0^2 \left| G^{(m+1)} - G^{(m)} \right| + \sum_{n=1}^{m} \omega_n^2 \left| \frac{n^{(n+1)} - n^{(n)}}{|F^{(n+1)} - F^{(n)}|} + G^{(m+1)} \frac{F^{(n+1)} - F^{(n)}}{|F^{(n+1)} - F^{(n)}|} \right|^2 ,$$

where $G^{(m)}$ is the inverse Jacobian, $n^{(m)}$ and $F^{(m)}$ are the charge and charge difference vector in iteration m. The weights are given by ω_0 and ω_m, respectively. The latter is calculated as

$$\omega_m = \frac{c}{\sqrt{F^{(m)} \cdot F^{(m)}}}, \tag{2.1}$$

c being a constant coefficient. [14].

The Broyden{} method can be configured using following properties:

MixingParameter	r	0.2
CachedIterations	i	-1
InverseJacobiWeight	r	0.01
MinimalWeight	r	1.0
MaximalWeight	r	1e5
WeightFactor	r	1e-2

MixingParameter Mixing parameter.

CachedIterations Number of charge vectors of previous iterations which should be kept in the memory. Older charge vectors are written to disc. If set to -1, all charge vectors will be kept in the memory. (You should only change its value if you are really short on memory.)

InverseJacobiWeight Weight for the difference of the inverse Jacobians (ω_0).

MinimalWeight Minimal allowed value for the weighting factors ω_m.

MaximalWeight Maximal allowed value for ω_m.

WeightFactor Weighting factor c for the calculation of the weighting factors ω_m in (2.1).

Anderson{}

Modified Anderson mixer. [8]

MixingParameter	r	0.05
Generations	i	4
InitMixingParameter	r	0.01
DynMixingParameters	(2r)*	{}
DiagonalRescaling	r	0.01

MixingParameter Mixing parameter.

Generations Number of generations to consider for the mixing. Setting it too high can lead to linearly dependent sets of equation.

InitMixingParameter Simple mixing parameter used until the number of iterations is greater or equal to the number of generations.

DynMixingParameters Allows to specify different mixing parameters for different levels of convergence during the calculation. These are specified as a list of tolerances below which a given mixing factor is used. If the loosest specified tolerance is reached, the appropriate mixing parameter supersedes that specified in MixingParameter.

DiagonalRescaling Used to increase the diagonal elements in the system of equations solved by the mixer. This can help to prevent linear dependencies occuring in the mixing process. Setting it to too large a value can prevent convergence. (This factor is defined in a slightly different way from Ref. [8]. See the source code for more details.)

Example:

```
Mixer = Anderson {
  MixingParameter = 0.05
  Generations = 4
  # Now the over-ride the (previously hidden) default old settings
  InitMixingParameter = 0.01
  DynMixingParameters = {
    1.0e-2  0.1 # use 0.1 as mixing if more converged that 1.0e-2
    1.0e-3  0.3 # again, but 1.0e-3
    1.0e-4  0.5 # and the same
  }
  DiagonalRescaling = 0.01
}
```

DIIS{}

Direct inversion of the iterative space is a general method to acceleration iterative sequences. The current implementation accelerates the simple mix process.

MixingParameter	r	0.2
Generations	i	6
UseFromStart	l	Yes

MixingParameter Mixing parameter.

Generations Number of generations to consider for the mixing.

UseFromStart Specifies if DIIS mixing should be done right from the start, or only after the nr. of SCC-cycles is greater or equal to the number of generations.

Simple{}

Constructs a linear combination of the current input and output charges as $(1-x)\rho_{\text{in}} + x\rho_{\text{out}}$.

MixingParameter	r	0.05

MixingParameter Coefficient used in the linear combination.

2.4.2 SpinPolarisation

No spinpolarisation ({})

No spin polarisation contributions to the energy or band-structure.

Colinear{}

Colinear spin polarisation in the z direction. The initialization of the calculation is spin restricted.

UnpairedElectrons	i	-
InitialSpin	p	{}

UnpairedElectrons Number of unpaired electrons. (Kept constant during the calculation.)

InitialSpin Initialisation for spin patterns. The default code behavior is an initial spin polarisation of 0.

InitialSpin The initial spin distribution can be set by specifying the spin polarisation on the atoms. For atoms without explicit specification, a spin polarisation of zero is assumed. The InitialSpin property must contain one or more AtomSpin blocks with the following properties:

Atoms	i+lm	-
SpinPerAtom	r	-

Atoms Atoms to specify an initial spin value. The Range{} method can be used to specify an continous interval of atoms.

SpinPerAtom Initial spin polarisation for each atom in this InitialSpin block.

Example:

```
SpinPolarisation = Colinear {
  UnpairedElectrons = 0.0
  InitialSpin = { # want to start from an anti-ferromagnetic ordering
    AtomSpin = {
      Atoms = { 1 }
      SpinPerAtom = -1.0
    }
    AtomSpin = {
      Atoms = { 2 }
      SpinPerAtom = +1.0
    }
  }
}
```

2.4.3 Eigensolver

Currently the following LAPACK 3.0 [1] eigensolver methods are available:

- Standard{}
- DivideAndConquer{}
  (this requires about twice the memory of the other solvers)
- RelativelyRobust{}
  (using the subspace form but calculating all states)

None of them needs any parameters or properties specified.

Example:

```
Eigensolver = DivideAndConquer {}
```

2.4.4 Filling

Fermi{}

Fills the levels according to a Fermi distribution. When using a finite temperature, the Mermin free energy (which the code prints) should be used instead of the total energy. This is given by $E - TS$.

Temperature	r	AdaptFillingTemp = No	0.0

Temperature [*energy*] Electron temperature in energy units. This property is ignored for thermostated MD runs, if the AdaptFillingTemp property of the thermostat had been set to Yes.

Example:

```
Filling = Fermi {
  Temperature [K] = 300
}
```

MethfesselPaxton{}

Produces a Fermi-like distribution but with much lower electron entropy [18]. This is useful for systems that require high electron temperatures (for example when calculating metals)

Temperature	r	AdaptFillingTemp = No	0.0
Order	i		2

Temperature [*energy*] Electron temperature in energy units. This property is ignored for thermostated MD runs, if the AdaptFillingTemp property of the thermostat had been set to Yes.

Order Order of the Methessel-Paxton scheme, the order must be greater than zero, and the 1st order scheme is equivalent to Gaussian filling.

2.4.5 SlaterKosterFiles

There are two different ways to specify the Slater-Koster files for the atom type pairs, explicit specification and using the Type2FileNames{} method.

Explicit specification

Every possible atom type pair connected by a dash must occur as property with the name of the corresponding file as assigned value.

Example (GaAs):

```
SlaterKosterFiles = {
  Ga-Ga = "./Ga-Ga.skf"
  Ga-As = "./Ga-As.skf"
  As-Ga = "./As-Ga.skf"
  As-As = "./As-As.skf"
}
```

If you treat shells from different species as shells of one atom by using the SelectedShells{} keyword in the MaxAngularMomentum{} block, you have to specify more than one file name for certain specie pairs. (For details see the description about the MaxAngularMomentum{} keyword.)

Type2FileNames{}

You can use this method to generate the name of the Slater-Koster files automatically using the element names from the geometry input. You have to specify the following properties

Prefix	s	""
Separator	s	""
Suffix	s	""
LowerCaseTypeName	l	No

Prefix Prefix before the first type name, usually the path.

Separator Separator between the type names.

Suffix Suffix after the name of the second type, usually extension.

LowerCaseTypeName If the name of the types should be converted to lower case. Otherwise they are used in the same way, as they were specified in the geometry input.

Example (for producing the same file names as in the previous section):

```
SlaterKosterFiles = Type2FileNames {
  Prefix = "./"
  Separator = "-"
  Suffix = ".skf"
  LowerCaseTypeName = No
}
```

The Type2FileNames method can not be used, if an extended basis was defined with the SelectedShells method.

2.4.6 KPointsAndWeights

The k-points for the Brillouin-zone integration can be either specified explicitely or using the KLines{} or the SupercellFolding{} methods. If the latter is used the KPointsAndWeights keyword is not allowed to have a modifier.

Explicit specification

For every k-point four real numbers must be specified: The coordinates of the given k-point followed by its weight. By default, the coordinates are specified in fractions of the reciprocal lattice vectors. If the modifier absolute is set for the KPointsAndWeights keyword, absolute k-point coordinates in atomic units are instead expected. The sum of the k-point weights is automatically normalized by the program.

```
KPointsAndWeights = {   # 2x2x2 MP-scheme
  0.25  0.25  0.25    1.0
  0.25  0.25 -0.25    1.0
  0.25 -0.25  0.25    1.0
  0.25 -0.25 -0.25    1.0
}
```

SupercellFolding{}

This method generates a sampling set containing all the special k-points in the Brillouin zone related to points that would occur in an enlarged supercell repeating of the current unit cell. If two k-points in the BZ are related by inversion, only one (with double weight) is used (this permitted by time reversal symmetry). The SupercellFolding{} method expects 9 integers and 3 reals as parameters:

$$\begin{array}{ccc} n_{11} & n_{12} & n_{13} \\ n_{21} & n_{22} & n_{23} \\ n_{31} & n_{32} & n_{33} \\ s_1 & s_2 & s_3 \end{array}$$

The integers n_{ij} specify the coefficients used to build the supercell vectors $\mathbf{A}_i$ from the original lattice vectors $\mathbf{a}_j$:

$$\mathbf{A}_i = \sum_{j=1}^{3} n_{ij}\,\mathbf{a}_j.$$

The reals s_i specify the point in the Brillouin-zone of the super lattice, in which the folding should occur. The coordinates must be given in relative coordinates, in the units of the reciprocal lattice vectors of the super lattice.

The original $l_1 \times l_2 \times l_3$ Monkhorst-Pack sampling [20] for cubic lattices corresponds to a uniform extension of the lattice:

$$\begin{array}{ccc} l_1 & 0 & 0 \\ 0 & l_2 & 0 \\ 0 & 0 & l_3 \\ s_1 & s_2 & s_3 \end{array}$$

where s_i is 0.0, if l_i is odd, and s_i is 0.5 if l_i is even. For the $2 \times 2 \times 3$ scheme, you would write for example

```
# 2x2x3 MP-scheme according original paper
KPointsAndWeights = SupercellFolding {
  2   0   0
  0   2   0
  0   0   3
  0.5  0.5  0.0
}
```

To use k-points for hexagonal lattices which are consistent with the erratum to the original paper [21], you should set the shift for the unique "c" direction, s_3, in the same way as in the original scheme. The s_1 and s_2 shifts should be set to be 0.0 independent of whether l_1 and l_2 are even or odd. So, for a $2 \times 3 \times 4$ sampling you would have to set

```
# 2x3x4 MP-scheme according modified MP scheme
KPointsAndWeights = SupercellFolding {
  2   0   0
  0   3   0
  0   0   4
  0.0  0.0  0.5
}
```

It is important to note that DFTB$^+$ does not take the symmetry of your system explicitely into account. For small high symmetric systems with a low number of k-points in the sampling this could eventually lead to unphysical results. (Components of tensor properties–e.g. forces–could be finite, even if they must vanish due to symmetry reasons.) For those cases, you should explicitely specify k-points with the correct symmetry.

KLines{}

This method specifies k-points lying along arbitrary lines in the Brillouin zone. This is usefull when calculating the band structure for a periodic system. (In that case, the charges should be initialised from the saved charges of a previous calculation with a proper k-sampling. Additionally for SCC calculations the number of SCC cycles should be set to 1, so that only one diagonalisation is done using the initial charges.)

The KLines{} method accepts for each line an integer specifying the number of points along the line segment, and 3 reals specifying the end point of the line segment. The line segments do not include their starting points but their end points. The starting point for the first line segment can be set by specifying a (zeroth) segment with only one point and with the desired starting point as end point. The unit of the k-points is determined by any modifier of the KPointsAndWeights property. (Default is relative coordinates.)

Example:

```
KPointsAndWeights [relative] = KLines {
  1   0.5  0.0  0.0    # Setting (and calculating) starting point 0.5 0.0 0.0
 10   0.0  0.0  0.0    # 10 points from 0.5 0.0 0.0  to  0.0 0.0 0.0
 10   0.5  0.5  0.5    # 10 points from 0.0 0.0 0.0 to 0.5 0.5 0.5
  1   0.0  0.0  0.0    # Setting (and calculating) a new starting point
 10   0.5  0.5  0.0    # 10 points from 0.5 0.5 0.0 to 0.5 0.5 0.0
}
```

2.4.7 OrbitalPotential

Currently only the FLL{} (fully localised limit) form of the LDA+U corrections [22] are implemented. This particular potential lowers the energy of states localized on the specified atomic shells while raising the energy of un-occupied localised states. This particular correction is most useful for lanthanide/actinide f states and some localised d states of transition metals (Ni3d for example).

Implementation	s		-
LConstants	p		-

Implementation Implementation type. Currently only Eschrig's purely onsite density matrix contributions to the chosen functional ("on-site") is implemented [10].

LConstants List of the $U - J$ values for each l-shell of each atom type in the calculation given in Hartree and starting with the first shell (usually s). If an atom type is not specified, the $U - J$ values are assumed to be zero.

```
OrbitalPotential = FLL {
  Implementation = "on-site"
  LConstants = {
    Er = {0.0 0.0 0.0 0.26}
    N = {0.0 0.0}
  }
}
```

2.4.8 ElectricField

This tag contains the specification for an external electric field. Currently only the electric field from point charges is implemented. The ElectricField block currently must contain one or more PointCharges blocks with the following properties:

CoordsAndCharges	(4r)+		-
GaussianBlurWidth	r	non-periodic system	0.0

CoordsAndCharges [*length*] Contains the coordinates and the charge for each point charge (four real values per point charge). A length modifier can be used to alter the units of the coordinates. The charge must be specified in proton charges. (The charge of an electron is -1.)

GaussianBlurWidth [*length*] Specifies the half width σ of the Gaussian charge distribution, which is used to delocalise the point charges. The energy of the coulombic interaction E_C between the delocalised point charge M with charge Q_M and the atom A with charge q_A is weighted by the error function as

$$E_\mathrm{C}(A,M) = \frac{q_A Q_M}{r_{AM}} \operatorname{erf}\left[\frac{r_{AM}}{\sigma}\right],$$

where r_{AM} is the distance between the point charge and the atom.

This delocalisation can only be used for non-periodic systems. A length modifier can be used to specify the unit for σ.

Example:

```
ElectricField = {
  # 1st group of charges, with Gaussian delocalisation
  PointCharges = {
    GaussianBlurWidth [Angstrom] = 3
    CoordsAndCharges [Angstrom] = {
      1.2   3.4  4.5    -1.2
      0.5  -9.2  3.3     3.2
    }
  }
  # 2nd group of charges, no delocalisation (sigma = 0.0)
  PointCharges = {
    CoordsAndCharges [Angstrom] = {
      3.3  -1.2  0.9     9.2
    }
  }
}
```

2.4.9 Dispersion

The dispersion corrects for the van der Waals interaction in your system. Currently only a Slater-Kirkwood type model is implemented as described in Ref. [6]. [1] Therefore, you must set Dispersion = SlaterKirkwood {}. Inside the SlaterKirkwood{} tag, the following property must be specified:

PolarRadiusCharge	(3r)+lm	-

PolarRadiusCharge [*volume,length,charge*] Determines the atomic polarisability, the radius for the damping function and the effective charge for every atom in the system. The atomic polarisabilities must be specified in volume units.

Example for H_2O molecule:

```
Dispersion = SlaterKirkwood {
  # Using Angstrom^3 for volume, Angstrom for length and default
  # unit for charge (note the two separating commas between the units)
  PolarRadiusCharge [Angstrom^3,Angstrom,] = {
    # Polar      Radius      Chrg
```

[1] Please note, that the paper contains two typos. Equation (7) should be read as $C_6^{\alpha\beta} = \frac{2C_6^\alpha C_6^\beta p_\alpha p_\beta}{p_\alpha^2 C_6^\beta + p_\beta^2 C_6^\alpha}$. In equation (9) the contribution from the dispersion should be read as $E_\mathrm{dis} = -\frac{1}{2}\sum_{\alpha\beta} f(R_{\alpha\beta}) C_6^{\alpha\beta} (R_{\alpha\beta})^{-6}$.

```
      0.560000   3.800000   3.150000     # Atom 1: O
      0.386000   3.500000   0.800000     # Atom 2: H
      0.386000   3.500000   0.800000     # Atom 3: H
    }
  }
```

Instead of setting the polarisabilities of the atoms individually, this can be also automatically done depending on their chemical type and the number of their neighbors (hybridisation). In that case, you can use the HybridDependentPol{} method to automate this process. Inside this tag, you must specify for each atom type in your sytem a property with the name of that atom type, and with the following properties inside:

CovalentRadius	r	-
HybridPolarisations	(13r)+	-

CovalentRadius [*length*] Covalent radius for the given atom type. Two atoms are considered to be neighbors, if their distance is not greater than the sum of the covalent radii for their species.

HybridPolarisations [*volume,length,charge*] Specifies for every species the polarisabilities and the damping radii depending on the hybridisation and the effective charge. You have to provide 13 reals. The first six specify the atomic polarisabilities used for an atom of that species, if it has no neighbors, one neighbor, two, three, four or more than four neighbors. The next four entries specify the damping radii for the same cases. The last entry specifies the effective charge (hybridisation independent).

Example:

```
Dispersion = SlaterKirkwood {
  PolarRadiusCharge = HybridDependentPol {
    O = {
      CovalentRadius [Angstrom] = 0.8
      HybridPolarisations [Angstrom^3,Angstrom,] = {
        # All polarisabilities and radii set the same
        0.560 0.560 0.560 0.560 0.560 0.560   3.8 3.8 3.8 3.8 3.8 3.8   3.15
      }
    }
    H = {
      CovalentRadius [Angstrom] = 0.4
      HybridPolarisations [Angstrom^3,Angstrom,] = {
        # Different polarisabilities depending on the hybridisation
        0.386 0.396 0.400 0.410 0.410 0.410   3.5 3.5 3.5 3.5 3.5 3.5   0.8
      }
    }
  }
}
```

Please note, that this automatic procedure sets up the polarisabilities and radii for each atom at the beginning of the calculation. There is no re-evaluation of those quantities during the run (they remain constant). When using the HybridDependentPol{} method, it is recommended to set first the StopAfterParsing keyword in the ParserOptions block to Yes (see p. 25) and inspect the generated polarisabilities, radii and charges for every atom in dftb_pin.hsd. If some fine tuning in the generated values turns out to be necessary, you should replace the hybrid dependent specification in the input file with the generated atom specific values from dftb_pin.hsd, and apply the necessary changes there.

2.5 Options

This block collects some global options for the run.

MullikenAnalysis	l	No
CalculateForces	l	No
WriteEigenvectors	l	No
WriteAutotestTag	l	No
WriteDetailedXML	l	No
WriteResultsTag	l	No
WriteDetailedOut	l	Yes
WriteBandOut	l	Yes
AtomResolvedEnergies	l	No
RestartFrequency	i	20
RandomSeed	i	0
MinimiseMemoryUsage	l	No
WriteHS	l	No
WriteRealHS	l	No

MullikenAnalysis If Yes, Mulliken analysis is carried out, even if it is not needed for the calculation (e.g. non-scc run). For state resolved Mulliken populations see section 3.3.

CalculateForces If Yes, force calculation is carried out, even if it is not needed for the actual calculation (e.g. static geometry calculation).

WriteEigenvectors Specifies, if eigenvectors should be printed in eigenvec.out and eigenvec.bin. For a description of the file format see p. 28.

WriteAutotestTag Turns the creation of the autotest.tag file on and off. (This file can get quite big and is only needed for the autotesting framework.)

WriteDetailedXML Turns the creation of the detailed.xml file on and off. (The detailed.xml file is needed among others by the waveplot utility for visualising molecular orbitals.)

WriteResultsTag Turns the creation of the results.tag file on and off. (That file is used by several utilities processing the results of DFTB$^+$.)

WriteDetailedOut Controls the creation of the file detailed.out. Since this contains the detailed information about the last step of your run, you shouldn't turn it off without good reasons.

WriteBandOut Controls the creation of the file band.out which contains the band structrure in a more or less human friendly format.

AtomResolvedEnergies Specifies whether the contribution of the individual atoms to the total energies should be calculated or not.

RestartFrequency Specifies the interval at which charge restart information should be written to disc for static SCC calculations. Setting it to 0 prevents the storage of restart information.

RandomSeed Sets the seed for the random number generator. The value 0 causes random initialisation. (This value can be used to reproduce earlier MD calculations by setting the initial seed to the same value.)

MinimiseMemoryUsage Tries to minimise memory usage by storing various matrices on disc instead of keeping them in memory. Set it to Yes to reduce the memory requirement for calculations with many k-points or spin polarisation.

WriteHS Instructs the program to build the square Hamiltonian and overlap matrices and write them to files. The output files are hamsqrN.dat and oversqr.dat, where N enumerates the spin channels. For a detailed description of the file format see p. 27.

Note: If either of the options WriteHS or WriteRealHS are set to Yes, the program only builds the matrices, writes them to disc and then stops immediately. No diagonalization, no SCC-cycles or geometry optimization steps are carried out. You can use the ReadInitialCharges option to build the Hamiltonian with a previously converged charge distribution.

WriteRealHS Instructs the program to build the real space (sparse) Hamiltonian and overlap matrices and write them to files. The output files are hamreal.dat and overreal.dat. For a detailed description of the file format see p. 27.

Note: If either of the options WriteHS or WriteRealHS are set to Yes, the program only builds the matrices, writes them to disc and then stops immediately. No diagonalization, no SCC-cycles or geometry optimization steps are carried out. You can use the ReadInitialCharges option to build the Hamiltonian with a previously converged charge distribution.

2.6 ParserOptions

This block contains the options, which are effecting only the behaviour of the HSD/XML parser and are not passed to the main program.

ParserVersion	i	current input version
WriteHSDInput	l	Yes
WriteXMLInput	l	No
IgnoreUnprocessedNodes	l	No
StopAfterParsing	l	No

InputVersion Version number of the input parser, which the input file was written for. If you are using an input file, which was created for an older version of DFTB$^+$, you should set it to the parser version number of that code version. (The parser version number is printed at the beginning of the program run to the standard output.) DFTB$^+$ internally converts the input to its current format. The processed input (written to dftb_pin.hsd) is always in the current format, and the InputVersion property in it is always set to the current parser version.

WriteHSDInput Specifies, if the processed input should be written out in HSD format. (You shouldn't turn it off without really good reasons.)

WriteXMLInput Specifies, if the processed input should be written out in XML format.

IgnoreUnprocessedNodes By default the code stops if it detects unused or erroneous keywords in the input, which probably indicates error(s) in the input. This *dangerous* flag suspends these checks. Use only for debugging purposes.

StopAfterParsing If set to Yes, the parser stops after processing the input and written out the processed input to the disc. It can be used to make sanity checks on the input without starting an actual calculation.

Chapter 3

Output of DFTB$^+$

This chapter contains the description of some output files of DFTB$^+$, where the output format is not self documenting. Unless indicated otherwise, numbers in the output files are given in atomic units (with Hartree as energy unit).

3.1 hamsqrN.dat, oversqr.dat

The files `hamsqrN.dat` and `oversqr.dat` contain the square (folded) Hamiltonian and overlap matrices. The number `N` in the filename `hamrealN.dat` indicates the spin channel. For spin unpolarized calculation it is 1, for spin polarized calculation it is 1 and 2 for spin-up and spin-down, respectively.

Only non-comment lines (lines not starting with "#") are documented:

- Flag for signalizing if matrix is real (`REAL`), number of orbitals in the sytem (`NALLORB`), number of kpoints (`NKPOINT`). For non-periodic (cluster) calculations, the number of kpoints is set to 1.
- For every K-point:
  - Number of the K-point. For molecular (non-periodic) calculations only 1 K-point is printed.
  - The folded matrix for the given K-point. It consists of `NALLORB` lines $\times$ `NALLORB` columns. If the matrix is not complex (`REAL` is `F`), every column contains two numbers (real and imaginary part).

3.2 hamrealN.dat, overreal.dat

The files `hamrealN.dat` and `overreal.dat` contain the real space Hamiltonian and overlap matrices. The number `N` in the filename `hamrealN.dat` indicates the spin channel. For spin unpolarized calculation it is 1, for spin polarized calculation it is 1 and 2 for spin-up and spin-down, respectively.

Note: The sparse format contains only the "lower triangle" of the real space matrix. For more details about the format and how to obtain the upper triangle elements, see Reference [3]. Also note, that for periodic systems the sparse format is based on the *folded* coordinates of the atoms, resulting in translation vectors (ICELL) which look surprising at first glance.

Only non-comment lines (lines not starting with "#") are documented:

- Number of atoms in the system (NATOM)
- For every atom:
  - Atom number (IATOM), number of neighbors including the atom itself (NNEIGH), number of orbitals on the atom (NORB)
- For every neighbor of every atom:
  - Atom number (IATOM1), neighbor number (INEIGH), corresponding image atom to the neighbor in the central cell (IATOM2F), coefficients of the translation vector between the neighbor and its corresponding image (ICELL(1), ICELL(2), ICELL(3)). Between the coordinates of the neighbor $\mathbf{r}_{\text{INEIGH}}$ and the image atom $\mathbf{r}_{\text{IATOM2F}}$ the relation

    $$\mathbf{r}_{\text{INEIGH}} = \mathbf{r}_{\text{IATOM2F}} + \sum_{i=1}^{3} \text{ICELL}(i)\,\mathbf{a}_i$$

    holds, where $\mathbf{a}_i$ are the lattice vectors of the supercell.
  - The corresponing part of the sparse matrix. The data block consists of NORB(IAT1) lines and NORB(IAT2F) columns.

3.3 eigenvec.out, eigenvec.bin

These files contain the eigenvectors from the Hamitonian, stored either as plain text (eigenvec.out) or in the native binary format of your system (eigenvec.bin).

The plain text format contains a list of the values of the components of each eigenvector for the basis functions of each atom. The atom number in the geometry, its chemical type and the particular basis function are listed, followed by the relevant value from the current eigenvector and then the Mulliken population for that basis function for that level. The particular eigenvector, K-point and spin channel are listed at the start of each set of eigenvector data.

The binary format contains the (unique) runId of the DFTB+ simulation which produced the output followed by the values of the eigenvectors. The eigenvector data is ordered so that the individual components of the current eigenvector are stored, with subsequent eigenvectors for that K-point following sequentially. All K-points for the current spin channel are printed in this order, followed by the data for a second channel if spin polarized.

3.4 charges.bin

The file charges.bin contains the orbitally-resolved charges for each atom, ordered as the charges on each orbital of an atom for a given spin channel, then each spin channel and finally over each atom. In later versions of DFTB$^+$ this format may change to include a checksum.

Appendix A

The HSD format

The Human-friendly Structured Data (HSD) format is a structured input format, which can be bijectively mapped onto a subset of the XML-language. Its simplified structure and notation should make it a more convenient user interface than reading and writing XML tags. This section contains a brief overview of the most important aspects of this format.

An input file in the HSD format consists basically of property assignments of the form

```
Property = value
```

where the value value was assigned to the property Property. The value must be one of the following types (detailed description of each follows later on):

- Scalar, such as
  - integer
  - real
  - logical
  - string
- list of scalars
- method
- list of further property assignments

An unquoted hash mark (#) is interpreted as comment sign, everything after it, up to the end of the current line, is ignored by the parser (hash marks inside of quotes are taken as literals not comments):

```
# Entire line with comment
Prop1 = "hell#oo"  # Note, that the first hashmark is quoted!
```

The name of the properties, the methods and the logical values are case insensitive, so the assignments

```
Prop1 = 12
prOP1 = 12
Prop2 = Yes
Prop2 = YES
```

are pairwise identical. Quoted strings (specified either as a value for a property or as a file name), however, are case sensitive.

Due to technical issues, the maximal line length is currently limited to 1024 characters. Lines longer than this are chopped without warning.

If a property, which should only appear once, is defined more than once, the parser uses the *first* definition and ignores all the other occurrences. *Thus specifying a property in the input a second time, does not override the first definition.* (For advanced use the HSD syntax also offers the possibility of conditional overriding/extending of previous definitions. For more details see A.6.)

A.1 Scalars and list of scalars

The following examples demonstrate the assignments with scalar types:

```
SomeInt = 1
SomeInt2 = -3
SomeRealFixedForm = 3.453
SomeRealExpForm = 2.12e-45
Logical1 = Yes
Logical2 = no
SomeString = "this is a string value"
```

As showed above, real numbers can be entered in either fixed or exponential form. The value for logical properties can be either Yes or No (case insensitive). Strings should always be enclosed in quotation marks, to make sure that they are treated as one string and that they are not interpreted by the parser:

```
String1 = "quoted string"
String2 = this value is actually a list of 9 strings  # list of strings!
String3 = "Method { ;"     # This is a string assignment
String4 = Method {         # This is syntactically incorrect, since
                          # it tries to assign a method to String4
```

A list of scalars is created by sequentially writing the scalars separated by one or more spaces:

```
PlottedLevels =  1 2 3
Origin =  0.0 0.0 0.0
ConfirmItTwice =  Yes Yes
SpecieNames =  "Ga" "As"
```

The assignments statements are usually terminated by the end of the line. If the list of the assigned values goes over several lines, it must be entered between curly (brace) brackets. In that case, instead of the line end, the closing bracket will signal the end of the assignment. It is allowed to put a list of scalars in curly brackets, even if it is only one line long.

```
PlottedLevels = {
  1 2 3
}
Origin = { 0.0
0.0 0.0 }
Short = { 1 2 3 }
```

If you want to put more than one assignment in a line, you have to separate them with a semi-colon:

```
Variable = 12; Variable2 = 3.0
```

If a property should be defined as empty, either the empty list must be assigned to it or it must be defined as an empty assignment terminated by a semi-colon:

```
EmptyProperty = {}
EmptyProperty2 = ;
```

Please note, that explicitely specifying a property to be empty is not the same as not having specified it at all. In the latter case, the parser substitutes the default value for that property (if there is a default for it), while in the first case it interprets the property to be empty. If a property without default value is not specified, the parser stops with an appropriate error message.

A.2 Methods and property lists

Besides the scalar values and the list of scalars, the right hand side of an assignment may also contain a method, which itself may contain one or more scalar values or further property assignments as parameters:

```
Diagonaliser = LapackDAC {}      # Method without further params
PlottedLevels = Range { 1 3 }   # Range needs two scalar params
PlottedRegion = UnitCell {      # UnitCell needs a property list
  MinEdgeLength = 1.0           # as parameter
  SomeOtherProperty = Yes
}
```

The first assignment above is an example, where the method on the right hand side does not need any parameters specified. Please note, that even if no parameters are required, the opening and closing brackets after the method are mandatory. If the brackets are missing, the parser interprets the value as a string.

In the second assignment, the method Range needs only two integers as parameters, while for the method UnitCell several properties must be specified. A method may contain either nothing or scalars or property assignments, but never scalars and property assignments together. So the following assignment would be invalid:

```
InvalidSpecif = SomeMethod {
  1 2 3
  Property1 = 12
  "Some strings here"
}
```

Very often a value for the property is represented by a list of further property assignments (as above, but without naming an explicit method beforehand). In that case, the property assignments must be put between curly brackets (property list):

```
Options = {
  SubOption1 = 12
  Suboption2 = "string"
}
```

A.3 Modifiers

Each property may carry a modifier, which changes the interpretation of the assigned value:

```
LatticeConstant [Angstrom] = 12.23
```

Here, the property LatticeConstant possesses the Angstrom modifier, so the specified value will be interpreted to be in Ångström instead of the default length unit. Specifying a modifier for a property which is not allowed to carry one leads to parsing error.

The syntax of the HSD format also allows methods (used as values on the right hand side of an assignment) to carry modifiers, but this is usually not used in the current input structures.

Sometimes, the assigned value to a property contains several values with different units, so that more than one modifiers can be specified. In that case, the modifiers must be separated by a comma.

```
VolumeAndChargePerElement [Angstrom^3,au] = {
   1.2  0.3  # first element
   4.2  0.1  # second element
}
```

You have to specify either no modifier or all modifiers. If you want specify the default units for some of the quantities, you can omit the name of the appropriate modifier, but you must include the separating comma:

```
# Specifying the default unit for the charge. Note the separating comma!
VolumeAndChargePerElement [Angstrom^3,] = {
   1.2  0.3  # first element
   4.2  0.1  # second element
}
```

Specifying not enough or too many modifiers leads to parser error.

A.4 File inclusion

It is possible to include files in an HSD-formatted input by using the <<< and <<+ operators. The former includes the specified file as raw text without parsing it, while latter parses the included text:

```
Geometry = GenFormat {
  <<< "geo_start.gen"
}
Basis = {
  <<+ "File_containing_the_property_definitons_for_the_basis"
}
```

The file included with the <<+ operator must be a valid HSD document in itself.

A.5 Processing

After having parsed and processed the input file, the parser writes out the processed input to a separate file in HSD format. This file contains the internal representation for all properties, which can be specified by the user. Especially, all default values are explicitely set and all automatic definitions (e.g. ranges) are converted to their internal representations.

Assuming the following example as input

```
# Lattice contant specified in Angstrom.
# Internal representation uses Bohr, so it will be converted.
LatticeConstant [Angstrom] = 12.0

# This property is not set, as its commented out, so the
# default value will be set for this (let's assume, it's Yes)
#DoAProperJob = No

# Plotted levels specified as a Range method with parameters 1 3.
# This will be replaced by an explicit listing of the levels
PlottedLevels = Range { 1 3 }
```

the parsed and processed input (written to a special file) should look something like

```
LatticeConstant = 22.676713499923075
DoAProperJob = Yes
PlottedLevels = {
  1 2 3
}
```

This processed form should be free from any conversion factors and automatic definitions. If you want to reproduce your calculation later, you should use this processed input. It should give you identical results, even if the default setting for some properties had been changed in the code.

Since the HSD format is mapped by the parser internally to an XML tree, most codes using this format allow (or hopefully will allow) to dumping out of the processed input in the XML format as well, and to use that later as an input, instead of the HSD formatted input. This is practical for people preferring to work with XML or if the input should be automatically generated by a script.

A.6 Extended format

As stated earlier, if a property, which should be defined only once, occurs more than once in the input, the parser uses per default the first definition and ignores all the others. Sometimes this is not the desired behaviour, therefore, the HSD format also offers the possibility to override properties that were set earlier. This feature can be very useful for scripts which are generate HSD input based on some user provided template. By just appending a few lines to the end of the user provided input the scripts can make sure that certain properties are set correctly. Thus, the script can modify the user input, without having to parse it at all.

The parser builds internally an XML DOM-tree from the HSD input. For every property or method name an XML tag with the same name (but lowercased) is created, which will contain the value of the property or the method. If the value contains further properties or methods, new XML tags are created inside the original one. Shortly, the HSD input is mapped on a tree, whereas the assignment and the containment (equal sign and curly brace) are turned into a parent-child relationships.[1] As an example an HSD input and the corresponding XML-representation is given below:

```
Level0Elem1 = 1                           <level0elem1>1</level0elem1>
Level0Elem2 = { 1 2 3 }                   <level0elem2>1 2 3</level0elem2>
Level0Elem3 = {                           <level0elem3>
  Level1Elem1 = 12                          <level1elem1>12</level1elem1>
  Level1Elem2 = Level2Elem1 {               <level1elem2>
                                              <level2elem1>
    Level3Elem1 = "abcd"                        <level3elem1>"abcd"</level3elem1>
    Level3Elem2 = {                             <level3elem2>
      Level4Elem1 = 12                            <level4elem1>12</level4elem1>
    }                                           </level3elem2>
  }                                           </level2elem1>
}                                           </level1elem2>
                                          </level0elem3>
```

By prefixing property and method names, the default behaviour of the parser can be overridden. Instead of creating a new tag (on the current encapsulation level) with the appropriate name, it will look for the *first occurrence* of the given tag and will process that one. Depending of the prefix character, the tag is processed in the following ways:

+: If the tag exists already, it's value is modified, otherwise the parser stops.

?: If the tag exists already, it's value is modified, otherwise the parser ignores the prefixed HSD construct.

***:** If the tag exists already, it's value is modified, otherwise it is created (and then it's value is modified).

/: If the tag does not exist yet, it is created and modified, otherwise the prefixed HSD construct is ignored.

!: The tag is newly created and modified. If it exists already, the old occurrence is deleted first.

The way the value of the tag is going to be modified, is ruled by the constructs inside the prefixed property or method name. If the parser finds non prefixed constructs here, the appropriate tags are

[1]In the internal tree representation of the HSD input there is no difference between properties and methods, both are just elements capable to contain some value or further elements. The differentiation in the HSD input is artificial and servers the human readability only (equal sign after property names, curly brace after method names),

just added, otherwise the behaviour is determined by the rules above, just acting one level deeper in the tree. The following examples should make this a little bit more clear.

- Changing the value of Level0Elem1 to 3. If the element does not exist, it should be created with the value 3.

  ```
  !Level0Elem1 = 3
  ```

- Changing the value of Level0Elem3/Level1Elem1 to 21 (the slash indicates the parent-child relationship). If the element does not exist, stop with an error message:

  ```
  # Make sure the containing element exists. If yes, go inside, otherwise die.
  +Level0Elem3 = {
    # Set the value of Level1Elem1 or die, if it does not exist.
    +Level1Elem1 = 21
  }
  ```

  Please note, that each tag in the path must be prefixed. Using the following construct instead of the original one

  ```
  # Not prefixed, so it creates a new tag with empty value
  Level0Elem3 = {
    # The new tag doesn't contain anything, so the parser stops here
    +Level1Elem1 = 21
  }
  ```

  would end with an error message. Since Level0Elem1 is not prefixed here, a tag is created for it with an empty value (no children). It does not matter, whether the tag already existed before or not, a new tag is created and appended as the last element (last child) to the current block. Then the parser is processing its value. Due to the +Level1Elem1 directive it is looking for a child tag <level1elem1>. Since the tag was newly created, it does not contain any children, so the parser stops with an error message.

- Create a new tag Level1Elem3 inside Level0Elem3 with some special value. If the tag already exists, replace it.

  ```
  # Modifing the children of Level0Elem3 or dying if not present
  +Level0Elem3 = {
    # Replacing or if not existent creating Level1Elem3
    !Level1Elem3 = NewBlock {
      NewValue1 = 12
    }
  ```

  This example also shows, that the value for the new property can be any arbitrary complex HSD construct.

- Provide a default value "string" for Level0Elem3/Level1Elem2/Level2Elem1/Level3Elem1. If the tag is already present do not change its value.

```
# Modify Level0Elem3 or create it if non-existent
*Level0Elem3 = {
# Modify Level1Elem2 and Level2Elem1 or create them if non-existent
  *Level1Elem2 = *Level2Elem1 {
    # Create Level3Elem1 if non-existent with special value.
    /Level3Elem1 = "string"
  }
}
```

- If Level0Elem3/Level1Elem2 has the value Level2Elem1, make sure that Level3Elem1 in it exists, and has "" as value. If Level1Elem2 has a different value, do not change anything.

```
# If Level0Elem3 is present, process it, otherwise skip this block
?Level0Elem3 = {
  # The same for the next two containers
  ?Level1Elem2 = ?Level2Elem1 {
    # Create or replace Level3Elem1
    !Level3Elem1 = ""
  }
}
```

Appendix B

Unit modifiers

The DFTB$^+$ code uses internally atomic units (with Hartree as the energy unit). The value of every numerical property in the input is interpreted to be in atomic units (au), unless the property carries a modifier.

The allowed modifiers and the corresponding conversion factors are given below.[1] (The modifiers are case insensitive).

Length:	
Angstrom, AA (for Ångström)	0.188972598857892E+01
Meter, m	0.188972598857892E+11
pm	0.188972598857892E-01
Bohr, au	1.000000000000000E+00
Volume:	
Angstrom^3, AA^3	0.674833303710415E+01
meter^3, m^3	0.674833303710415E+31
pm^3	0.674833303710415E-05
bohr^3, au	1.000000000000000E+00
Energy:	
Rydberg, Ry	0.500000000000000E+00
Electronvolt, eV	0.367493245336341E-01
kcal/mol	0.159466838598749E-02
Kelvin, K	0.316681534524639E-05
Joule, J	0.229371256497309E+18
Hartree, Ha, au	1.000000000000000E+00
Force:	
eV/Angstrom, eV/AA	0.194469064593167E-01
Joule/meter, J/m	0.121378050512919E+08
Hartree/Bohr, Ha/Bohr, au	1.000000000000000E+00

[1]The conversion factors listed here were calculated with double precision on i686-linux architecture. Depending on your architecture, the values used there may deviate slightly.

Time:

femtosecond, fs	0.413413733365614E+02
picosecond, ps	0.413413733365614E+05
second, s	0.413413733365614E+17
au	1.000000000000000E+00

Charge:

Coulomb, C	0.624150947960772E+19
au, e	1.000000000000000E+00

Velocity:

au	1.000000000000000E+00
m/s	0.457102857516272E-06
pm/fs	0.457102857516272E-03
AA/ps	0.457102857516272E-04

Appendix C

Description of the gen format

The general (gen) format can be used to describe clusters and supercells. It is based on the xyz format introduced with xmol, and extended to periodic structures. Unlike some earlier implementations of gen, the format should not include any neighbour mapping information.

The first line of the file contains the number of atoms, *n*, followed by the type of geometry. C for cluster (non-periodic), S for supercell in Cartesian coordinates or F for supercell in fractions of the lattice vectors. The supercells are periodic in 3 dimensions.

The second line contains the chemical symbols of the elements present separated by one or more spaces. The following *n* lines contain a list of the atoms. The first number is the atom number in the structure (this is currently ignored by the program). The second number is the chemical type from the list of symbols on line 2. Then follow the coordinates. For S and C format, these are x, y, z in Å, but for F they are fractions of the three lattice vectors.

If the structure is a supercell, the next line after the atomic coordinates contains the coordinate origin in Å (this is ignored by the parser). The last three lines are the supercell vectors in Å. These four lines are not present for clusters.

Example: Geometry of GaAs with 2 atoms in the fractional supercell format

```
2  F
Ga As
1 1     0.0  0.0  0.0
2 2     0.25 0.25 0.25
0.000000     0.000000     0.000000
2.713546     2.713546     0.
0.           2.713546     2.713546
2.713546     0.           2.713546
```

Appendix D

Atomic spin constants

These are suggested values for some atomic spin constants (W values) as given in reference [17], only the first two decimal places of the finite spin constants are numerically significant. These constants may eventually be included in the Slater-Koster files directly. Check the documentation of the Slater-Koster files required for a calculation to decide whether to use the LDA or PBE-GGA spin constants.

W		LDA			PBE		
		s	*p*	*d*	*s*	*p*	*d*
H	*s*	-0.064			-0.072		
C	*s*	-0.028	-0.024		-0.031	-0.025	
	p	-0.024	-0.022		-0.025	-0.023	
N	*s*	-0.030	-0.026		-0.033	-0.027	
	p	-0.026	-0.025		-0.027	-0.026	
O	*s*	-0.032	-0.028		-0.035	-0.030	
	p	-0.028	-0.027		-0.030	-0.028	
Si	*s*	-0.018	-0.013	0.000	-0.020	-0.015	0.000
	p	-0.013	-0.012	0.000	-0.015	-0.014	0.000
	d	0.000	0.000	-0.019	0.002	0.002	-0.032
S	*s*	-0.019	-0.016	0.000	-0.021	-0.017	0.000
	p	-0.016	-0.014	0.000	-0.017	-0.016	0.000
	d	0.000	0.000	-0.010	0.000	0.000	-0.080
Fe	*s*	-0.013	-0.009	-0.003	-0.016	-0.012	-0.003
($3d^7 4s^1$)	*p*	-0.009	-0.011	-0.001	-0.012	-0.029	-0.001
	d	-0.003	-0.001	-0.015	-0.003	-0.001	-0.015
Ni	*s*	-0.009	-0.009	-0.003	-0.016	-0.012	-0.003
	p	-0.009	-0.010	-0.001	-0.012	-0.022	-0.001
	d	-0.003	-0.001	-0.017	-0.003	-0.001	-0.018

Appendix E

Dispersion constants

The following table contains recommended dispersion constants for some elements. The values have been tested for biological systems, C, N, O and H predominantly for DNA [6]. If you would like to calculate different systems or you're looking for other elements, check references [19] and [15]. The values of the atomic polarisabilities and cutoffs are given for zero, one, two, three, four and more than four neighbors.

Element	Polarisability [Å^3]	Cutoff [Å]	Chrg	Note
O	0.560 0.560 0.000 0.000 0.000 0.000	3.8 3.8 3.8 3.8 3.8 3.8	3.15	
N	1.030 1.030 1.090 1.090 1.090 1.090	3.8 3.8 3.8 3.8 3.8 3.8	2.82	
C	1.382 1.382 1.382 1.064 1.064 1.064	3.8 3.8 3.8 3.8 3.8 3.8	2.50	
H	0.386 0.386 0.000 0.000 0.000 0.000	3.5 3.5 3.5 3.5 3.5 3.5	0.80	
P	1.600 1.600 1.600 1.600 1.600 1.600	4.7 4.7 4.7 4.7 4.7 4.7	4.50	PO_4 only
S	3.000 3.000 3.000 3.000 3.000 3.000	4.7 4.7 4.7 4.7 4.7 4.7	4.80	S, not SO_2

Appendix F

Publications to cite

The following publications should be considered for citation, if you are publishing any results calculated by DFTB$^+$.

DFTB$^+$ code	[3]
non-SCC DFTB	[23], [24]
SCC DFTB	[7]
Spin polarisation	[16]
QM/MM coupling (external charges)	[11], [5]
Van der Waals interaction (dispersion)	[6]
DFTB+U	[13]

Bibliography

[1] http://www.netlib.org/lapack/index.html. 17

[2] H. C. Andersen. Molecular dynamics at constant pressure and/or temperature. *J. Chem. Phys.*, 72:2384, 1980. 9

[3] B. Aradi, B. Hourahine, and Th. Frauenheim. DFTB+, a sparse matrix-based implementation of the DFTB method. *J. Phys. Chem. A*, 111(26):5678, 2007. 27, 42

[4] H. J. C. Berendsen, J. P. M. Postma, W. F. Van Gunsteren, A. Dinola, and J. R. Haak. Molecular-dynamics with coupling to an external bath. *J. Chem. Phys.*, 81(8):3684–3690, 1984. 9

[5] Q. Cui, M. Elstner, T. Frauenheim, E. Kaxiras, and M. Karplus. Combined self-consistent charge density functional tight-binding (scc-dftb) and charmm. *J. Phys. Chem. B*, 105:569, 2001. 42

[6] M. Elstner, P. Hobza, T. Frauenheim, S. Suhai, and E. Kaxiras. Hydrogen bonding and stacking interactions of nucleic acid base pairs: a density-functional-theory based treatment. *J. Chem. Phys.*, 114:5149, 2001. 22, 41, 42

[7] M. Elstner, D. Porezag, G. Jungnickel, J. Elsner, M. Haugk, T. Frauenheim, S. Suhai, and G. Seifert. Self-consistent-charge density-functional tight-binding method for simulations of complex materials properties. *Phys. Rev. B*, 58:7260, 1998. 42

[8] V. Eyert. A comparative study on methods for convergence acceleration of iterative vector sequences. *J. Comp. Phys.*, 124:271, 1996. 15

[9] T. Frauenheim, G. Seifert, M. Elstner, T. Niehaus, C. Kohler, M. Amkreutz, M. Sternberg, Z. Hajnal, A. Di Carlo, and S. Suhai. Atomistic simulations of complex materials: ground-state and excited-state properties. *J. Phys. Cond. Matter*, 14(11):3015–3047, Mar 2002. 2

[10] M. J. Han, T. Ozaki, and J. Yu. O(N) LDA+U electronic structure calculation method based on the nonorthogonal pseudoatomic orbital basis. *Phys. Rev. B*, 73:045110, 2006. 21

[11] W. Han, M. Elstner, K. J. Jalkanen, T. Frauenheim, and S. Suhai. Hybrid scc-dftb/molecular mechanical studies of h-bonded systems and of n-acetyl-(l-ala)n-n'-methylamide helices in water solution. *Int. J. Quant. Chem.*, 78:459, 2000. 42

[12] S. C. Harvey, R. K. Z. Tan, and T. E. Cheatham. The flying ice cube: Velocity rescaling in molecular dynamics leads to violation of energy equipartition. *J. Comp. Chem.*, 19(7):726–740, 1998. 9

[13] B. Hourahine, S. Sanna, B. Aradi, C. Kohler, T. Niehaus, and Th. Frauenheim. Self-interaction and strong correlation in DFTB. *J. Phys. Chem. A*, 111(26):5671, 2007. 42

[14] D. D. Johnson. Modified broyden's method for accelerating convergence in self consistent calculations. *Phys. Rev. B*, 38:12807, 2003. 14

[15] Y. K. Kang and M. S. Jhon. Additivity of atomic static polarizabilities and dispersion coefficients. *Theoretica Chimica Acta*, 61:41, 1982. 41

[16] C. Köhler, G. Seifert, and T. Frauenheim. Density-functional based calculations for fe(n), (n$\leq$32). *Chem. Phys.*, 309:23, 2005. 42

[17] Christof Köhler. *Berücksichtigung von Spinpolarisationseffekten in einem dichtefunktional-basierten Ansatz*. PhD thesis, Department Physik der Fakultät fur Naturwissenschaften an der Universität Paderborn, 2004. http://ubdata.uni-paderborn.de/ediss/06/2004/koehler/. 40

[18] M. Methfessel and A. T. Paxton. High-precision sampling for brillouin-zone integration in metals. *Phys. Rev. B*, 40:3616, 1989. 17

[19] K. J. Miller. Additivity methods in molecular polarizability. *J. Am. Chem. Soc.*, 112:8533, 1990. 41

[20] H. J. Monkhorst and J. D. Pack. Special points for brillouin-zone integrations. *Phys. Rev. B*, 13:5188, 1976. 20

[21] H. J. Monkhorst and J. D. Pack. "special points for brillouin-zone integrations"–a reply. *Phys. Rev. B*, 16:1748, 1977. 20

[22] A. G. Petukhov, I. I. Mazin, L. Chioncel, and A. I. Lichtenstein. Correlated metals and the LDA+U method. *Phys. Rev. B*, 67:153106–4, 2003. 21

[23] D. Porezag, T. Frauenheim, T. Köhler, G. Seifert, and R. Kaschner. Construction of tight-binding-like potentials on the basis of density-functional theory: Application to carbon. *Phys. Rev. B*, 51:12947, 1995. 42

[24] G. Seifert, D. Porezag, and T. Frauenheim. Calculations of molecules, clusters, and solids with a simplified lcao-dft-lda scheme. *Int. J. Quant. Chem.*, 58:185, 1996. 42

Index