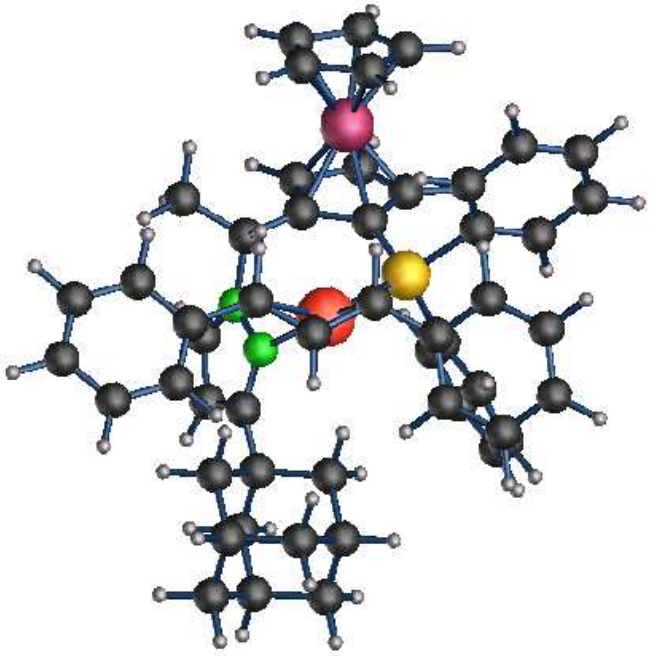
Manual for the Projector Augmented Wave Method

Version 2.0



Peter E. Blöchl, Clausthal University of Technology
(November 15, 2008)

1

[1]The title picture shows the a chiral Pd complex with P,N ligands, a highly enantio-selective catalyst for allylic amination [**?**].

Contents

1 Initial remarks

I am frequently changing the CP-PAW program. Therefore, it is unavoidable that this description is incomplete in some places, that it may lists options that are no longer supported, or that it contains errors. Please let me know if you find something unclear or even incorrect. Other users will be grateful.

I have attempted to make the program test for inconsistencies of the input data, such as the selection of conflicting options. You will find that the program stops in such cases and tries to advise the user concerning what has gone wrong. New users are particularly creative in the combinations of options they choose. If you run into problems that the program does not detect, for example if it crashes without giving a useful message, please let me know *before* you get used to working around the problem. Your input is particularly valuable in making the the code more secure to use.

Any other suggestions on how to improve the clarity of this description or the code are most welcome.

Programs become outdated nearly as quickly as newspapers. Note that the executable of this code may also have an expiration date. The code will print a warning about a month before it expires. Please make sure that you know the expiration date when you receive an executable. If the license expires, discuss with your contact person about obtaining an extension of the license and a new executable program.

Note that the CP-PAW code including all related material is copyrighted. You require a valid license to execute it.

2 What is the projector augmented wave method?

The projector augmented wave method [**?**] is an all-electron electronic structure method, which allows accurate electronic structure calculations and ab-initio molecular dynamics simulations on the basis of density functional theory.

What is all that?

Density functional theory (DFT) [**?**, **?**]. Density functional theory describes the ground state of a many-electron system by electrons that do not interact other than through an effective potential that depends on the electron density. It is based on an exact theorem, which specifies that such a description, based on the electron density rather than on the electronic many-particle wave function, be rigorously possible for ground states. In practice the density functional, which also defines

the effective potential as a functional of the density, is not exactly known. However, highly successful approximations have been found. In contrast to Hartree-Fock calculations, density functional theory explicitly treats electron correlation. The accuracy is typically comparable to that of MP2 calculations, i.e. only a few kcal/mol [**?**, **?**].

Ab-initio molecular dynamics (AIMD) [**?**, **?**, **?**] is an extension of traditional electronic structure methods which has been invented in 1985 by Roberto Car and Michele Parrinello [**?**]. The best way to think of it is as a series of electronic structure calculations, one for each time slice, for always different atomic positions. From one time slice to the next, the atomic positions are changed according to Newton's equations of motion $M_i \ddot{R}_i = F_i$. Here M_i is the mass of a nucleus, R_i the position, F_i the force acting on the nucleus as calculated from the electronic structure, and the double-dots stand for the second time derivative. Self-consistent iterations at each time step are avoided by a dynamical evolution of the wave functions, and thus simulations of several picoseconds are possible, which is sufficient to simulate directly chemical reactions and diffusion with low barriers or at high temperatures.

Whereas the basic idea of ab-initio molecular dynamics is to perform real-time and finite temperature simulations, it can be used like a traditional electronic structure method – using a friction to "cool" the temperature to zero – and it has been combined with statistical approaches to study processes with large barriers.

The **projector augmented wave method (PAW)** [**?**] has been developed by the author in response to the invention of the ab-initio molecular dynamics approach. Whereas the latter was based on the plane wave pseudopotential approach, a new method was needed to enhance the accuracy and computational efficiency of the approach and to provide the correct wave functions, rather than the fictitious wave functions provided by the pseudopotential approach. The PAW method describes the wave function by a superposition of different terms: There is a plane wave part, the so-called pseudo wave function, and expansions into atomic and pseudo atomic orbitals at each atom. On one hand, the plane wave part has the flexibility to describe the bonding and tail region of the wave functions, but used alone it would require prohibitive large basis sets to describe correctly all the oscillations of the wave function near the nuclei. On the other hand, the expansions into atomic orbitals are well suited to describe correctly the nodal structure of the wave function near the nucleus, but lack the variational degrees of freedom for the bonding and tail regions. The PAW method combines the virtues of both numerical representations in one well-defined basis set.

Of course, one does not want to make two electronic structure calculations –

one using plane waves and one with atomic orbitals –, and thus double the computational effort. Therefore, the PAW method does not determine the coefficients of the "atomic orbitals" variationally. Instead, they are unique functions of the plane wave coefficients. It is possible to break up the total energy, and most other observable quantities, into three almost independent contributions: one from the plane wave part and a pair of expansions into atomic orbitals on each atom. The contributions from the atomic orbitals can be broken down furthermore into contributions from each atom, so that strictly no overlap between atomic orbitals on different sites need to be computed.

The PAW method is in principle able to recover rigorously the density functional total energy, if plane wave and atomic orbital expansions are complete. This provides us with a systematic way to improve the basis set errors. The present implementation uses the frozen core approximation, even though the general formalism allows extensions in this respect. It provides the correct densities and wave functions, and thus allows us to calculate hyperfine parameters etc. Limitations of plane wave basis sets to periodic systems (crystals) can easily be overcome by making the unit cell sufficiently large and decoupling the long-range interactions [**?**]. Thus the present method can be used to study molecules, surfaces, and solids within the same approach.

3 How the code is built

This section is about programming philosophy. I write this up because I myself often wonder when I use other software why something is done in a particular fashion. Therefore, I shall present my reasoning here. Feel free to skip this section if you wish.

The program is written in an **OO (object-oriented)** manner. This means that it consists of agents (objects) that perform certain operations or provide the selected information. Each agent holds the data needed to its job, or it can request the data it needs from other agents. This is a major software strategy, widely used today, which allows the programmer to hide certain details, he need not really worry about at the present level of programming. It also allows him to assemble the code from little "boxes", which are easy to maintain and enhance.

As part of the OO design, the program uses its own low level object library, which customizes a number of common operations, such as interproceessor communication, error handling, file handling, tree-linked-list structures for intuitive IO, periodic table, constants, DFT functionals, tracing, timing, string handling,

and a few more. These low-level objects are rather unspecific to the PAW code, and can be used in combination with self-developed analysis tools. (However, they are still subject to the license agreement for the PAW code.)

The language used is **FORTRAN90**. FORTRAN90 is itself not an OO language such as C++ and Java, and thus limits the possibilities in this respect. However, it has a number of advantages for number crunching, such as good compilers and, for my taste, a natural and easily comprehensible way to write mathematics. Compared to FORTRAN77 it is a significant advance towards the OO features of C++ or Java because it incorporates features such as dynamic memory allocation, derived data types (structures), operator overloding, modules, etc. The option of using templates has been implemented using a self-made preprocessor. FORTRAN90 is largely compatible with FORTRAN77, so you need to pick up only a few new things if you want to work with the existing tools.

The program allows **parallel processing** using MPI (Message Passing Interface) [**?**]. It is (almost) scalable in central memory and CPU time. The scalar version program is identical to the parallel version with the exception that a dummy interface is used instead of MPI.

The program relies heavily on linear algebra packages such as LAPACK, BLAS and FFTW. These libraries takes care of basic computations such as matrix multiplications and FFTs (Fast Fourier transforms). A large fraction of the total CPU time is spent in these routines. In this way the code development concentrates on algorithmic developments and not, for example, on how to optimize a FFT. The latter will be done by experts who continually adapt this library to modern computer architectures.

One programming principle I try to follow is the German engineering motto **"Fewer Parts!"**. For the user this implies that he will find few instances where two options provide the same functionality. I hope that the limitations will be offset by clarity.

The program uses **Hartree atomic units**, that is $\hbar = e = m_e = 4\pi\epsilon_0 = 1$, and Cartesian coordiantes. Angles are handled in radian ($2\pi \text{ rad} = 360 \text{ deg}$). The conversion to other units is often done during printing. The conversion factors are provided by a particular agent (see section **??**), which is based on the values of fundamental physical constants recommended in 1986 by CODATA (Committee on Data for Science and Technology of the International Council of Scientific Unions) [**?**].

The program divides work into **three steps: simulation – analysis – visualization**. The program consists of the simulation code, which is the core of CP-PAW, and a number of tools used to analyze the results. The simulation code

calculates energies, densities, wave functions, trajectories, etc., and writes the resulting data into files. These files are then read by tools, which collect the desired information, and bring it into the desired form, typically as another file that can be read by your graphics utilities.

Why this three-layer strategy?

Analysis is both an iterative process and an art: when you find something interesting and want to take a closer look. Sooner or later, you will probably want to make your own analysis tools, because you have found a way to understand your data that nobody has thought of before. Or you want examine a property, which nobody has tried to calculate. Or you have a unique visualizer and need an interface for it. And because you want to discuss this particular result with your collegues at the upcoming coffee break, you choose the quick-and-sloppy approach. You do not want to do this inside the simulation code, because you dare not jeopardize its operation.

In contrast to analysis, simulation is computationally expensive. Therefore, it is desirable to archive as much data from the simulation as possible, and be able to go back later and look at it again. Hence the simulation writes most data rather unspecifically in machine-readable form (which saves a considerable amount of disk space). The data are written in a simple format so that it is easy to read them into the analysis tools.

Visualization is also separated from analysis, because many tools exist and the choice is a matter of taste and wealth. The analysis tools that come with the CP-PAW code will write data formats for the visulatization tools that we currently use. (Those are currently the IBM Dataexplorer [**?**] for 3D representations of the wave functions or trajectories, xmgrace [**?**] for x-y plots, and CryMolCAD[**?**] for the analysis of the atomic structure.) However, you can easily change the format to adapt them to your preferred graphics programs.

Of course you can see the most important information, such as total energies and one-particle energies directly while the simulation is proceeding. You may also contact other users about further analysis tools and exchange them. If you wish to write a graphical interface that hides all three steps from the user, feel free to do so.

4 Input data structure

The input data of the simulation code and the tools uses a format that attempts to be both general and intuitive. Logically, the data are arranged in a tree structure

similar to pull-down menues or directory trees in unix. It allows one to hide options of the program that the user may not be interested in, which then can be handled by default values, and it avoids the unnecessary restrictions of formatted input. The PAW library has objects that can handle such structures easily, so that it is widely used for data input for both the simlation code and the analysis tools. The following section shall make you familiar with the general layout of input data.

4.1 Syntax rules for the input data

Input data are structured as nested data blocks, each identified by a key word. Each data block may contain other data blocks and data. Data are again specified by keywords.

The following simple expample shall illustrate the structure

```
!FIRSTBLOCK
  DATA=5
  !SECONDBLOCK OTHERDATA=T !END
  !SECONDBLOCK OTHERDATA=F !END
  !THIRDBLOCK_OFF TEXTDATA='THIS IS A TEXT' !END
!END
!EOB
```

The indentation and the arrangement of the data is arbitrary. The only requirements are that data and keywords be separated by blanks or line breaks, and that their sequence observes the logical tree structure described below.

Every block starts with a block identifier and ends with the string "!END". The identifier starts with an exclamation mark such as !FIRSTBLOCK. (Of course one can use any other name instead of "FIRSTBLOCK". The recognized block key words are described in the later sections of this manual.) The last block must be followed by "!EOB" (End-Of-Buffer) to indicate the end of all data blocks. Each block, together with its data and any contained subblocks, can be made invisible by appending "_OFF" to the block name as done for !THIRDBLOCK in the example. Data blocks may occur multiply such as !SECONDBLOCK, if specified in this manual. The order in which the data blocks are given is irrelevant as long as their hierarchy is observed. An exception are multiple data or datablocks, where the order in which they occur may or may not be significant. Each data refers only to its block. For example, the two occurrences of OTHERDATA= are dif-

ferent even though their key words are identical, because they are within different subblocks.

The general format of input data is a key word, as specified in this manual separated by an "=" sign from the input data. The input data can be simple data or arrays. Higher dimensional arrays are treated like in Fortran with the first dimension incremented first, such as a(1,1) a(2,1) a(1,2) a(2,2). The data are read in free format.

Note that a mistyped key word makes the data or entire branches of the tree structure invisible. There is no way to warn the user if some key words have not been recognized. If the same key word is given several times in a given data block, without being specified as a multiple, only the first occurrence is recognized. The same is true for data blocks. It is therefore recommended that the protocol be checked to determine, whether all data have been used as intended.

The type of data is determined as follows:

- if a data contains a single or double apostrophe, it is assumed to be of character type,
- if a data contains an open parenthesis, it is assumed to be of complex type,
- if a data is either T, F, .true. or .false. it is assumed to be of type logical,
- if a data contains a period it is assumed to be of type real,
- otherwise, the data is assumed to be of type integer.

These rules are checked in the order given here. It is allowed to precede a data item by an integer and a multiplication sign such as $3 * 0.$, which is shorthand for 0. 0. 0. (In some cases, one is allowed to provide both integer and real. Therefore, if you provided a number of the wrong type and, against your expectation, the program does not complain, a conversion has been done by rounding the number to the nearest integer or a simple type conversion from integer to real has occured.)

The description uses the following notation. Data blocks are indicated by a frame. The enclosed name contains the entire hierarchy of data blocks. Only the last one must be specified on the date file. However, this data block must be enclosed by the higher level data blocks. The key words relate to the individual data. If we cross reference data, we use the entire hierarchy of parent blocks, followed by a colon and the data keyword.

4.2 Extended notation for atoms

The PAW code refers to atoms by name. No two atoms must be given the same name. Arbitrary names, given a length limit, are allowed. However, the recommended notation choses the first two characters as the element symbol, with blanks replaced by underscores, and the remainder is a number. Starting the name with the element symbol allows to exploit some default settings. The names are defined in the structure input file and can be referred to in other specifications and by the PAW tools.

In solids, it will be neccesary to distinguish between periodic images of the same atom. An example is a bondlength constraint between two atoms, of which one is a periodic image of a original atom. In that case we use in some cases an extended naming convention.

In the extended notation the atom name has the form "*name*:*ijk*", where *name* is the name of the atom in the original atom cell. and *i*,*j*,*k* are the three translations along the lattice vectors. The latter can be single digit integer numbers with or without preceeding sign. To give a few examples "H_2:100", "H_2:+1+0+0", "H_2:10-1".

Note, that the translations do *not* specify a unit cell that contains that atom. For example, an atom that moves about will never change its lattice translations, despite traversing several unit cells. Rather the atoms are considered first as a initial cluster, which forms a lattice by repetition and lattice translations. There is no restriction that the atoms of the initial cluster are confined in one unit cell.

The extended notation will be used increasingly. However, please refer in all cases to the manual before using the extended notation. It is also mandatory that original atom names are chosen such that no confusion can occur. A simple rule that avoids confusion, is not to use colons in atom names.

5 Before starting... ($PAWDIR and $ROOT)

In this manual, I frequently refer to $PAWDIR and $ROOT. They are explained in the following:

5.1 The PAW directory

$PAWDIR is the name of the directory where the PAW code is installed.

In the directory $PAWDIR you will find the following directories

- `src` contains the source codes. The sources for the analysis tools are located in a subdirectory `Tools`.
- `doc` contains the manual and installation notes
- `bin` will be created if it does not exist. It contains the executable codes
- `dx` contains files required by the dataexplorer to visualize wave functions, densities and atomic structures

If the program is set up properly, you will have an environment variable `$PAWDIR`, which refers to the directory where the PAW code is installed. You can try it with "`echo $PAWDIR`", which will print the PAW directory. Then you can go to the PAW directory using "`cd $PAWDIR`", and see its contents using "`ls -l`".

5.2 The project data structure

All files for a given system should be located in one directory, and it is recommended that this directory contain only that project. All filenames belonging to that system will have a common root, which we denote in a Unix-like fashion by $ROOT (the value of the variable "`ROOT`"). The individual files are distinguished by their extensions. For example, you can look into the directory "`$PAWDIR/Sample`". Here you will find a number of files that begin with "`h2co`". In this case, the root for the filenames ($ROOT) is "`$PAWDIR/Sample/h2co`". Note that te root is always specified by its absolute path. The filenames have extensions of the kind "`.cntl`", "`.strc`", "`.rstrt`", etc. Each extension defines a particular role of the file within the project. For example, in the file "`$ROOT.cntl`" one defines which operations the simulation code shall do, in the file "`$ROOT.strc`" one defines the molecule or crystal to be studied, and the file "`$ROOT.rstrt`" holds, among other data, the instantaneous atomic coordinates and electron wave functions that the simulation code needs to know in order to continue the simulation.

You can make exceptions to this rule and define the filenames explicitly. This may be useful, for example, if you wish to store the bulkiest file elsewhere, because you have run out of space on the disk, where you store the projects.

Note that, unlike $PAWDIR, your environment does not know what $ROOT is. Therefore, you should use $ROOT in this manual as a placeholder for the full root name, unless the variable "`ROOT`" is explicitly defined.

6 The simulation code

6.1 How to perform a simulation

6.1.1 Input files

In order to run the simulation code, two input files have to be prepared:

1. The **structure input file** (or often simply called STRC), describes the system to be studied. Here you provide information about which atoms you wish to simulate, what their specifics are, whether you wish to simulate a molecule or a crystal and so on.

2. The **control input file** (sometimes abbreviated as CNTL) describes what the program shall do, such as optimizing the wave functions, relaxing the atoms or simlating at finite temperature.

3. The **setup files** contain element-specific information about the augmentation, i.e. partial waves and projector functions, and the core density. Currently, these files are supplied by the author. The setups depend via the core density on the functional used in their construction. It is also important to be consistent regarding the division of electrons into core and valence electrons.

After the simlation has finished, the program writes a restart file, which holds the actual wave functions and the positions of the atoms. This file holds all the necessary information to continue a simulation from the point where you finished. It is important to keep the restart file and the structure file if you wish to return later to what you have done.

6.1.2 Execute the simulation code

The simulation code is executed using the command

`paw_fast.x` *controlfile* `1>err 2>&1 &`

where *controlfile* is the name of the control input file described in the next section. With `1>err` one redirects the print statements to the file out. If the file existed before it is overwritten. This information is normally irrelevent and is used for development purposes. However, it is crucial for tracing errors. With `2>&1` one

redirects also error messages into the same file, namely `err`. Thus also error messages be it from PAW or from the system are written in the same directory. The last & simply sends the job into the background. That is you can continue working in the same window while the job is running.

`paw_fast.x` is simply the name of the PAW executable. There will be several executables. If you are using a parallel computer using MPI you will need an executable that starts with `ppaw` instead of `paw`. The executable with `fast` is the one compiled with all reasonable optimization flags for the compiler. For all normal purposes this one should be used. In addition there may be an executable with `dbg` instead of `fast` for debugging purposes and one with nothing, with no special flags, which can be used to find errors induced by optimization.

You can also obtain information about the executable

`--help`	print help information
`-h`	print help information
`?`	print help information
`--version`	print version information
`--parmfile`	print parameter file used for compilation

The version information shall allow to correlation the executable to a well-defined version of the source code. The parameter file in addition provides information about what libraries are linked and which compilation flags have been used etc. This information is important to track bugs in the code.

Sometimes it is convenient to write a little wrapper shell-script for the execution command.

```
#!/bin/sh
#=================================================
# sample doit file to execute the simulation code ==
#=================================================
#                                  define the rootname
ROOT=/home/user/Tree/Testrun/H2CO/h2co
#                                execute paw simulation
paw_fast.x ${ROOT}.cntl 1>${ROOT}.err 2>&1
```

This file would be called `$ROOT.doit`, which is also the command to execute the code. Such a wrapper avoids some unnecessary typing if you run the job repeatedly. It can easily be modified to execute the simulation code several times with different control files.

What does the wrapper do? Let us discuss it line by line. (Comment lines, i.e. those starting with a hatch sign, are not discussed.)

1. The first line `#!/bin/sh` simply specifies that the wrapper is a shell script

2. The second line defines a variable `ROOT`. In the following commands $ROOT is shorthand for `/home/user/Tree/Testrun/H2CO/h2co`. Thus one can use the same wrapper for different simulations by simply changing the value of `ROOT`.

   The last line executes the the simulation code using the control file `$ROOT.cntl`.

In order to track the simulation one can issue the command

```
tail -f $ROOT.prot
```

which continually prints the lines written to the protocoll file also to the screen. Another way to trace the simulation is the command `paw_show` described in section **??**.

6.1.3 Terminate the simulation

The execution can be stopped before regular completion by creating a so-called exit file

```
touch exitfile
```

where *exitfile* is the name of the exit file. The standard name is $ROOT`.exit`, but this name can be changed in the control input file.

As an alternative to setting the exit file, the CP-PAW code also contains a signal trap. The command

```
kill -30 PID
```

where *PID* is the process id number, causes the program to terminate the execution after the next iteration. Note, however, that this command can sometimes crash the code, namely if the code is writing to a file at the moment you issue the command. Its use is therefore not recommended, and it may even be removed from future versions. The process id number can be obtained either from the command `top` or from the command `ps -elf |grep paw`.

6.2 The control input file “CNTL”

The control file is responsible for “what to do” in the simulations. These data are generally unspecific to the system.

6.2.1 Examples for the control input file “CNTL”

The following is a particularly simple example for a control input file. This file can be used for getting started.

```
!CONTROL
  !GENERIC START=T NSTEP=100 !END
  !FOURIER EPWPSI=30 CDUAL=2 !END
  !PSIDYN
    !AUTO FRIC(+)=0.3 FACT(+)=1.
          FRIC(-)=0.3 FACT(-)=0.97 !END
  !END
!END
!EOB
```

The following example illustrates a few more options that give a quick impression of the choices available. The detailed description of all options is given in the next section.

```
!CONTROL
 !GENERIC START=T DT=10 NSTEP=100 NWRITE=100 !END
 !DFT     TYPE=2 !END
 !FOURIER EPWPSI=30 CDUAL=2 !END
 !PSIDYN  STOP=T FRIC=0.005 RANDOM=0.1
          MPSI=1000 MPSICG2=0.5
   !AUTO  FRIC(-)=0.3 FACT(-)=0.97
          FRIC(+)=0.5 FACT(+)=0.97 !END
   !THERMOSTAT_OFF STOP=T FRIC=0
          T[K]=100 FREQ[THZ]=10 !END
 !END
 !RDYN    STOP=T FRIC=0 RANDOM[K]=0
   !AUTO  FRIC(+)=0.5 FACT(+)=0.97
          FRIC(-)=0.3 FACT(-)=0.97 !END
   !THERMOSTAT_OFF STOP=T FRIC=0
```

```
          <EKIN>=0.02 FREQ[THZ]=50 !END
 !END
 !FILES
   !FILE  ID='EXIT' EXT=F NAME='~/EXIT' !END
 !END
 !ANALYSE
   !HYPERFINE ATOM='H_1' EFG=T ISOMERSHIFT=T
              FERMICONTACT=T ANISOTROPIC=T   !END
   !DENSITY FILE='./density.ev' TYPE='TOTAL'
         DR=0.4 OCC=T DIAG=T CORE=F !END
   !WAVE FILE='./wavefunction.wv'
         DR=0.4 B=10 K=1 S=1 IMAG=F !END
   !POTENTIAL FILE='./potential.wv'
         dr=0.4 !END
 !END
!END
!EOB
```

6.2.2 Argument keywords for the control input file "CNTL"

6.2.3 !CONTROL

Rules: optional
Description: defines the operations performed on the system; largely independent of the system

6.2.4 !CONTROL!FILES

Rules: optional
Description: specifies the file names that deviate from the standard values

ROOT

rootname. Files defined as extension will have this name combined with the extension. All files connected as default are defined as extensions.

Type:	character
Rules:	optional
Default:	string preceding the '.cntl' ending of the control input file

6.2.5 !CONTROL!FILES!FILE

Rules:	optional, multiple
Description:	Specifies one file

ID

identifier for the file; options are:

'PROT'	protocol; extension: `.prot` monitors the simulation.
'STRC'	structure input file; extension: `.strc` defines atoms, structure, electron occupations etc.
'CNTL'	control input file; extension: `.cntl` used to control the simulation.
'RESTART_IN'	input restart file; extension `.rstrt` instantaneous coordinates
'RESTART_OUT'	output restart file; extension `.rstrt` see 'RESTART_IN'
'EXIT'	exit file; extension: `.exit` simulation terminates if exit file exists
'PDOS'	projected density of states; extension: `.pdos` used by the paw_dos tool.
'CONSTRAINTS'	constraint report; extension: `_constr.report`
'POSITION_TRAJECTORY'	atomic positions; extension `_r.tra` trajectory used by the paw_tra tool.
'BANDS_TRAJECTORY'	one-particle energies; extension `_r.tra` (not yet used)

Type: character
Rules: mandatory
Default: none

NAME

filename. Can be the relative file name or an extension to the PAW "root". Standard output can be specified by NAME='stdout' and EXT=.false.

Type: character
Rules: mandatory
Default: none

EXT

.true.: NAME specifies the extension only/ .false.: full name
Type: logical
Rules: optional
Default: .false.

6.2.6 !CONTROL!GENERIC

Rules: optional
Description: general data that do not fit into other blocks

START

T: start with random wave functions, and atomic positions from file ”STRC”
F: wave functions and atomic positions are taken from restart file, unless specified otherwise
Type: logical
Rules: optional
Default: F

NSTEP

number of time steps
Type: integer
Rules: optional
Default: 100

DT

time step Δ in a.u. (1 a.u.$\approx$ 0.024 fs)
Type: real
Rules: optional
Default: 10.0

NWRITE

Every “NWRITE” time steps, the program writes detailed information into the protocoll file and it updates the restart file. Note that writing the restart file is time consuming.

Type: integer
Rules: optional
Default: 100

TRACE

provides trace information when entering or leaving subroutines under trace control. Used for debugging purposes.

Type: logical
Rules: optional
Default: .false.

ENDIAN

defines whether unformatted files are written in little endian as typical on intel computers or in big endian as on ibm computers. The value can be ’little’,’intel’,’big’,’ibm’. The first two values have identical meaning and the last two have identical meaning. It is only functional using certain compilers (absoft).

Type: character
Rules: optional
Default: little

RUNTIME

A soft stop is initialized after the given time has elapsed since start of the code. Runtime is specified as a three element vector containing (hours,minutes,seconds). The three elements of the vector are internally converted into seconds and added up.

Type: integer
Rules: optional
Default: ∞

AUTOCONV

The autopilot defines a strategy to vary the friction for various dynamical variables, to test the convergence and to terminate the optimization loop. The decision to terminate is taken if the energy remains within an energy window of 10^{-5} H for autoconv time steps.

Type: integer
Rules: optional
Default: 20

RSTRTTYPE

allows to cut down the information on the restart file. Option RSTRTTYPE='STATIC' only stores one set of wave functions. This reduces the amount of diskspace by nearly a factor of two. This option should not be used in a dynamical simulation, because, the velocity of the wave functions will be set to zero.

Type: character
Rules: optional
Default: NONE

6.2.7 !CONTROL!DFT

Rules: optional
Description: selects density functional parameterization; default is Perdew-Zunger parameterization of the Ceperley Alder quantum Monte Carlo calculation

TYPE

density functional parameterization. possible values are:

(1) Perdew-Zunger parameterization [**?**] of Ceperley-Alder [**?**];
(2) Perdew-Wang 91 parameterization [**?**] of Ceperley-Alder [**?**];
(3) Local exchange (X_α with α=2/3);
(4) X-alpha (alpha=0.7);
(6) Local exchange and Becke gradient correction [**?**] for exchange;
(7) Perdew-Wang LSD [**?**] and Becke gradient correction for exchange [**?**];
(71) Perdew-Zunger[**?**] and Becke-88 gradient correction[**?**];
(8) like (7) + Perdew-86 gradient correction for correlation [**?**];
(81) Perdew-Zunger[**?**] + Becke gradient correction[**?**] + Perdew86 gradient correction for correlation[**?**];
(9) like (7) + Perdew-Burke Ernzerhof correlation [**?**];
(10) Perdew-Burke-Ernzerhof (PBE)exchange and correlation [**?**]; Almost identical to PW91 GGA[**?**] but simpler formulation.
(11) RPBE exchange and correlation, includes Hammer's correction[**?**] to Perdew-Burke-Ernzerhof exchange and correlation [**?**];

Type: integer
Rules: optional
Default: 1

VDW

Adds van der Waals Interactions[**?**]. The van der Waals interaction is described by an interatomic pair-potential that has a long-ranged r^{-6} behavior and which is cut of smoothly at short distances.

Type: logical
Rules: optional (Untested)
Default: F

6.2.8 !CONTROL!FOURIER

Rules: optional

Description: Plane wave cutoffs $E_{PW} = \frac{1}{2}G^2_{max}$ for wave functions and charge density. A cutoff of 30 Ry=15 H for the wave function and CDUAL=2 is sufficient for most applications. In order to account for all plane wave components in the density, the plane wave cutoff for the density should in prinicple be four times the cutoff for the wave function. (For low cutoffs, even CDUAL$>$4 may be required since the exchange and correlation functional is nonlinear, and therefore the energy may not necessarily decrease monotonically as the basis set size is increased with a fixed CDUAL. In practice this is rarely a problem.)

EPWPSI

plane wave cutoff for the wave functions in Rydberg (1 Ry=0.5 a.u.). All plane waves up to a maximum kinetic energy equal to the plane wave cutoff are considered. Note that the plane wave cutoff refers only to the plane wave part of the basis functions.

Type: real
Rules: optional
Default: 30.

EPWRHO

plane wave cutoff for the charge density in Rydberg (1 Ry=0.5 a.u.)

Type: real
Rules: optional
Default: 4*EPWPSI

CDUAL

EPWRHO=CDUAL*EPWPSI; The default cdual=4 produces the correct result; cdual=2 is often a good choice.

Type: real

Rules: optional; this value is overwritten by any occurance of “EPWRHO”
Default: 4

EPWBUCKET

Used mostly for cell dynamics. See section **??** for an explanation of the sawtooth behavior. Parameter E_B in Ry specified as follows: If specified an additial potential energy term $\sum_{G,n} f_n \langle \tilde{\Psi}_n | G \langle B(G) \langle G | \tilde{\Psi}_n \rangle$, is added top the total energy where $B(G) = c_B \theta(\frac{1}{2}G^2 - E_B)(G - \sqrt{2E_B})^2$. When this term is set to a fixed value, the plane wave convergence is artificially accelerated, which is important to evaluate stresses. The energy converged this way corresponds however not to equal to the energy converged without this term, but corresponds to the total energy obtained with a plane wave cutoff slightly higher than EPWBUCKET.
Type: real
Rules: optional;mandatory of BUCKETPAR is specified
Default: none

BUCKETPAR

Parameter c_B see also description of EPWBUCKET above.
Type: real
Rules: optional; mandatory, if EPWBUCKET is specified
Default: 0.

6.2.9 !CONTROL!PSIDYN

Rules: optional; default uses default values for the electron dynamics.
Description: Parameters used to control the dynamics of the wave functions. The wave function dynamics is governed by the equation

$$m_\Psi |\ddot{\tilde{\Psi}}_n\rangle = -\frac{\partial E}{\partial \langle \tilde{\Psi}_n|} - m_\Psi |\dot{\tilde{\Psi}}_n\rangle \alpha - \sum_m |\tilde{\Psi}_m\rangle \Lambda_{mn},$$

where α can be tuned by a Nosé thermostat or the parameters below. The equation of motions are integrated using the Verlet algorithm [**?**]. The friction parameter α is converted into a parameter $c_\alpha = \alpha\Delta/2$, which can range from 0 to 1. A value of $c_\alpha = 0$ indicates frictionless dynamics, whereas a value of $c_\alpha = 1$ indicates steepest descent. Δ is the time step. A discussion of the virtues of friction dynamics can be found in Ref. [**?**]

STOP

if STOP=true start with zero velocity of the wave functions
Type: logical
Rules: optional
Default: F

MPSI

mass m^0_Ψ for the wave function dynamics. The wave function mass is an operator of the form

$$\hat{m}_\psi = \sum_{\vec{G}} |\vec{G}\rangle m^0_\psi (1 + c\vec{G}^2)\langle\vec{G}|$$

The coefficient c is set in MPSICG2.
The G-dependent wave function mass is used to reduce the otherwise very large frequency of the wave function components with large $|\vec{G}|$. For the large G-components of the auxiliary wave functions we can approximate the effective potential by a constant, which makes the Hamiltonian diagonal in $|\vec{G}\rangle$. This allows to estimate the frequency from

$$\hat{m}^0_\psi (1 + c\vec{G}^2)\ddot{\Psi}_G = \left[\frac{\vec{G}^2}{2} + v_{eff}\right]\Psi_G$$

Type: real
Rules: optional
Default: 10 Δ^2 (for Δ see "!CONTROL!GENERIC:DT")

MPSICG2

high-G enhancement c for the wavefunction mass. The fictitious electron mass for the wave function dynamics is G-dependent $m_\Psi(G) = m_\Psi(1 + cG^2)$. A recommended value is $c = 0.5$. A reasonable value is $\frac{1}{2}(\frac{x}{2\pi})^2\frac{\Delta^2}{m_\Psi}$, where x is the fraction of shortest oscillation period for the electron dynamics and the time step. The stability criterion of the Verlet algorithm requires $x > 3$. The default uses this expression with $x = 10$. For proper mass renormalization (discounting the nuclear mass by the effective mass of the wave function cloud) the values of !STRUCTURE!SPECIES:PS<G2> and !STRUCTURE!SPECIES:PS<G4> need to be specified.
Type: real
Rules: optional
Default: $\frac{1}{2}(\frac{10}{2\pi})^2\frac{\Delta^2}{m_\Psi}$

FRIC

constant friction c_α for the wave function dynamics
Type: real
Rules: optional, not used if "!CONTROL!PSIDYN!AUTO" is selected.
Default: 0.0

SAFEORTHO

chooses the way the orthogonality constraints for the wave functions is enforced. If SAFEORTHO=T is selected the traditional way is used, which results in a strictly energy-conserving dynamics. If SAFEORTHO=F a new method is used which converges at eigenstates of the Hamiltonian, but is energy conserving only if the states are sufficiently close to eigenstates. This option allows to calculate excited states or to use the Mermin functional[**?**] in order to introduce Fermi occupation numbers. It must be used in connection with dynamical occupations such as the Mermin functional and the tetrahedron method for Brillouin-zone integration.
Type: logical
Rules: optional
Default: T

SWAPSTATES

Only used with SAFEORTHO=F. For SWAPSTATES=F, the approximate eigenstates are ordered in increasing energy expectation values. This is used for optimizing electronic and atomic structures. Thus the given occupations refer to increasing one-particle energies in the final state.
If, in a photochemical reaction with a state crossing with fixed occupations, the system shall remain on the upper branch of the energy surface, choose SWAPSTATES=F. If the system shall follow the crossing to the lower branch of the energy surface, choose SWAPSTATES=T.
In combination with the Mermin functional the following happens at a band crossing at the Fermi level.

- SWAPSTATES=F: the wave functions change their character, which results in wave function heating. The time scale of the conversion of the state depends on matrix elements and on the fictitious mass of the wave functions. This process is therefore rather uncontrolled. The occupations remain approximately constant. The crossing is treated like an avoided crossing.

- SWAPSTATES=T: the wave function maintains its character, but the occupations dynamically change from occupied to unoccupied and vice versa. If the croosing is an avoided crossing, the wave functions are reordered only if the if the conversion of the character of the wave functions is sufficiently retarded, that a band crossing of the one-particle energies takes place.

Type: logical
Rules: optional, not tested
Default: F

6.2.10 !CONTROL!PSIDYN!AUTO

Rules: optional

Description: Automatic annealing procedure for wave function dynamics. (overwrites setting of !CONTROL!PSIDYN:FRIC.) For optimizing the electronic structure the default set has been proven very useful. The friction switches between a lower and an upper value depending on whether the energy decreases or increases. The upper and lower friction values are each scaled in each time step. With this option, the program will stop if the total energy changes during 20 iterations by less than 10^{-5} Hartree. This does not necessarily imply convergence of the annealing step. It can happen, in particular near convergence, that the friction jumps from step to step between the lower and the upper value. In this case the system is not converged, even if the total energy is virtually constant, and a constant friction should be used.

FRIC(-)

start value for the friction factor c_α used for decreasing energy

Type: real
Rules: optional
Default: 0.3

FACT(-)

factor multiplied with the friction factor FRIC(-) in each step that reduces the energy

Type: real
Rules: optional
Default: 0.97

FRIC(+)

start value for the friction factor c_α used for increasing energy

Type: real
Rules: optional
Default: 0.3

FACT(+)

factor multiplied with the friction factor FRIC(+) in each step that reduces the energy

Type: real
Rules: optional
Default: 1.0

MINFRIC

minimum friction used

Type: real
Rules: optional
Default: 0.

6.2.11 !CONTROL!PSIDYN!THERMOSTAT

Rules: optional

Description: (presently in test phase:
Thermostat for the wave functions. The thermostat acts like a Nose-Hoover thermostat[**?**, **?**, **?**] , but it does not aim at constructing a canonical ensemble. Instead it starts acting only when the wave function kinetic exceeds the target temperature estimated for Born-Oppenheimer kinetic energy.[**?**] The latter obtained from the instantaneous atomic velocities, the wave function masses and the values of !STRUCTURE!SPECIES:PS<G2> and !STRUCTURE!SPECIES:PS<G4> provided the structure input file. Do not use this thermostat without providing PS<G2> and PS<G4>. If the thermostat kicks in, it reshuffles kinetic energy from the wave functions into the atomic motion. The target kinetic energy is subtracted from the wave function kinetic energy in the protocol. A friction on the thermostat shall be used to avoid instabilities. The thermostat shall be used only if the atoms are moving.
Old version: Nosé thermostat for constant temperature ensemble for the electrons [**?**]. The target kinetic energy should be a little (about 50%) larger than

$$\langle E_{kin}^{\Psi} \rangle_T = \sum_R 2 \frac{m_\Psi}{M_R} \tilde{T}_R k_B T, \qquad (1)$$

where $\tilde{T}_R = \sum_n \langle \tilde{\Psi}_n | - \nabla^2/2 | \tilde{\Psi}_n \rangle$ is the kinetic energy of the pseudo wave functions of the corresponding isolated atom and T is the physical temperature of the nuclei.

<**EKIN**>

average kinetic energy of the wave functions. Do not use in the new version.

Type: real
Rules: optional
Default: 0.01

T[K]

New version: atom temperature in Kelvin.
Old version: average kinetic energy of the wave functions divided by Boltzmann's constant.

Type:	real
Rules:	optional
Default:	see <EKIN>

PERIOD

Oscillation period of the Nosé variable in a.u.

Type:	real
Rules:	optional
Default:	see !CONTROL!PSIDYN!THERMOSTAT:FREQ[THz]

FREQ[THZ]

Frequency of the Nosé variable in THz.

Type:	real
Rules:	optional; not used if !CONTROL!PSIDYN!THERMOSTAT:PERIOD present
Default:	100.

FRIC

Friction acting on the Nosé variable. Can have values between zero and one. The largest value before overdamping is $\frac{4\pi}{period}$, where period is that of the thermostat variable defined in this section.

Type:	real
Rules:	optional
Default:	0.

STOP

set velocity for Nosé variable to zero in the first iteration

Type: logical
Rules: optional
Default: F

OLD

selects the “old” implementation of the Nose thermostat
Type: logical
Rules: optional
Default: F

6.2.12 !CONTROL!RDYN

Rules: optional; default is fixed atomic positions
Description: parameters used to control the dynamics of the atomic positions. The atomic dynamics is governed by the equation

$$M_R\ddot{R} = -\frac{\partial E}{\partial R} - M_R\dot{R}\alpha,$$

where α can be tuned by a Nosé thermostat or the parameters below. The friction parameter α is converted into a parameter $c_\alpha = \alpha\Delta/2$, which can range from 0 to 1. A value of $c_\alpha = 0$ indicates frictionless dynamics, whereas a value of $c_\alpha = 1$ indicates the steepest descent. Δ is the time step.
Remark: Do not use this option before the wave functions are optimized, because unreasonable forces will result from an incorrect electron distribution.

STOP

start with zero velocity for the atomic positions
Type: logical
Rules: optional
Default: F

START

Take initial structure from structure control file instead of the restart file.

Type: logical
Rules: optional
Default: F

RANDOM[K]

adds random velocities at the start; the velocity distribution corresponds to a temperature in Kelvin. Can be used to bring the system quickly to a finite temperature. Note, however, that the electron wave functions cannot immediately follow this sudden kick. They will start to lag behind, and deviate from the Born Oppenheimer surface. A small friction acting on the wave functions will allow the wave functions to recover.
Type: real
Rules: optional
Default: 0

FRIC

constant friction c_α
Type: real
Rules: optional, not used if "AUTO" is selected.
Default: 0.0

NONEGEFRIC

Experimental option! Avoids negative friction on the atoms. It is recommended for structure optimizations. It must not be used in connection with thermostats. The friction can become negative via the compensation for the drag by the wave function cloud. A friction on the wave functions implicitely causes an effective friction on the atoms. This effective friction is compensated by adding a negative friction on the atoms. In this case the total friction on some atoms can become negative so that the atoms cannot come to rest. [Future developents shall automatically exclude simultaneous use together with thermostats.]
Type: logical
Rules: optional
Default: false

USEOPTFRIC

Experimental option! Estimates the optimum friction for optimization of the atomic structure. as

$$a_{opt} = \Delta \sqrt{\frac{\dot{\vec{R}}\dot{\vec{F}}}{\dot{\vec{R}}\mathbf{M}\dot{\vec{R}}}}$$

In addition a floating average of this friction value is used. The formula is derived from a projection os the Lagrangian onto a one-dimensional motion and the assumption of a harmonic oscillator. The force constant can then be estimated from the change of the forces between two time steps. See Sec. **??** for further information.

Type: logical
Rules: optional
Default: false

6.2.13 !CONTROL!RDYN!AUTO

Rules: optional
Description: Automatic annealing procedure for atomic motion (overwrites setting of !CONTROL!RDYN:FRIC). See the description for !CONTROL!PSIDYN!AUTO. Note that frequent switching between the lower and upper friction indicates a problem that can even result in a heating up of the electrons. In this case, switch to constant friction.

FRIC(-)

start value for the friction factor c_α used for decreasing energy

Type: real
Rules: optional
Default: 0.0

FACT(-)

factor multiplied with the friction factor FRIC(-) in each step that reduces the energy

Type: real
Rules: optional
Default: 1.0

FRIC(+)

start value for the friction factor c_α used for increasing energy

Type: real
Rules: optional
Default: 0.01

FACT(+)

factor multiplied with the friction factor FRIC(+) in each step that reduces the energy

Type: real
Rules: optional
Default: 1.0

MINFRIC

minimum friction used

Type: real
Rules: optional
Default: 0.

6.2.14 !CONTROL!RDYN!WARMUP

Rules: optional, not tested
Description: Heating procedure for QM nuclei. Applies a series of sinusoidal heat pulses in orthogonal directions to provide optimal excitation of the atomic system without dislodging the wavefunctions from the BO surface. This is an approximate procedure.

TARGET_TEMP

Target temperature that should be achieved in Kelvin. Due to approximate nature of the algorithm, it will not be reached exactly in most cases.

Type: real
Rules: mandatory
Default: none

NPULSES

The required temperature increase specified in TARGET_TEMP will be applied in NPULSES steps.

Type: integer
Rules: mandatory
Default: none

PULSE_LENGTH

Each of the NPULSES heat pulses will stretch over PULSE_LENGTH time steps.

Type: integer
Rules: mandatory
Default: none

6.2.15 !CONTROL!RDYN!THERMOSTAT

Rules: optional

Description: Nosé thermostat [**?**, **?**, **?**] for creating a constant temperature ensemble for the ions. When using this thermostat, the friction acting indirectly on the atoms, and that results from a friction on the wavefunctions, is corrected for by an opposing force $C_i \dot{R}_i f_\Psi$ acting on the atoms. f_Ψ is the friction acting on the wave functions, which is either fixed or dynamically tuned by a wave function thermostat

This thermostat has the added option of a friction for faster equilibration. The friction must be zero to obtain a canonical ensemble.

T[K]

temperature of the ions in Kelvin
Type: real
Rules: optional
Default: 293.15 (=room temperature=20^{o}C)

<**EKIN**>

average kinetic energy of the atoms
Type: real
Rules: optional
Default: see T[K]

PERIOD

Oscillation period of the Nosé variable in a.u.
Type: real
Rules: optional
Default: see !CONTROL!RDYN!THERMOSTAT:FREQ[THz]

FREQ[THZ]

frequency of the Nosé variable in THz.
Type: real
Rules: optional; not used if !CONTROL!RDYN!THERMOSTAT:PERIOD is present
Default: 10.

FRIC

friction on the Nosé variable. Sensible values lie between 0 and 1; FRIC=0 will give the original Nosé thermostat.
Without friction, the thermostat tends to induce large oscillations when the temperature is changed. This oscillations may break your system in the high-temperature peaks, and they decrease only slowly to their physical amplitude. In this case it is recommended to use a friction of about 0.01 until the system is about stationary at the new temperature. Once the stationary state has been reached, the friction must be removed, so that the system can approach the canonical ensemple

Type: real
Rules: optional
Default: 0

STOP

velocity for Nosé variable is set to zero in the first iteration

Type: logical
Rules: optional
Default: F

6.2.16 !CONTROL!MERMIN

Rules: optional

Description: Describes the treatment of variable occupations of the one-electron energy levels. It allows to describe the occupations as dynamical variables in the framework of the Mermin functional[**?**], which includes finite temperature effects. The total energy in this case is the free energy and includes an entropic term $-TS$ for the partially occupied orbitals at finite temperatures. The occupations will converge to the Fermi distribution function: $f_i = (1 + e^{-(\epsilon_i - \mu)/kT})^{-1}$. The eigenvalues are the $\epsilon_n = \langle \Psi_n | H | \Psi_n \rangle$.
In the adiabatic mode of operation also the improved tetrahedron method [**?**]can be used, which is the recommended option for static ground state calculations of metallic systems.
Use this option (variable occupations) only with eigenstates of the Hamitlonian, i.e. with !CONTROL!PSIDYN:SAFEORTHO=.false. Otherwise the results are unpredictable.
It is important to initialize the occupations once using START=.TRUE.. (If occupations are close to zero or one, they will take a very long time to move away from these values.) if Start=.false., total charge and total spin is obtained from the restart file and not from the structure input file
Remark the dynocc object has been largely rewrittem 23-28.12.06. Please report any suspicious observations.

T[K]

Temperature of the electrons in Kelvin (enters the Fermi distribution function)
Type: real
Rules: optional
Default: 1000

M

mass for the occupation dynamics. Should be larger than 200.
Type: real
Rules: optional

Default: 300

ADIABATIC

chooses quasi-adiabatic mode for occupations. New energy levels are introduced that approach the energy levels $\epsilon = \frac{\partial E}{\partial f_n}$.in a retarded fashion. The occupations are determined for these retarded energies. Note that in the previous formula the energy also includes the fictitious kinetic energy. This mode of operation does not produce an energy conserving dynamics. It is recommended for static calculations. The current implementation of the tetrahedron method for Brillouin zone integration only works with adiabatic=T.

Type: logical
Rules: optional
Default: .false.

RETARD

The energy levels used for the Brillouin-zone integration in the adiabatic mode approach the actual enegy levels as $\sim \exp(-\frac{1}{r}\frac{t}{\Delta})$.

Type: real
Rules: optional, used only if !ADIABATIC=T
Default: 0.

TETRA+

if true, the Brillouin zone integration is performed with the improved version of the tetrahedron method[**?**]. If false, the Brillouin zone integration is done by sampling and the Mermin functional. The tetrahedron method is the recommended method for metallic systems. However it is not fully variational and restricted to the quasi-adiabatic mode. The tetrahedron method usually causes problems when only 1 k-point is used, because this results in delta-function peaks of the density of states. In that case it is better to use TETRA=false and a finite temperature.

Type: logical
Rules: optional, requires ADIABATIC=T
Default: .false.

STOP

set velocity for occupation dynamics to zero in the first iteration

Type: logical
Rules: optional
Default: F

STARTTYPE

defines how the initial occupations are determined.

'X': Occupations are read from restart file.

'E': energies are read from restart file. Occupations are constructed using the fermi distribution function.

'N': States are filled with $T = 0$ assuming absolutely flat bands ordered according to increasing energy.

Type: character
Rules: optional; 'X' incompatible with ADIABATIC=T
Default: 'N'

START

Restart occupations, total spin and total charge from Lagrange multipliers for wave function orthogonalization, which approximate the one-particle energies. If start=.false., the information is read from the restart file.

Type: logical
Rules: optional, not ised if !CONTROL!MERMIN:STARTTYPE is set.
Default: F

MOVE

Propagate occupations. Otherwise the occupations are kept frozen.

Type: logical
Rules: optional
Default: T

FRIC

friction for the occupation dynamics
Type: real
Rules: optional
Default: 0.0

FIXQ

conserve total charge
Type: logical
Rules: optional
Default: .true.

FIXS

conserve total spin
Type: logical
Rules: optional
Default: .false.

EFERMI[EV]

chemical potential of the elctrons in eV; only used if fixq=.false.
Type: real
Rules: optional
Default: 0.0

MAGFIELD[EV]

external magnetic field in eV; only used if fixs=.false.
Type: real
Rules: optional
Default: 0.0

6.2.17 !CONTROL!MERMIN!DIAL

Rules: optional
Description: allows to linearly vary the thermodynamic variables of the Mermin functional such as temperature, (not implemented: total charge,total spin, chemical potential, magnetic field). The initial Value is taken from the actual setting of the object

NAME

Specifies the variable to be modified. Can be TEMPERATURE[K]; CHARGE[E]; SPIN[HBAR].
Type: character
Rules: mandatory
Default: none

RATE

Specifies the rate of change of the variabble. The unit of the variable is that indicated by the name. The time scale unit is a.u.
Type: real
Rules: mandatory
Default: none

FINAL

Specifies the final value of the modified quantity. After this value is reached, the quantity remains constant.
Type: real
Rules: mandatory
Default: none

6.2.18 !CONTROL!CELL

Rules: optional; default is fixed unit cell, and no stress calculation
Description: This option is experimental!. Invokes pressure calculation, and Parrinello-Rahman dynamics of the unit cell, if requested.

MOVE

Dynamical evulution of the unit cell.
Type: logical
Rules: optional
Default: T

STOP

reset velocities to zero
Type: logical
Rules: optional
Default: F

FRIC

friction parameter
Type: real
Rules: mandatory if !control!cell:move=.true.
Default: none

M

mass for unit cell dynamics
Type: real
Rules: mandatory if !control!cell:move=.true.
Default: none

P

external pressure
Type: real
Rules: optional
Default: 0.0

CONSTRAINTTYPE

Constraints on the cell dynamics. Allowed values are the following:

- 'ISOTROPIC' only allows isotropic expansions and contractions.
- 'NOSHEAR' allows anisotropic expansions and contractions but no shearing.
- 'FREE' does not restrict the dynamics other than to remove the rotation from the stress tensor.
- 'NOSTRESS' sets the stresses to zero. (Only used for tests).

Type: character
Rules: optional
Default: 'FREE'

6.2.19 !CONTROL!QM-MM

Rules: optional
Description: controls parameters for the dynamics of the molecular mechanics environment specified in the structure file.
Remark: Switch this option on only after the wave functions are optimized, because unreasonable electrostatic forces will act on the environment atoms if the electron density is incorrect

STOP

sets initial velocities of the environment to zero.
Type: logical
Rules: optional
Default: false

FREEZE

keeps environment atoms except link atoms frozen

Type: logical
Rules: optional
Default: false

ADIABATIC

Relaxes the environment in every time step to zero (in contrast to damped or undamped dynamics)
Type: logical
Rules: optional
Default: true

FRIC

Friction (FRIC=1 corresponds to steepest decent; FRIC=0 to zero friction)
Type: real
Rules: optional
Default: 0.0

RANDOM[K]

randomizes the atomic positions (not yet implemented)
Type: real
Rules: optional
Default: 0.0

MULTIPLE

makes environment multiple times faster (multiple time steps). For very large values the QM reaction center experiences free energy forces from the environment.
Type: integer
Rules: optional
Default: 1

6.2.20 !CONTROL!QM-MM!AUTO

Rules: optional
Description: Automatic annealing procedure for the dynamics of the MM-environment. See also remarks for !CONTROL!PSIDYN!AUTO

FRIC(+)

start value for the friction factor c_α used for increasing energy
Type: real
Rules: optional
Default: 0.3

FACT(+)

factor multiplied with the friction factor FRIC(+) in each step that reduces the energy
Type: real
Rules: optional
Default: 1.0

FRIC(-)

start value for the friction factor c_α used for decreasing energy
Type: real
Rules: optional
Default: 0.3

FACT(-)

factor multiplied with the friction factor FRIC(-) in each step that reduces the energy
Type: real
Rules: optional
Default: 0.97

MINFRIC

minimum friction used
Type: real
Rules: optional
Default: 0.

6.2.21 !CONTROL!QM-MM!THERMOSTAT

Rules: optional
Description: Nosé thermostat [**?**, **?**, **?**] to for constant temperature ensemble for the QM-MM environment atoms. It has the added option of a friction for faster equilibration and faster acceleration from small velocities. To regain the original thermostat leave the default FRIC=0.

T[K]

temperature of the ions in Kelvin
Type: real
Rules: optional
Default: 293.15 (=room temperature=20^{o}C)

<**EKIN**>

average kinetic energy
Type: real
Rules: optional
Default: see T[K]

FREQ[THZ]

frequency of the Nosé variable in THz. This is an alternative to !CONTROL!RNOSE:PERIOD
Type: real
Rules: optional; is overwritten by PERIOD
Default: see PERIOD

FRIC

friction on the Nosé variable. Sensible values lie between 0 and 1; FRIC=0 will give the original Nosé thermostat, while FRIC=1 is used for equilibration and does not produce a canonical ensemble.
Type: real
Rules: optional
Default: 0

STOP

velocity for Nosé variable is set to zero in the first iteration
Type: logical
Rules: optional
Default: F

6.2.22 !CONTROL!COSMO

Rules: optional
Description: controls parameters for the dynamics of the continuum description of solvent. Essentially identical to the COSMO method by Klamt and Schuurmann, with modifications that allow consistent forces[**?**].

STOP

sets initial velocities of the environment to zero
Type: logical
Rules: optional
Default: false

START

restarts screening charges with value zero
Type: logical
Rules: optional
Default: false

M

Mass for the surface charge dynamics
Type: real
Rules: optional
Default: 1000.

MULTIPLE

Number of time steps of the surface charge dynamics per PAW timestep
Type: integer
Rules: optional
Default: 1

ADIABATIC

chooses optimizations of screening charges in each time step
Type: logical
Rules: optional
Default: false

ETOL

energy convergence criterion for ADIABATIC=T. Inner loop terminates, if estimated energy deviation from minimum is smaller than etol.
Type: logical
Rules: if ADIABATIC=T either ETOL or QTOL or both must be specified. Not used for ADIABATIC=F.
Default: 10^{-7} if qtol is not specified

QTOL

charge convergence criterion for ADIABATIC=T. Inner loop terminates, if estimated charge deviation from minimum , i.e. $|\vec{q} - \vec{q}_{min}|$ is smaller than etol.

Type: logical
Rules: if ADIABATIC=T either ETOL or QTOL or both must be specified. Not used for ADIABATIC=F.
Default: charge criterion is not used

FRIC

friction for the dynamics of the screening charges
Type: real
Rules: optional
Default: 0.

OPTFRIC

If true, the optimum friction scheme used for the relaxation of the screening charges. See Sec. **??** for further information. Is only used in combination with option ADIABATIC.
Type: logical
Rules: optional, only used in combination with ADIABATIC=T
Default: .false.

RETARD

Only used in combination with OPTFRIC=T. The running average of the optimum friction scheme adjusts within RETARD time steps to one-half of the original deviation.
Type: real
Rules: optional, only used in combination with OPTFRIC=T
Default: 0.d0

6.2.23 !CONTROL!ANALYSE

Rules: optional
Description: Prepares special data for analysis

OPTIC

This option is currently disconnected. Writes data required for the optic package of M. Alouani. Attention! Many files are written starting with extension ".optics...". Does not work on parallel computers, and not for non-collinear spin density.

Type: logical
Rules: optional, currently disconnected
Default: false

6.2.24 !CONTROL!ANALYSE!TRA

Rules: optional
Description: select trajectories to be written.

R

positions an point charges.

Type: logical
Rules: optional
Default: .true.

FORCE

forces acting on the atoms.

Type: logical
Rules: optional
Default: .false.

E

total energy contributions.

Type: logical
Rules: optional
Default: .false.

BANDS

one-particle energies.(not yet implemented)
Type: logical
Rules: optional
Default: .false.

NSKIP

number of time slices to be skipped between two written ones.
Type: integer
Rules: optional
Default: 0

6.2.25 !CONTROL!ANALYSE!HYPERFINE

Rules: optional,multiple
Description: calculates hyperfine parameters for a selected atom. Note that hyperfine parameters are extremely sensitive quantities, and therefore plane wave convergence should be checked carefully. In some cases it is also necessary to use more partial waves in the augmentation than normally used for total energy calculations.
The magnetic hyperfine field B_i^N describes the magnetic field produced at the nuclear site by the electron spin distribution. The total hyperfine field depends on the direction of the electron spin, which is imposed by the external magnetic field. Hence the anisotropic hyperfine field is provided as a matrix that needs to be multiplied by a normalized vector pointing in the direction of the external magnetic field to give the contribution of the anisotropic part to the total hyperfine field. The total hyperfine field is obtained by adding the Fermi-contact field and the anisotropic hyperfine field. The hyperfine splitting ΔE is obtained by multiplication of the total hyperfine field B_i^N with $2\hbar\frac{m^N}{S_N}$, where m^N is the nuclear magnetic moment and S^N is the nuclear spin.

ATOM

atom name consistent with structure input file
Type: character
Rules: mandatory
Default: none

EFG

select electric field gradient calculation. Result will be documented in the protocol. The electric field gradient is related to the second derivatives of the potential at the nuclear site
Type: logical
Rules: optional
Default: .false.

ISOMERSHIFT

select isomer shift calculation. Result will be documented in the protocol. The isomer shift is related to the charge density at the nuclear site.
Type: logical
Rules: optional
Default: .false.

FERMICONTACT

calculates the Fermi contact field. It is the magnetic field acting on the nucleusResult will be documented in the protocol. The Fermi contact term is related to the spin density at the nuclear site.
Type: logical
Rules: optional
Default: .false.

ANISOTROPIC

selects anisotropic hyperfine parameter calculation. Result will be documented in the protocol. The anisotropic hyperfine parameter is related to the $\ell = 2$ component of the spin density near the nucleus

Type: logical
Rules: optional
¡ Default: .false.

6.2.26 !CONTROL!ANALYSE!CORELEVELS

Rules: optional, not tested
Description: calculates the positions of the core levels (absolute and relative to the isolated atom (atomic calculation)). Note that core relaxation effects, which contribute typically to 15 %, are not included. The absolute position of the core levels is only meaningfull for isolated molecules. For extended systems, the zero of the electrostatic potential is not the vacuum level, so that only relative shifts within a single calculation are meaningful. For Tests see Pasquarello et al., Phys. Rev. B 53, 10942 (1996).
Note does not work yet for parallel execution. Minor changes are required

DEFAULT

Selects if core-levels shall be calculated for atoms, which are not in the list of selected atoms.(see below).

Type: logical
Rules: optional
Default: .false.

ATOMS

List of atoms, for which the core levels shall be calculated or avoided. If default=F the core levels of specified are calculated. If default=T, the core levels of all atoms not specified are calculated.

Type: character,array of arbitrary length

Rules: optional
Default: none

6.2.27 !CONTROL!ANALYSE!WAVE

Rules: optional,multiple
Description: Writes a wave function. The file created can then be processed by the paw_wave tool to produce an input file for OPENDX[**?**]

TITLE

title of the image. Currently it is not used other than in the print-out.
Type: character
Rules: optional
Default: none

FILE

full file name of the file to be produced, which can be converted into a input file for OPENDX[**?**] using the paw_wave tool.
Type: character
Rules: mandatory
Default: none

DR

grid spacing for the output file. rounded to get an integer factor relative to the real-space grid used in the calculation.
Type: real
Rules: optional
Default: 0.4

B

band index
Type: integer

Rules: mandatory
Default: none

K

k-point index
Type: integer
Rules: mandatory
Default: 1

S

spin index
Type: integer
Rules: mandatory
Default: 1

IMAG

use imaginary part
Type: logical
Rules: optional
Default: F

6.2.28 !CONTROL!ANALYSE!DENSITY

Rules: optional,multiple
Description: Writes a density. The file created can then be processed by the paw_wave tool to produce an input file for OPENDX[**?**].

TITLE

title of the image. Currently it is not used other than in the print-out.
Type: character
Rules: optional
Default: none

FILE

full file name of the file to be produced, which can be converted into a input file for OPENDX[**?**] using the paw_wave tool.
Type: character
Rules: mandatory
Default: none

DR

grid spacing for the output file. rounded to get an integer factor relative to the real-space grid used in the calculation.
Type: real
Rules: optional
Default: 0.4

TYPE

can be 'TOTAL', 'SPIN', 'UP' or 'DOWN'. Determines the weights of the states in the density plots. 'TOTAL' takes the actual occupations and k-point or uniform weights (depending on 'OCC'). 'SPIN' is like 'TOTAL', but counts states for spin=2 negative. 'UP' and 'DOWN' give the spin up and down densities respectively.
Type: character
Rules: optional
Default: 'TOTAL'

OCC

use acutal occupations
Type: logical
Rules: optional
Default: T

DIAG

use eigenstates in the subspace of the dynamic wave functions.
Type: logical
Rules: optional
Default: T

CORE

include core density.
Type: logical
Rules: optional
Default: F

EMIN[EV]

lowest eigenenergy to be included.
Type: real
Rules: optional
Default: -1^{-10}

EMAX[EV]

highest eigenenergy to be included.
Type: real
Rules: optional
Default: 1^{-10}

BMIN

lowest band to be included.
Type: integer
Rules: optional
Default: 1

BMAX

highest band to be included.
Type: integer
Rules: optional

Default: 10000000

6.2.29 !CONTROL!ANALYSE!POTENTIAL

Rules: optional
Description: Writes the hartree potential. The file created can then be processed by the paw_wave tool to produce an input file for OPENDX[**?**].

TITLE

title of the image. Currently it is not used other than in the print-out.
Type: character
Rules: optional
Default: none

FILE

full file name of the file to be produced, which can be converted into a input file for OPENDX[**?**] using the paw_wave tool.
Type: character
Rules: mandatory
Default: none

DR

grid spacing for the output file. It is rounded to get an integer factor relative to the real-space grid used in the calculation.
Type: real
Rules: optional
Default: 0.4

6.3 The structure input file “STRC”

The structure input file defines the molecule or crystal to be studied.

6.3.1 Example for the structure input file “STRC”

This is an example of a structure input file for formaldehyde in a fcc supercell with lattice constant of 10 Å.

```
!STRUCTURE
  !GENERIC LUNIT=1.8897259926 !END
  !OCCUPATIONS NBAND=8 NSPIN=1 !END
  !LATTICE T= 0.00000  5.00000  5.00000
              5.00000  0.00000  5.00000
              5.00000  5.00000  0.00000 !END
  !SPECIES NAME= 'H_' ZV=1. M=1.9380
           FILE='/u/blo.2/PAW/Setups/BP86/001H/h1.out'  !END
  !SPECIES NAME= 'C_' ZV=4. M=18.9984
           FILE='/u/blo.2/PAW/Setups/BP86/006C/c1.out'  !END
  !SPECIES NAME= 'O_' ZV=6. M=15.99946
           FILE='/u/blo.2/PAW/Setups/BP86/008O/o1.out'  !END
  !ATOM    NAME= 'H_1'  R= -0.50000  0.00000 -0.86600 !END
  !ATOM    NAME= 'H_2'  R= -0.50000  0.00000  0.86600 !END
  !ATOM    NAME= 'C_3'  R=  0.00000  0.00000  0.00000 !END
  !ATOM    NAME= 'O_4'  R=  1.50000  0.00000  0.00000 !END
  !ISOLATE NF=3 RC=0.5 RCFAC=1.5 GMAX2=3.0 DECOUPLE=T !END
!END
!EOB
```

The following is not a working example. It is aimed at illustrating the syntax of the input file. It is more complex than a typical case because it incorporates (almost) all options at the same time.

```
!STRUCTURE
  !GENERIC LUNIT=1.0 !END
  !LATTICE T=0. 10. 10. 10. 0. 10. 10. 10. 0. !END
  !KPOINTS K= 0 0 0 !END
  !SPECIES  NAME='H_' ZV=1.  M=1.008 PSEKIN=0.0
           FILE=~/PAW/Setups/006H/h1.out !END
  !SPECIES NAME='C_' ZV=4. M=12.011
           FILE=~/PAW/Setups/006C/c1.out !END
  !ATOM NAME='alpha-hydrogen'
                  R= 1.0 1.0 1.0 SP='H_' !END
```

```
  !ATOM NAME='H_2' R=-1.0000 -1.0000  1.0000 !END
  !ATOM NAME='H_3' R=-1.0000  1.0000 -1.0000 !END
  !ATOM NAME='H_4' R= 1.0000 -1.0000 -1.0000 !END
  !ATOM NAME='C_1' R=0.0 0.0 0.0 M=5.0 !END
  !OCCUPATIONS
    NBAND=10 NSPIN=2 CHARGE[E]=0. SPIN[HBAR]=1.
    !STATE K=1 S=1 B=5 F=1.000 !END
    !STATE K=1 S=1 B=6 F=1.000 !END
  !END
  !ISOLATE NF=2 RC=1.0 RCFAC=1.5 GMAX2=3.
           DECOUPLE=T !END
  !GROUP NAME='BOND1'  ATOMS='C_1' 'H_1' !END
  !GROUP NAME='BOND2'  ATOMS='C_1' 'H_2' !END
  !CONSTRAINTS
    !RIGID GROUP='BOND1' !END
    !FREEZE GROUP='BOND2' !END
    !FREEZE ATOM='H_3'  !END
    !TRANSLATION GROUP='ALL' !END
    !TRANSLATION GROUP='BOND1' DIR= 1. 0. 0. !END
    !ORIENTATION  GROUP='BOND1' AXIS= 1. 0. 0. !END
    !LINEAR
      !LINE ATOM='H_1' VEC= 0.5 0.0  0.0 !END
      !LINE ATOM='H_2' VEC= 0.5 0.0 0.0 !END
    !END
  !END
!END
!EOB
```

6.3.2 Argument description for the structure input file "STRC"

6.3.3 !STRUCTURE

Rules: mandatory

Description: Defines the system to be studied. Specifies geometry, atom types, electronic occupations, atomic masses. etc...

6.3.4 !STRUCTURE!GENERIC

Rules: optional
Description: data that do not fit into other blocks.

LUNIT

specifies the length unit (in atomic units) for structural input in this file. All data are converted into atomic units (1 a.u.=0.529167$\times 10^{-10}$ m) by multiplication with LUNIT
Type: real
Rules: optional
Default: 1.0

6.3.5 !STRUCTURE!LATTICE

Rules: mandatory
Description: lattice translation vectors

T

Type: real(3,3)
Rules: mandatory
Lattice vectors in the order; (x,y,z of first vector; x,y,z of second vector...)
Default: none

6.3.6 !STRUCTURE!KPOINTS

Rules: optional
Description: specifies the k-point grid. Default is Γ-point sampling.

R

defines the density of k-points for the automatic generation of k-points. $\Delta k = 2\pi/R$.

Type: real
Rules: optional
Default: 12.

DIV

fractions of reciprocal lattice vectors defining the reciprocal sublattice for the k-points.If K is not specified, the k-points are equally spaced an set compatible with the inversion symmetry. Caution with non-orthogonal lattice vectors: $\vec{G}_1 = \frac{1}{V}\vec{T}_2 \times \vec{T}_3$, $\vec{G}_2 = \frac{1}{V}\vec{T}_3 \times \vec{T}_1$, $\vec{G}_3 = \frac{1}{V}\vec{T}_1 \times \vec{T}_2$, where $\vec{G}_i$ are the reciprocal space lattice vectors and $\vec{T}_i$ are the real space lattice vectors and $V = \vec{T}_1(\vec{T}_2 \times \vec{T}_3)$ is the volume of the real space unit cell.
Type: integer(3)
Rules: optional; mutually exclusive with R
Default: 2 2 2

SHIFT

shift is displaces the K-point grid such that the Γ-point is avoided. The three components of SHIFT specify the shift directions into the three lattice vectors. Each component may have the values 0 or 1. For zero, the grid is not shifted along the specified direction. For a value of 1, the grid is shifted by one-half of the grid spacing along the corresponding reciprocal lattice vector. While the density of the k-point grid is not affected by shifting the k-point grid, the k-point convergence is apparently faster. Shifting the grid avoids the high-symmetry points. Thus, one does not see the band edges, which usually lie on high symmetry points. The computational effort is not much affected by shifting the grid.
Type: integer(3)
Rules: optional
Default: 0 0 0

K