# INDEX

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 GENERAL</td>
<td>12</td>
</tr>
<tr>
<td>2 INPUT</td>
<td>27</td>
</tr>
<tr>
<td>2.1 Introduction</td>
<td>27</td>
</tr>
<tr>
<td>2.1 Running the program</td>
<td>27</td>
</tr>
<tr>
<td>2.1 Structure of the input</td>
<td>28</td>
</tr>
<tr>
<td>2.1 Keywords</td>
<td>29</td>
</tr>
<tr>
<td>2.1 Minimal input</td>
<td>31</td>
</tr>
<tr>
<td>2.2 Main Options</td>
<td>36</td>
</tr>
<tr>
<td>2.2 Parallel Execution</td>
<td>36</td>
</tr>
<tr>
<td>2.2 Run Types</td>
<td>36</td>
</tr>
<tr>
<td>2.2 Atomic Coordinates</td>
<td>38</td>
</tr>
<tr>
<td>2.2 Geometry Optimization</td>
<td>41</td>
</tr>
<tr>
<td>2.2 Transition State</td>
<td>44</td>
</tr>
<tr>
<td>2.2 Linear Transit</td>
<td>45</td>
</tr>
<tr>
<td>2.2 Intrinsic Reaction Coordinate</td>
<td>47</td>
</tr>
<tr>
<td>2.2 Optimization: Special Features</td>
<td>49</td>
</tr>
<tr>
<td>2.2 Frequencies</td>
<td>54</td>
</tr>
<tr>
<td>2.2 Fragments</td>
<td>55</td>
</tr>
<tr>
<td>2.2 QM/MM</td>
<td>57</td>
</tr>
<tr>
<td>2.2 Model Hamiltonian</td>
<td>57</td>
</tr>
<tr>
<td>2.2 Time-dependent DFT: Excitation Energies, (Hyper) Polarizabilities</td>
<td>67</td>
</tr>
<tr>
<td>2.2 ESR</td>
<td>75</td>
</tr>
<tr>
<td>2.2 Electronic Configuration</td>
<td>76</td>
</tr>
<tr>
<td>2.2 Multiplet States</td>
<td>80</td>
</tr>
<tr>
<td>2.2 Precision and Self-Consistency</td>
<td>82</td>
</tr>
<tr>
<td>2.2 Interpretation of Input</td>
<td>86</td>
</tr>
<tr>
<td>2.2 Restarts</td>
<td>90</td>
</tr>
<tr>
<td>2.2 Printed Output</td>
<td>97</td>
</tr>
<tr>
<td>2.2 ASCII Output Files with Atomic Coordinates</td>
<td>113</td>
</tr>
<tr>
<td>2.3 More Options</td>
<td>113</td>
</tr>
<tr>
<td>2.3 General</td>
<td>114</td>
</tr>
</tbody>
</table>

---

**PREFACE**

10

1 GENERAL

1.1 Introduction .................................................................................................................................................. 12
Characterization of ADF .................................................................................................................................... 12
Fragments .............................................................................................................................................................. 13
Files ....................................................................................................................................................................... 16
Parallel execution .................................................................................................................................................. 17

1-2 Technical remarks, Terminology ..................................................................................................................... 19
Basis functions and orbitals ................................................................................................................................. 19
Fit functions .......................................................................................................................................................... 22
Three-step build-up of the bonding ...................................................................................................................... 23
Transition State procedure .................................................................................................................................. 24

2 INPUT ............................................................................................................................................................ 27

2.1 Introduction .................................................................................................................................................. 27
Running the program ........................................................................................................................................... 27
Structure of the input ............................................................................................................................................ 28
Keywords ............................................................................................................................................................. 29
Minimal input ....................................................................................................................................................... 31

2.2 Main Options .................................................................................................................................................. 36
Parallel Execution ................................................................................................................................................. 36
Run Types ........................................................................................................................................................... 36
Atomic Coordinates .............................................................................................................................................. 38
Geometry Optimization ........................................................................................................................................ 41
Transition State .................................................................................................................................................... 44
Linear Transit ........................................................................................................................................................ 45
Intrinsic Reaction Coordinate .......................................................................................................................... 47
Optimization: Special Features .......................................................................................................................... 49
Frequencies ........................................................................................................................................................... 54
Fragments ............................................................................................................................................................ 55
QM/MM ............................................................................................................................................................... 57
Model Hamiltonian ............................................................................................................................................. 57
Time-dependent DFT: Excitation Energies, (Hyper) Polarizabilities .................................................................. 67
ESR ....................................................................................................................................................................... 75
Electronic Configuration ...................................................................................................................................... 76
Multiplet States .................................................................................................................................................... 80
Precision and Self-Consistency ............................................................................................................................ 82
Interpretation of Input .......................................................................................................................................... 86
Restarts ................................................................................................................................................................. 90
Printed Output ..................................................................................................................................................... 97
ASCII Output Files with Atomic Coordinates ................................................................................................ 113

2.3 More Options .................................................................................................................................................. 113
General ................................................................................................................................................................. 114

---

**GENERAL**

12

2.2 Main Options .................................................................................................................................................. 27

---

**INPUT**

27
### 3 RECOMMENDATIONS, PROBLEMS, QUESTIONS

#### 3.1 Recommendations

- Precision: 139
- Electronic Configuration: 140
- Geometry Optimization: 141
- Basis Sets for Organic Molecules: Single-zeta vs. Double-zeta: 142
- Frequencies: 143
- Relativistic methods: 143

#### 3.2 Trouble Shooting

- License file corrupt: 144
- Recover from Crash: 144
- Memory Management: 145
- SCF: 146
- Geometry Optimization: 147
- Frequencies: 149
- Input ignored: 149
- SFO Populations: 150
- Error Aborts: 150
- Warnings: 150

#### 3.3 Questions

- ...: 151

### 4 FILES

#### 4.1 Parallel Execution

- ...: 152

#### 4.2 Standard output

- Input Echo, Output Header: 153
- Main Job Characteristics: 153
- Build Info: Fragments and Function Sets: 154
- Technical Parameters: 154
- Computational Report: 154
- Results: 156
- Frequencies Results: 159
- Exit Procedure: 160
- Logfile: 160

#### 4.3 Log file

- ...: 161

#### 4.4 TAPE21

- Contents of TAPE21: 162
- Using Data from TAPE21: 163

#### 4.5 TAPE13

- Contents of TAPE13: 194
<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>RESULTS</td>
<td>196</td>
</tr>
<tr>
<td>5.1 Properties</td>
<td>196</td>
</tr>
<tr>
<td>Electronic Configuration, Orbital Energies</td>
<td>196</td>
</tr>
<tr>
<td>Populations and Atomic Charges</td>
<td>196</td>
</tr>
<tr>
<td>Energy</td>
<td>197</td>
</tr>
<tr>
<td>Thermodynamics</td>
<td>198</td>
</tr>
<tr>
<td>5.2 Plots: Density, Potential, Orbitals</td>
<td>200</td>
</tr>
<tr>
<td>APPENDICES</td>
<td>201</td>
</tr>
<tr>
<td>6.1 Database</td>
<td>201</td>
</tr>
<tr>
<td>Data File for Create</td>
<td>201</td>
</tr>
<tr>
<td>Example: Calcium</td>
<td>204</td>
</tr>
<tr>
<td>6.2 Elements of the Periodic Table</td>
<td>207</td>
</tr>
<tr>
<td>6.3 Symmetry</td>
<td>209</td>
</tr>
<tr>
<td>Schönfließ symbols and symmetry labels</td>
<td>209</td>
</tr>
<tr>
<td>Molecular orientation requirements</td>
<td>211</td>
</tr>
</tbody>
</table>
REFERENCES  212
PREFACE

ADF (Amsterdam Density Functional) is a Fortran program for calculations on atoms and molecules (in gas phase or solution). It can be used for the study of such diverse fields as molecular spectroscopy, organic and inorganic chemistry, crystallography and pharmacochemistry. A separate program in the ADF package (BAND) is available for the study of periodic systems: crystals, surfaces, and polymers.

The underlying theory is the Kohn-Sham approach to the Density-Functional Theory (DFT). This implies a one-electron picture of the many-electron systems but yields in principle the exact electron density (and related properties) and the total energy.

If ADF is a new program for you we recommend that you carefully read Chapter 1, section 1.2 “Technical remarks, Terminology”, which presents a discussion of a few ADF-typical aspects and terminology. This will help you to understand and appreciate the output of an ADF calculation.

ADF has been developed since the early 1970s (at that time called HFS, later AMOL), mainly by the two theoretical chemistry groups of, respectively, the Vrije Universiteit in Amsterdam (http://tc.chem.vu.nl) and the University of Calgary, Canada (http://www.cobalt.chem.ucalgary.ca/group/master.html). Other researchers have also contributed. A recent (new) development center is the Theoretical Chemistry group at the Groningen university (http://theochem.rug.nl). As a major research tool of these academic development groups ADF is in continuous development and retains a firm basis in the academic world.

Maintenance and distribution of the commercial (export) version of the program is by Scientific Computing & Modelling NV (SCM) (http://www.scm.com), a company based in Amsterdam, formally split off from the theoretical chemistry group in Amsterdam but practically still very much a part of it.

Documentation such as User manuals, Installation instructions, Examples, Theoretical documents can be found at the SCM web site.

Publications based on research with ADF should include appropriate references to the program. We recommend that references are made both to the program itself and to publications related to its development and structure. See the “Publication Citations” document, available at the SCM web site.

ADF 2000.02

ADF2000.02 is an upgrade of ADF2000.01. The “2000” indication does not imply that it is a substantially different version from the “1999” releases. Since 1999 we have adopted the convention to name each release version simply after the year of release, and to number different releases within one year by increasing the suffix. No other meaning is implied by release numbers.

In comparison to ADF1999.02, the 2000.02 release offers the following new functionality:

- QM/MM: the mixed Quantum Mechanics (QM) / Molecular Mechanics (MM) approach to treat large systems in which only a smaller part requires accurate (QM) treatment, while the remainder may be handled with approximate (MM) methods to represent environment effects, steric and/or electrostatic effects, and so on. A detailed discussion is given in the separate QMMM manual. The relevant input keys are mentioned in this manual.
There are a few limitations regarding the use of the QMMM functionality. The QMMM feature cannot be used with a Frequencies run. In a Linear Transit the LT parameters must not involve any MM atoms.

- **ZORA-NMR**: the NMR chemical shifts functionality in ADF1999.x is now available also in combination with the advanced ZORA relativistic approach. In many cases this yields a marked improvement over the traditional relativistic approximation in the Pauli formalism.

Furthermore, several deficiencies, bugs and inaccuracies in ADF1999 have been improved upon, more sample runs are available and the documentation has been increased for further clarification of some issues. A more extended list of “what is new or different” can be found in the Updates document.
1 GENERAL

1.1 INTRODUCTION

The installation of the Amsterdam Density Functional program package (ADF) on your computer is explained in the Installation manual. This User's Guide describes how to use the program, how input is structured, what files are produced, and so on. Some special applications of ADF are described in the Examples document.

Where references are made to the operating system (OS) and to the file system on your computer the terminology of UNIX type OSs is used and a hierarchical structure of directories is assumed.

The ADF package is in continuous development to extend its functionality and applicability, to increase its efficiency and user-friendliness, and of course to correct errors. We appreciate comments and suggestions for improvement of the software and the documentation.

Characterization of ADF

Functionality

- Single Point calculation
- Geometry Optimization
- Transition States
- Frequencies and thermodynamic properties
- Tracing a Reaction Path
- Computation of any electronic configuration
- Excitation energies and (hyper) polarizabilities, using Time-Dependent Density Functional Theory (TDDFT)
- NMR chemical shifts
- Excitation Energies
- (Hyper) Polarizabilities
- Van der Waals Coefficients
- Treatment of large systems en environment by the QM/MM (Quantum Mechanics / Molecular Mechanics) hybrid approach.

Applicability

All elements of the periodic table can be used. The database contains for each of the elements basis sets of different sizes, ranging from minimal to high quality. Special basis sets are provided for relativistic calculations within the ZORA approach and for response calculations that require additional diffuse basis functions.
**Model Hamiltonian**

- A choice of Density Functionals, both for the Local Density Approximation (LDA) and for the Generalized Gradient Approximation (GGA)
- Spin: restricted or unrestricted
- Relativistic effects: scalar approximation and spin-orbit (double-group symmetry), using the (now recommended) ZORA or the (previously used) Pauli formalism
- Environment: Solvent Effects, Homogeneous Electric Field, Point Charges (Madelung Fields), QM/MM method

**Analysis**

- Decomposition of the bond energy in ‘chemical’ components (steric interaction, Pauli repulsion, orbital interactions...)
- Representation of data (Molecular Orbital coefficients, Mulliken Populations) in terms of the constituent chemical fragments in the molecule, along with the conventional representation in elementary basis functions
- Atomic charge determination by Hirshfeld analysis and by Voronoi analysis, along with the classical Mulliken populations

**Technical**

- The implementation is based upon a highly optimized numerical integration scheme for the evaluation of matrix elements of the Fock operator, property integrals involving the charge density, etc. The code has been vectorized and parallelized.
- Basis functions are Slater-Type Orbitals (STOs). A database is available with several basis sets for each atom in the periodic table of elements.
- The Coulomb potential is evaluated via an accurate fitting of the charge density with so-called fit functions, which are Slater-type exponential functions centered on the atoms. The fit functions are included in the database files.
- A frozen core facility is provided for an efficient treatment of the inner atomic shells.
- Extensive use is made of point group symmetry. Most of the commonly encountered symmetry groups are available.

**Fragments**

ADF has a fragment oriented approach: the poly-atomic system to be computed is conceptually built up from fragments, the molecular one-electron orbitals are calculated as linear combinations of fragment orbitals, and final analyses of e.g. the bonding energy are in terms of fragment properties. The fragments may be single atoms or larger moieties.

When you compute a system in terms of its constituent fragments, these fragments must have been computed before and their properties must be passed on to the current calculation. This is done by attaching fragment files,
which contain the necessary information. A fragment file is simply the standard result file of an ADF calculation on that fragment.

**Basic atoms**

Obviously there must be a set of fundamental fragments that are not defined in terms of smaller fragments. Therefore ADF has two modes of execution: the normal mode, using fragments, and the CREATE mode, in which a fundamental fragment is generated. Such a fundamental fragment must be a single atom, spherically symmetric and spin-restricted (i.e. spin-α and spin-β orbitals are spatially identical, they are equally occupied, and fractional occupations are applied, if necessary, to distribute the electrons equally over symmetry-degenerate states). Such a fundamental fragment is denoted a *basic atom*. The basic atoms are the smallest building blocks from which any ‘real’ calculations are started.

One should realize that the basic atoms are artificial objects that are convenient in the computational approach but that do not necessarily represent real atoms very well (in fact, usually not at all). The bonding energy of a molecule with respect to basic atoms, for instance, should be corrected for this discrepancy in order to get a decent comparison against experimental data. See ref. [1] for a discussion and for examples of applicable values.

A basic atom is computed in the conventional way. The one-electron orbitals are determined as linear combinations of basis functions; the frozen core approximation may be applied for the inner atomic states; a particular type of density functional can be chosen, et cetera. You may have, for instance, different basic Copper atoms by using different basis sets, by choosing different levels of frozen core approximations, or by applying different density functionals.

**Database**

The ADF package is equipped with a database to help you generate basic atoms. Each data file in the database contains a standard basis set (and related information) for the creation of one basic atom. The data files are relatively small ASCII files. You can easily inspect them. In Appendix 1 a definition is given of such a file. This enables you to create variations and construct your own adapted basis sets.

The basis functions used in ADF are commonly known as Slater Type Orbitals (STOs). A basis set can roughly be characterized by its size (single-, double-, triple-zeta; with or without polarization) and by the level of frozen core approximation. The database directory `atomicdata` has five subdirectories (I, II, III, IV, and V) for the creation of basic atoms. Each of these subdirectories contains a set of data files. The numbering I through V corresponds to an increasing quality (and size) of the basis set.

In addition, the database contains directories with special basis sets:

- **ZORA** contains basis sets that should be used (exclusively) for relativistic calculations with the ZORA approach. Using ‘normal’ basis sets in a ZORA calculation may give highly inaccurate results, in particular for heavy elements. The ZORA directory contains subdirectories I, II, III, IV and V. These correspond to the ‘normal’ basis set directories I through V.
- Vdiff contains basis sets that include very diffuse functions. These are to be used for Response calculations. They are absolutely necessary to get good results for excitation energies corresponding to high lying orbitals.
- OLD contains basis sets that were contained in the 2.3 release, but that we feel should not be used anymore unless with great care, primarily because the involved frozen cores are too large to justify the frozen core approximation. In some cases we found uncomfortably large errors in equilibrium geometries resulting from such too-large cores.

Furthermore, you will find in the database:
- AE: this contains basis sets for all-electron calculations. However, these files cannot be used as such, because they don't contain any fit sets. Using basis sets without fit sets is pointless and is in fact not possible at all. (The usage and relevance of fit functions is explained later). Therefore, they serve as starting point for the development of (new) basis sets. For some of the all-electron sets appropriate fit sets have already been generated. The corresponding data base files can be found in the appropriate subdirectories I, II, III et cetera.
- Cerius. This directory contains data files that are used in the Cerius2-ADF graphical user interface.
- Dirac contains the input files for the DIRAC auxiliary program (see the UTILITIES document)
- Band contains input files for the BAND program (see the BAND User’s Guide)
- ForceFields contains force field files to be used in the QM/MM functionality. Their structure and contents are described in the QM/MM manual.

The files in I/ (and ZORA/I/) have minimal basis sets: single-zeta without polarization. The exponents of the functions correspond to the standard STO-3g basis sets used in programs that employ Gaussian type basis functions. The frozen core approximation is applied, however, for the inner atomic shells. Type-I database files are provided only for the lighter elements, up to Kr.

The files in II can be characterized as double-zeta basis sets without polarization functions. A triple-zeta set is used for the 3d shells of the first row transition metals, the 4f shells of the Lanthanides, and the 5f shells of the Actinides. In all these cases a double-zeta set provides a rather poor expansion basis for the true (numerically computed) atomic orbital.

The basis sets in III are derived from II, extended with a polarization function. This type of basis sets is thus far provided only for the elements up to Ar, and for the 4p series Ga through Kr, because of the lack of experience with polarization functions for other atoms.

IV contains triple-zeta basis sets. A polarization function is added for H through Ar and for Ga through Kr (from III).

V finally gives extended basis sets: triple-zeta with two polarization functions, for H through Ar.

Multiple occurrences of one chemical element in the same basis set subdirectory correspond to different levels of the frozen core approximation. Manganese for instance may have a basis set for an atom with a frozen 2p shell and another one with a frozen 3p shell. The file names are self-explanatory: Mn.2p stands for a data file for Manganese with frozen core shells up to the 2p level. An all-electron basis set would correspond to a file that has no frozen-core suffix in its name.

Another type of multiple occurrence of one element in one database directory may be found when basis sets have been developed for different electronic configurations: the Slater-type basis sets are fitted then to numerical orbitals from runs with different occupation numbers. Currently this applies only for Ni (in database directories II and IV), where basis sets are supplied for the d^8s^2 and the d^9s^1 configurations respectively. Since in earlier
releases only the d\textsuperscript{8}s\textsuperscript{2} variety was available, the names of the database files are Ni_{2p} (for d\textsuperscript{8}s\textsuperscript{2}) and Ni_{d9.2p}, and likewise Ni_{3p} and Ni_{d9.3p}.

As mentioned above, some all-electron basis sets are present in the basis set directories I through V, but not for all elements of the periodic table. The heavier elements, from Rb on, are missing. For these elements, basis sets are provided in a separate directory A\textsc{e}, which contains basis sets of single-, double- and triple-zeta quality indicated respectively by suffixes ‘sz’, ‘dz’, and ‘tz’. The files in A\textsc{e}/ are not complete database files, because they don’t contain fit sets (the usage and relevance of fit functions is explained later). The development of fit sets and their testing is not a triviality. It is absolutely a bad idea to take a fit set from another database file, corresponding to some frozen core level, and use that in an all-electron basis set: this will give significant errors and make results worthless. We intend to construct fits sets for the heavier elements in a later release.

**Fragment libraries**

If you create from all the data files the corresponding basic atoms and store the result files in a special directory, you have an adequate and flexible library of fragments to start any normal calculations. Of course, you may augument the fragment library by compiling additional data files with other basis sets and generate other basic atoms, according to taste or necessity. More importantly: you can store the result files of larger molecular fragments in libraries. These can then serve in turn as building blocks (fragments) for subsequent calculations. In general one should be careful with using fragment files from an earlier release of the ADF package for calculations with later versions. Often it is simply not possible at all, when the binary result files are not compatible. But even when they are, you should be careful since there may be numerical (and other) differences that affect the analysis of results. We recommend that you do not mix/use result files from earlier releases for the newer versions at all.

**Files**

Any files produced by the program are generated in the local (working) directory where the calculation runs. If you want to keep them, make sure to move them after the calculation has finished to wherever you want to store them. Files attached to the job, such as fragment files, are by default also assumed to exist in the local directory. You must take care to move or copy required files to that directory before starting the calculation, or to provide via input adequate information to the program where to find the files. In many cases you can specify a complete path to the file.

Most files that are generated by the program, in particular the standard result file that can be used as a fragment file in other calculations, are binary files. A binary file should usually not be moved from one machine to another, i.e. it may not be readable by another machine than the one that generated the file, unless the two machines are of the same type. The ADF package provides utilities to convert the ADF binary result files from
binary to ASCII, and vice versa, so that you don't have to regenerate your fragment libraries when going to another machine. See the UTILITIES document.

Two of the files that are produced by ADF deserve special attention. The first is the general result file TAPE21. It is a binary file that contains a lot of information about the calculation, such as the one-electron orbitals expressed in the basis functions. It can be used as a fragment file for subsequent calculations (although only TAPE21 files from spin-restricted calculations can be used as fragment files). Like all files produced by the program, it is generated in the directory where the job runs. Having done a calculation, you will usually store TAPE21 somewhere under a suitable name so that you can later reuse it, as a fragment file, for a restart, to feed it to an analysis program, and so on.

The second is an ASCII log file, called logfile. It accumulates messages from ADF into a (brief) summary of the run. You can inspect it during the calculation to check how far the calculation has proceeded, whether there are important warnings and so on. At the end of the run this file is copied to the tail of the normal standard output file.

**Standard output**

ADF is a program that lends itself particularly well for chemical analysis. This is a direct result of the fragment-based approach, where properties of the molecule are related to the properties of the constituent fragments, which is precisely how the chemist thinks. Molecular Orbitals are (optionally) analyzed extensively as how they are composed from occupied and virtual fragment orbitals. This inherently implies a large amount of output. Even computations on small molecules may produce startlingly many pages of output. This is not necessarily so because you can regulate the production of output in detail. Obviously, some kind of default production of output had to be implemented. The field of ADF users is so wide and diverse that it is hard to satisfy everybody as regards this default level of output. Depending on your purposes the automatic settings, which determine how much output is generated without instructions to the contrary, may yield boringly many numbers that you just skip through in search for the one value you're interested in, or it may be widely insufficient. Therefore, take notice of the possibilities to regulate output.

Above all, however get familiar with the analysis tools that ADF provides to see in what ways these may help to interpret your results. In a later chapter a global description of output is given as it is normally produced. The chapter below gives an introduction in some of the essential features of ADF, which may be sufficiently different from what you are used to in other Quantum Chemistry codes to deserve your attention.

**Parallel execution**

If a parallel version of ADF has been installed you should be aware of a few special aspects of running ADF in parallel. Partially this depends on the platform and on the installation settings.

First of all, you may specify (by command-line options in the run-script and/or by defining suitable environment variables) explicitly how many parallel processes are to be used. Secondly, you should realize that most of the files that you would have in a single-node run are in a parallel run distributed over the parallel processes. Some parts of the file may be identical across the processes while other parts are not and would only after a
recombination yield the data of the corresponding single-node file. The normal result files, (standard output, the logfile and the binary result file TAPE21) are complete at the master process.

**File names**

A special problem arises when the parallel processes use the same directory to create their files in. Whether or not this is the case depends on the platform. Consult the Installation Manual or ask the person who has installed the program, or someone else who is familiar with parallelization on your machine.

*If* the parallel processes use the same directory a conflict of names would arise if the different processes all tried to create for instance a file called TAPE21. Therefore, the file names are modified in this situation and all normal file names are appended by \( n \) where \( n \) is an integer between 0 and \( N(N+1) \) being the total number of parallel processes. The \( _1 \) files correspond to kid #1, etc. The program run scripts automatically take care of this aspect by renaming or copying files as required. The program run scripts are part of the ADF package and should be used, as described later in this manual (see also the Examples document), in particular for parallel runs.

Standard output is a special case here: the parent writes its normal ('print') output to standard output while the *kids* each write to a file KidOutput. When file names are modified as discussed above, the kid output files become KidOutput\( _n \). There is also a KidOutput\( _0 \) (from the parent) but this file is empty (apart from a dummy record): the parent writes its output to standard output.
1.2 TECHNICAL REMARKS, TERMINOLOGY

A few words about ADF as regards its technical setup and the names and abbreviations used in this manual. References to these will be made in the discussion of output and print switches.

Basis functions and orbitals

Let us make a clear distinction between (basis) functions and orbitals, even where these phrases are sometimes mixed up in the traditional terminology. Orbitals are always specific combinations of the basis functions. Orbitals are related to the computed eigenfunctions of some Fock operator or Hamiltonian occurring in the run or in a related preceding calculation. Functions are merely the elementary mathematical entities in which the orbitals are expressed. A Slater Type Orbital (STO), for instance is a function, not an orbital.

The physical meaning of one-electron orbitals in DFT has often been questioned. We believe that they are useful quantities for interpretation, just like the HF orbitals. For a recent discussion see [2].

Cartesian function sets, spurious components

ADF employs Slater-type exponential basis functions centered on the atoms. Such a function consists of an exponential part $\exp(-\alpha r)$ and a polynomial pre-factor $r^{k_x} x^{k_y} y^{k_z} z^{k_z}$. A function set is characterized by its radial behavior (the exponential part and the power of $r$, $kr$) and by its angular momentum quantum number $l$.

The functions in such a set consist of all possible combinations $x^{k_x} y^{k_y} z^{k_z}$, such that $k_x+k_y+k_z=l$. These are denoted the Cartesian spherical harmonics.

The Cartesian function sets are very suitable for computational manipulations, but they have a drawback. By inspection it is easily verified that a $d$-set consists of 6 Cartesian functions, while there can of course be only 5 true $d$-type functions among them: one (linear combination) of them is in fact an $s$-type function ($x^2+y^2+z^2$).

Similarly, there are 10 $f$-type Cartesian functions, 3 of which are in fact $p$-functions. And so on. In ADF all such lower-$l$ (combinations of) functions are projected out of the basis and not employed. As a consequence the basis set size in the sense of the number of degrees of freedom and hence the number of possible eigenfunctions of the Fock operator is smaller than the number of expansion coefficients that refer to the primitive (Cartesian) basis functions.

The abbreviation BAS is used for references to the elementary Cartesian basis functions.

Frozen core: Core Orbitals and Core Functions

To speed up the computation the innermost atomic shells are kept frozen. The frozen Core Orbitals (CO), which are solutions of a large-basis all-electron calculation on the isolated atom, are expressed in an auxiliary set of (Slater-type) basis functions COR-BAS, distinct from the valence set. The core basis set and the expansion coefficients that give the COs expressed in them are stored in the database data files.

Orthogonality of the valence Molecular Orbitals (MO) to the COs is achieved with the help of so-called Core Functions (CF). These functions are included in the valence set but they are not additional degrees of freedom.

Each of the normal valence functions is combined with a linear combination of all CFs in the molecule in such a
way that the transformed function (CBAS) is orthogonal to all frozen COs in the molecule. There are exactly as many CFs as COs so the orthogonality condition for all valence basis functions amounts to the solution of a linear system where the number of conditions equals the number of parameters.

This aspect once more increases the discrepancy between the number of expansion coefficients of an MO and the number of MOs: the expansion coefficients in the most elementary BAS representation run over all BAS functions, including the CFs among them. At some places there may, alternatively, be expansions in the core-orthogonalized BAS functions, CBAS, where the CFs do not count anymore: they are included implicitly in the CBAS functions.

**Symmetry**

The Overlap and Fock matrices become block-diagonal by using symmetry-adapted combination of the (C)BAS functions, such that each such function transforms under the symmetry operators as one of the subspecies of the irreducible representations (irrep) of the symmetry group. Symmetry adapted functions are denoted (C)SBAS.

For a given irrep and subspecies not all elementary basis functions can participate in the symmetry adapted combinations. For instance, for an atom in a reflection plane a basis function that is antisymmetric with respect to the reflection cannot be part of any symmetric combination of functions. In particular for higher symmetries the number of BAS functions that are relevant for a subspecies may be considerably smaller than the total number of BAS functions. This is used to cut down expansion lengths, both as used internally in the computation and construction of the Fock matrix, and in printed output. The printed expansion coefficients (in the BAS representation) refer only to the participating BAS functions. A defining list of them is printed at an early stage of the run for each of the subspecies.

**Orthonormal basis**

It is often computationally convenient to use an orthonormal basis. This is constructed from the CSBAS basis by a Löwdin orthogonalization procedure. The resulting symmetry-adapted orthonormal basis is denoted LOW. The MOs are computed by diagonalization of the Fock matrix in the LOW representation. The resulting eigenvectors are easily transformed back to any other representation whenever suitable, such as for instance to the primitive cartesian BAS representation (including the CFs).

**Fragments**

Except in Create mode, where a basic atom is constructed, the system is built up from fragments and the corresponding fragment files are attached to the run. The program reads from the files the fragment MOs and these are used as (compound) basis functions for the molecular calculation. The fragment MOs are called Fragment Orbitals (FO).

FOs belong of course to one of the symmetry representations of the fragment, but not necessarily to a symmetry representation of the new molecule. The FOs are therefore combined into symmetry-adapted combinations, SFOs, to serve as a symmetry-adapted basis in the molecule. These combinations may involve one or more FOs from the same fragment and/or from different fragments. In the latter case the fragments must be symmetry related by one.
of the operators of the molecule. Symmetry related fragments must of course be identical, apart from their spatial location: they must be of the same fragment type.

FOs are naturally orthogonal to the Core Orbitals of their own fragment, but not necessarily to COs of other fragments. By a suitable combination of the SFOs with all CFs in the molecule we obtain the core-orthogonalized symmetry-adapted CSFOs.

The CSFOs can be transformed to an orthonormal basis by a Löwdin transformation. The resulting basis is called LOW, as above.

**Summary of functions and orbitals**

In Create mode the (conceptual) approach is:

\[
\text{BAS} \Rightarrow \text{(core-orthogonalization)} \Rightarrow \text{CBAS} \Rightarrow \text{(symmetry)} \Rightarrow \text{CSBAS} \Rightarrow \text{(orthonormality)} \Rightarrow \text{LOW} \Rightarrow \text{(Fock diagonalization)} \Rightarrow \text{MO}
\]

In Fragment mode:

\[
\text{FO (=MO from fragment file)} \Rightarrow \text{(symmetry)} \Rightarrow \text{SFO} \Rightarrow \text{(core-orth.)} \Rightarrow \text{CSFO} \Rightarrow \text{(orthonormality)} \Rightarrow \text{LOW} \Rightarrow \text{(Fock diagonalization)} \Rightarrow \text{MO}
\]

**Acronyms:***

**BAS**

elementary cartesian basis functions, consisting of a radial part (exponential factor and power of r) and an angular part (cartesian spherical harmonic). The complete BAS set contains spurious lower-l combinations; these combinations are projected out and not used in the calculation. The BAS set contains also Core Functions.

**SBAS**
symmetry-adapted combination of BAS functions.

**CF**

Core Function, part of the BAS set. The CFs do not represent degrees of freedom in the basis set but serve only to ensure orthogonalization of the valence space to all frozen Core Orbitals.

**CBAS**
core-orthogonalized elementary basis functions: the true valence (not-CF) BAS functions transformed by adding a suitable combination of the CFs. The total number of CBAS + the total number of of CFs equals the total number of BAS.

**CSBAS**
symmetry-adapted combination of CBAS functions.

**CO**
frozen Core Orbitals, expressed as linear combinations of an auxiliary CORBAS basis set. The CORBAS set plays no role in the further discussion. The CORBAS functions are not the CFs.

The number of COs equals the number of CFs.
Lowdin orthonormalized symmetry-adapted core-orthogonalized basis. In Create mode they are derived directly from the BAS functions, in Fragment mode from the Fragment Orbitals, which are themselves of course expressible in the BAS set.

**FO**

Fragment Orbital: the MO of a fragment calculation, now used as a basis function in the molecule of which the fragment is part.

**SFO**

Symmetry adapted combination of FOs.

**CSFO**

Core-orthogonalized SFO.

### Fit functions

Using Slater-type basis functions yields awkward multi-center integrals in the evaluation of the Coulomb potential. This is remedied by employing an auxiliary set of fit functions. Like the basis functions, the fit functions are Slater-type exponential functions centered on the atoms. The true density, a sum of products of basis functions, is then replaced (approximated) by a linear combination (not products!) of the fit functions. The combination coefficients are called the fit coefficients.

\[ \rho(r) \approx \text{Error!} \]  

(1.2.1)

The Poisson equation for the fit functions is easily solved, yielding the (approximate) Coulomb potential as an expansion in fit potential functions \( f_{c,i}(r) \).

\[ f_{c,i}(r) = \text{Error!} \]  

(1.2.2)

\[ V_{\text{Coulomb}}(\rho(r)) \approx \text{Error!} \]  

(1.2.3)

In the SCF procedure the fit coefficients are computed by a least-squares minimization of

\[ \text{Error!} \, dr \]  

(1.2.4)

with the constraint that \( \rho_{\text{fit}} \) contain the correct number of electrons, \( \rho_{\text{exact}} \) is defined as the sum of occupied orbitals (squared and multiplied by the appropriate occupation number). The accuracy of the fit approximation is important and the fit set plays a role similar to the basis set: too few functions (or badly chosen function characteristics) yield inferior results and there is also such a thing as the fit set limit. The fit functions on an atom are consequently an integral part of the definition of the basic atom and they are included in the Create data files. Fortunately, the size of the fit set does not determine the computational effort in such a drastic way as the size of the basis set does. We have chosen therefore to use always fair (though not extreme) fit sets, with the purpose that the effect of fit-incompleteness should in all cases be small enough to be ignored compared with basis set effects,
numerical integration errors and Density Functional deficiencies. This does of course depend somewhat on the computed molecule and the studied properties, so a general guarantee cannot be given and, as with basis set effects, one should always have an open eye for possible problems and check the pertaining information in the output file.

One of the most important properties of a molecule is its energy, or its bonding energy with respect to the constituent fragments. The fit incompleteness introduces two types of errors. The first is that, since the Coulomb potential is only approximated, the SCF solution itself, i.e., the set of self-consistent Molecular Orbitals and their energy eigenvalues may be slightly wrong, yielding an error in the charge density and hence in the energy. Since the energy is to first order stable with respect to changes in the MO coefficients this error in the energy can be assumed very small. The second type of error derives from the computation of the energy from the (self-consistent) charge density, via the Coulomb potential. Let

$$\rho = \rho_{\text{exact}}(r) = \rho_{\text{fit}}(r) + \delta(r)$$

(1.2.5)

and

$$V_{\text{fit}}(r) = \text{Error!}$$

(1.2.6)

For the Coulomb energy of the charge density we have

$$2E_{\text{Coul}} = \text{Error!} = \text{Error!} + \text{Error!} = \text{Error!} + \text{Error!}$$

(1.2.7)

from which we see that the fit error is corrected to first order (by adding the fit deficiency $\delta(r)$ to the exact charge density when integrating against the fit potential) and that only a second order term remains that cannot be evaluated, the last term in the right-hand-side of (1.2.7).

A fair impression of the fit quality and the importance of the second order error term is obtained by checking $a)$ the size of the first order correction term $[V_{\text{fit}}(r) \delta(r)$ and $b)$ the norm of the deficiency function, $|\delta^2(r)\,dr$.

Both are printed in standard output, at the end of the output of the SCF procedure computational report. They are usually very small, which gives some confidence that the second order fit error can be ignored.

### Three-step build-up of the bonding

The approach of ADF is based on fragments. This applies not only in the analysis at the end of the computation but also in the set-up of the program. The computation of the molecule from its constituent fragments takes place in three steps, and these are reflected in the analysis of bond energy components.

First, the (free, unrelaxed) fragments are placed at their positions in the molecule. This implies an electrostatic interaction: for each fragment the Coulomb interaction of its undisturbed charge density with the fields of the other fragments.
Next, the Pauli exclusion principle is applied. Even without considering self-consistency the one-electron orbitals of the combined fragments cannot represent a correct one-determinant wavefunction because the orbitals of different fragments are not orthogonal to one another. The program performs an orthonormalization of the occupied Fragment Orbitals to obtain an antisymmetrized product. This implies a change in the total molecular charge density from the sum-of-fragments to what is called the sum-of-orthogonalized-fragments. The corresponding (repulsive) energy term is evaluated separately and is called Exchange repulsion, or alternatively Pauli repulsion. The phrase orthogonal(ized) fragments, if you find it elsewhere in this manual or in the source code of ADF, refers to this aspect. The sum of Pauli repulsion and electrostatic interaction is called the steric interaction.

The third phase is the relaxation to self-consistency, with of course the ensuing contributions to the bond energy.

**Transition State procedure**

This phrase stands for an analysis method described in ref. [3] and has no relation to transition states in chemical reactions. An extensive discussion of bond energy analysis by ADF is given in [4, 5].

The energy associated with a change in charge density, say the relaxation to self-consistency from the sum-of-orthogonal-fragments, can be computed by subtracting final and initial energies, each obtained from the corresponding charge density. For purposes of analysis the change in energy $\Delta E$ can be reformulated as

$$
\Delta E = \text{Error!}
$$

(1.2.8)

F(\rho) is the Fock operator belonging to the charge density \( \rho \).

By writing the density difference $\rho_{\text{final}} - \rho_{\text{initial}}$ as a summation over contributions from the different irreducible representations $\Gamma$ of the molecular symmetry group, an expression is obtained that lends itself for a decomposition of the bond energy into terms from the different symmetry representations:

$$
\Delta E = \text{Error!}
$$

(1.2.9)

The integral of the Fock operator over the charge density is now approximated by a weighted summation (in fact, a Simpson integration):

$$
\text{Error!} \approx \text{Error!}F(\rho\text{Error!}) + \text{Error!}F(\rho\text{Error!}) + \text{Error!}F(\rho\text{Error!})
$$

(1.2.10)

$$
\rho_{\text{average}} = \text{Error!}
$$

(1.2.11)

The term with the Fock operator due to the average charge density has given rise to the phrase transition state. To avoid confusion we will often refer to it as to the transition field.
The approximate integral (1.2.10) involves two errors. The first, rather obvious, is the approximation of the exact integral in (1.2.9) by the weighted sum in (1.2.10). Except in pathological cases this approximation is highly accurate.

The second error comes from the fact that the Coulomb and XC potentials in the Fock operator are computed from the fit density. This is only an approximation to the true density, while in the original bond-energy expression (energy due to the final density minus energy due to the initial density) no potentials occur and the exact charge density can be used. As mentioned before, these fit-related errors are usually small.

All such errors in the total bonding energy are easily corrected by comparing the summation over the Γ's with the correct value for the total bonding interaction term. The difference is simply added to the total bond energy, so no true error remains. We only have a (correction) term that can't be split in contributions from the distinct symmetry representations. In the printed bond energy analysis such small corrections are ‘distributed’ over the other terms by scaling the other terms such that their sum is the correct total value.
2 INPUT

2.1 INTRODUCTION

Running the program

When ADF has been installed you can run it by supplying appropriate input and starting the binary executable ("adf.exe" in $ADFBIN). For a parallel run a few things have to be arranged first, such as the ‘export’ of some environment variables to the parallel processes. All this is arranged by a run script “adf”, located in $ADFBIN. You can use this run script also for the serial version. For other programs in the package, there are similar run scripts (“band”, “dirac”, and so on). We recommend to always apply the script, rather than the binary executable directly. Running the program using the run script involves the following steps:

- Construct an ASCII input file, say in.
- Run the program by typing (under UNIX):
  
  $ADFBIN/adf {-n nproc} <in >out

  The part between curly brackets is optional, so the shortest application has the format
  
  $ADFBIN/adf <in >out

- Move / copy relevant result files (in particular TAPE21) to the directory where you want to save them, and give them appropriate names.
- Inspect the standard output file out to verify that all has gone well.

During the run you may inspect the logfile, to see how far the program has proceeded, or whether you should interrupt the calculation.

In the above scheme, adf is the name of the run script that invokes the adf.exe program executable. During the installation the script has been put in the same directory where the program executables are generated: $ADFBIN. You may have moved it to another place, or renamed it. We recommend that you adjust your $PATH variable so that you can omit $ADFBIN from the execution command.

To run another program from the ADF suite, just use the appropriate program run script.

The input for the program is read from standard input, and output is written to standard output, redirected in the example above to in and out, respectively.

The part between square brackets is optional and is only meaningful for a parallel program version. The \(-n\) flag specifies the number of parallel processes (nproc) to use. If omitted the default applies, which is the value of the environment variable $NSCM, if such variable exist, otherwise it is defined by installation parameters (in the $ADDFHOME/settings file, see the Installation Manual).

Irrespective of the \(-n\) value, the $NSCM variable value and any installation parameters, two items define an absolute upper bound for the number of parallel processes that can be used. The first is given (PVM) by the so-called virtual parallel machine, i.e. by the number of hosts that constitute the virtual parallel machine. See the
PVM manuals. The second item is the maximum number of parallel processes that you allow to run on each particular host that is part of the virtual parallel machine. This information must be stored in a file `nodeinfo`, located in the `$ADFBIN` directory. For any host not mentioned in this file, the maximum number of parallel processes on that host is one (1). Note that all hosts to be included must be known to PVM as being part of the virtual parallel machine, otherwise no process can be spawned on that host at all.

Two examples may illustrate this. First: on an IBM SP you should run one process per node. Therefore, no file `nodeinfo` is needed. Second: on an SGI Power Challenge with 4 CPUs, you should create a file `$ADFBIN/nodeinfo`, with one record: the name of the host (as it is known to PVM, consult your PVM manual, or see our Installation manual) followed by one or more spaces and then the number 4.

The program run scripts have, in fact, more flags and arguments, for special usage. You can get a survey by typing

```
$ADFBIN/adf -h
```

**The run script “start”**

A note for those who are used to the ADF 2.3 release, where the run script start was used. The start script still exists (in `$ADBIN`) and can be used as before (more or less), but this is not necessary anymore. All program run scripts are links to the start script.

**Files produced by ADF**

The ADF program may generate several output / result files, along with the standard output file. The most important one is `TAPE21`, the general result file. `TAPE21` contains relevant information about the outcome of the calculation. You may want to store this file somewhere under an appropriate name, for future usage. The meaning of any other files that are produced are explained later in this User's Guide.

We will start now with a discussion of the input file for ADF.

**Structure of the input**

Much of the general remarks about input for ADF apply also to related property and utility programs. See the Utilities document for details and any differences.

**Delimiters**

An input record may contain several items. The general rule is that each sequence of characters that does not contain a delimiter is an entity. Delimiters in this context are: 1) the blank or space character `' '`, 2) the comma `' , ` and 3) the equal sign `' = '`. It is assumed throughout that only characters of the Fortran character set are used.
DO NOT USE TABS IN THE INPUT FILE! The program may not see them as delimiters and the effects are hard to predict!

**Uppercase and lowercase**

Virtually all input items are case-insensitive, but take notice of the obvious exceptions: names of files and directories are case-sensitive.

**Keywords**

Input for ADF is structured by keywords, in short: keys. A key is a string of characters that does not contain a delimiter (blank, comma or equal sign). Keys are not case sensitive. Input is read until either the end-of-file condition (EOF) becomes true, or until a record END INPUT is encountered, whichever comes first. (END INPUT is not a key.)

Key-controlled input occurs with two formats. In the first you have only one record, which contains both the key and - depending on the case - associated data: the key argument:

```
KEY argument
```

The whole part of the line that follows after the key is the argument. It may consist of more than one item.

The alternative format is a sequence of records, collectively denoted as a key block. The first record of the block gives the key (which may have an argument). The block is closed by a record containing (only) the word END. The other records in the block constitute the data block, and provide information related to the key.

```
KEY {argument}
data record
data record...(etc.)...
...
END
```

In this manual, when items are optional, such as the argument in the scheme above, they are typed enclosed in curly brackets {}. The { and } characters themselves are not part of the item.

Block type keys may have subkeys in their data block. The subkeys may themselves also be block type keys. The data blocks of block type subkeys, however, do not end with END, but with SUBEND:

```
KEY {argument}
data
```
The items that can be addressed with keys and the keys themselves are listed in the Index.

*Irrelevant keys, misspelling of keys*

Specification of a key that is not relevant in the calculation will go unnoticed. Similarly, if you misspell a key such that it is not recognized, the incorrectly labeled input data will be ignored and the program will proceed as if the intended key had not occurred. This results in the application of pre-defined default values or in an error abort, depending on the case. Therefore, whenever the output suggest that part of your input has been ignored, check the spelling.

*In this context we stress again: be alert on TAB characters: don’t use them at all.*
ADF may recognize a key if it is spelled incompletely, that is, if only some initial substring is given, and also if redundant characters are typed after the end of the key. The reason is that often only a small initial part of the true keyname is checked against the input items. Don't rely on this, however: it is not formally supported and it may get disabled in a next release without further notice.
We advise therefore to stick to the correct key names. In particular, you must avoid to use different abbreviated or elongated forms when a key occurs more than once in input: ADF will likely assume that you want to indicate distinct keys and it will associate only one of them with the key you had in mind.

**Minimal input**

Most keys are optional. Default values apply for omitted keys. Assuming that the defaults are sensible, short input files can often be used. We will examine first the minimal input that is required to run ADF. Having read that part, you can start to do calculations. In particular, you can create *basic atoms* from the database.

**Create mode**

In Create mode the input file can be extremely simple. First, the geometry is trivial: one atom at the origin. Indeed, no coordinates etc. are read from input; any such items are ignored.
Second, the problem is computationally so simple that default settings for precision aspects, such as convergence criteria and levels of numerical integration accuracy, are internally defined to be much more stringent than in normal calculations. These aspects don’t have to be looked after.

In Create mode you need only a one-line input file of the following form:

```
CREATE  Atomtype  Datafile
```

CREATE
  is the keyword. The remainder of the record (atomtype datafile) is the argument.
Atomtype
  is a name for the basic atom that you want to create. The program reads and interprets this name. Therefore, the name must begin with the standard chemical symbol (H, He, Li...) of the element to be created. Optionally the name may have an suffix of the form .text. The suffix begins with a period (.); the part after the period (text) is at your discretion as long as it does not contain a delimiter. A few examples:
The `datafile` part is optional. If you omit it, ADF assumes that the file name is identical to the atom type name, i.e. `CREATE Atomtype` is equivalent to and interpreted as `CREATE Atomtype Atomtype`

In view of the restrictions that apply to the atom type name, the option to use the short form can only be used if the file name has the appropriate format.

To make the input file easier to understand for a human reader you may, for `datafile`, also type `file=Datafile`, where `file=` must be typed as such, and `datafile` is the name of the file.

So you could have a very simple calculation as follows (the ‘creation’ of a Carbon atom):

```plaintext
$ADFBIN/ADF << eor
Create C.dzp
eor
```

The presence of the keyword `CREATE` sets the computational mode of ADF to: create a basic atom. The argument (C.dzp) is then analyzed and found to have as initial part C, telling ADF that we'll be creating a Carbon atom. Since the file-specification part is missing, the data file with the basis set etc. must be the (local) file with the name C.dzp.

More often you will directly address a file (with the basis set) that is not local, but located in the database of your ADF package. The script could then be:

```plaintext
$ADFBIN/ADF << eor
Create C $ADFHOMED/atomicdata/II/C.1s
eor
```

Here you address the file “C.1s” in the database subdirectory II/ (this contains basis sets of double-zeta quality).

<table>
<thead>
<tr>
<th>appropriate names</th>
<th>inappropriate names for an atom type</th>
</tr>
</thead>
<tbody>
<tr>
<td>K</td>
<td>Si-with-core: no period after the chemical symbol</td>
</tr>
<tr>
<td>Li.newbasis</td>
<td>$HOME/atomicdata/C.dzp: not beginning with the chemical symbol</td>
</tr>
<tr>
<td>P.1992/Feb./30</td>
<td>Ga.nocore.smallbasis: contains a comma (a delimiter)</td>
</tr>
<tr>
<td></td>
<td>Sodium.2s: Sodium is not the symbol for this element (Na)</td>
</tr>
</tbody>
</table>

Examples of appropriate (left) and inappropriate (right) atom type names used with the keyword `CREATE`. 

---

**Datafile**

specifies the data file that contains the basis set and related items. It may contain a full path if the file does not reside in the working directory of the job.

The `datafile` part is optional. If you omit it, ADF assumes that the file name is identical to the atom type name, i.e.

`CREATE Atomtype`

is equivalent to and interpreted as

`CREATE Atomtype Atomtype`

In view of the restrictions that apply to the atom type name, the option to use the short form can only be used if the file name has the appropriate format.

To make the input file easier to understand for a human reader you may, for `datafile`, also type `file=Datafile`, where `file=` must be typed as such, and `datafile` is the name of the file.

So you could have a very simple calculation as follows (the ‘creation’ of a Carbon atom):
A considerable number of data files are included in the ADF database. To apply such a file for the creation of a basic atom:

Make a copy of the data file in the directory where you want to run the program. Since the standard data file names satisfy the requirements for atom type names you can now use the simplest option to use the `CREATE` key:

Construct a one line input file in:

```
( CREATE name-of-data-file-copy )
```

Run ADF by typing

```
adf <in >out
```

When the calculation has finished, give the result file `TAPE21` a suitable name and move it to a directory where you build your database of fragment libraries.

Examine `logfile` and `out` to check that everything has gone well.

You may want to define alternative basic atoms, different from those in the standard ADF database, for instance to try out a different basis set developed by yourself. By inspection of one of the standard data files you can see what the contents of such a file should be. A complete description is given in Appendix 1.

You can also create basic atoms corresponding to so-called *Alternative Elements*, with for instance a non-integer nuclear charge or a different mass. See the section Geometry in Chapter 2.3.

**Fragment mode**

In Fragment mode more input is required than in Create mode: you have to specify at least: (1) the atomic positions and (2) how the total system is built up from fragments. We recommended to specify also (3) the point group symmetry.

Example of an input file for the \( \text{C}_2\text{H}_4 \) molecule:

```plaintext
atoms
C  0  0  .6685
C  0  0  -.6685
H  .927  0  -1.203
H  -.927  0  -1.203
H  .927  0   1.203
H  -.927  0   1.203
end

fragments
c  tape21c.dzp
h  tape21h.dzp
end

symmetry  D(2h)

end input
```
Three keys are used: ATOMS, FRAGMENTS and SYMMETRY. The first two are block keys.

ATOMS
defines the atomic positions: each record in the data block contains the chemical symbol of an atom followed by its Cartesian coordinates in Angstroms.
Z-matrix type input of atomic positions is also possible. This will be explained in a later section.

FRAGMENTS
lists the fragment files each record contains a fragment type followed by the corresponding fragment file. In the example the files are local files. Files in other directories are addressed by giving the complete file path.
Note: if a parallel calculation is performed, be sure that each ‘kid’ finds the specified fragment files. This will usually require that the files are not local to the job, but first be moved to some shared volume, and that the references to the fragment files in the input contain the full path. An alternative is to ensure that the (local) files in the parent directory are copied first to the ‘kid’ directories before the parallel calculation starts.

SYMMETRY
specifies the point group symmetry by a Schönfliess type symbol. Appendix 3 contains a complete list of all Schönfliess symbols that are recognized by ADF. If no symmetry is specified ADF will take the true symmetry of the nuclear frame as the user-specified symmetry. If (electric) fields are used, see later, the computed symmetry will take this into account. Note that the computed symmetry may not occur in the list of allowed symmetries (see Appendix 3), in which case you have to explicitly specify the (lower) point group symmetry you wish to apply.

The atomic coordinates must conform to the point group symmetry; the program will check this and abort if the atomic system does not have the specified symmetry. It is allowed, however, to specify a lower symmetry than what is actually present in the set of atomic positions. The specified symmetry determines how results are analyzed and how irreducible representations and subspecies are labeled. It also determines various algorithmic aspects: the program runs more efficiently with the highest possible symmetry.

The spatial orientation of the molecular coordinate system is not arbitrary. ADF requires for each pointgroup symmetry a specific standard orientation. In axial groups for instance, the main rotation axis must be the z-axis. This implies a restriction on how you can define the atomic coordinates under ATOMS. The orientation requirements for all point groups are listed in Appendix 3. If the specified symmetry equals the true symmetry of the nuclear frame ADF will adjust the input orientation of the molecule to the requirements (if necessary). If you have specified a subgroup of the true nuclear symmetry, no such orientation adjustment is carried out and the user has to make sure that his input data yield the correct orientation, lest an error will occur.

Restrictions apply to the symmetry (as specified) of the molecule, related to the symmetries of the fragments as they were stipulated in the preceding fragment calculations. All symmetry operators of the molecule that internally rotate or reflect a fragment but leave it at the same position in the molecule, must also be operators of the symmetry group in which the fragment has been computed. Furthermore, two fragments must not be symmetry-equivalent in the molecule only by an improper rotation. The implied internal reflection of the fragment must be one of the symmetry operators in the point group symmetry that is used in the fragment
calculation and the molecular symmetry group must also contain a proper rotation that maps the two fragments onto each other.

The example of Fig. 1 implicitly assumes that all fragments are single atom fragments. When the fragments are larger the data records in the ATOMS key have to be extended: you must specify which atoms belong together in one fragment.

```
SYMMETRY  T(D)
ATOMS
 Ni 0 0 0 0
 C-1.06 -1.06 1.06 f=co/1
 C-1.06 1.06 -1.06 f=co/2
 C-1.06 1.06 -1.06 f=co/3
 C1.06 -1.06 -1.06 f=co/4
 O1.71 1.71 1.71 f=co/1
 O-1.71 -1.71 1.71 f=co/2
 O-1.71 1.71 -1.71 f=co/3
 O1.71 -1.71 -1.71 f=co/4
END
FRAGMENTS
 co  tape21co.yesterday
 Ni  t  21ni.dzp
END
END INPUT
```

Another sample input file; using a single atom Ni fragment and four molecular CO fragments. The keys SYMMETRY and FRAGMENTS operate as before. Again we have two types of fragments (here: Ni and CO); for each of them, the fragment file is specified.

Under the key ATOMS the chemical symbols and the nuclear coordinates are listed. Added is the f=...-part; f stands here for fragment and tells the program that the carbon and oxygen atoms belong to CO fragments. The last part /n enumerates the individual CO fragments: here you define which C and O belong together in one CO fragment.

The record for Ni contains no f= part, implying the default for this atom: it is a fragment on its own. In the C2H4 example before the default applied to all atoms.

There are more possibilities with the keys ATOMS and FRAGMENTS. This is worked out later. The purpose of this section was to provide a quick and easy start.
2.2 MAIN OPTIONS

In the sections and chapters below all keys are discussed in detail. The keywords are typed in SMALL CAPS, subkeys in ITALIC SMALL CAPS. Schematic examples illustrate how the keys are used, and which keys are block keys or general keys.

Structures like 'key=value' should be read as: type 'key=' as such, followed by a suitable value. Different allowed / eligible values are separated by a bar (|). Brackets [ ] around an item, argument, or value indicate that it is optional.

We proceed with a discussion of the most important keys: keys to set the precision of the calculation, keys to regulate the model Hamiltonian (in particular the Density Functional) and keys to specify and control the run type.

Parallel Execution

Control of parallelization on the first line of the input file, as in release 2.3, has been disabled in ADF1999 (it is ignored). You now control the parallelization by (a) environment variables and (b) command line argument to the program run script. See the discussion in the introduction, the Installation manual and the Examples document.

Run Types

The different run types are characterized by how the geometry is manipulated:

SinglePoint
The SCF solution is computed for the input geometry.

GeometryOptimization
The atomic coordinates are varied in an attempt to find a (local) energy minimum. One may let all coordinates free or only a subset, keeping the others frozen at their initial values.

TransitionState
Search for a saddle point. Similar to a GeometryOptimization, but now the Hessian at the stationary point presumably has one negative eigenvalue.

LinearTransit
The geometry is modified step by step from an initial to a final configuration. All of the coordinates or only a subset of them may be involved in the transit. The coordinates to be modified are the LinearTransit parameters. For each of the LinearTransit points (geometries) the computation may be a Single Point SCF calculation or a GeometryOptimization. In the latter case only those coordinates (or a subset of them) can be optimized that are not LinearTransit parameters. The LinearTransit feature can be used for instance to sketch
an approximate reaction path in order to obtain a reasonable guess for a transition state, from where a true TransitionState search can be started.

**IRC or IntrinsicReactionCoordinate**

Tracing a reaction path from a transition state to reactants and/or products. A fair approximation of the transition state must be input. The end-point(s) – reactants / products – are determined automatically.

**Frequencies**

Computation of force constants and from these the normal vibrational modes and harmonic frequencies. The force constants are calculated by numerical differentiation of the energy gradients at the equilibrium geometry and the slightly deviating geometries (making small displacements of the atoms). No analytical second derivatives have yet been implemented.

For all features that involve changes in geometry, i.e. all run types except the SinglePoint, it is imperative that you use single-atom fragments. Larger molecular fragments can only be applied in SinglePoint calculations.

Three keys are involved in the specification of the geometry and its manipulation:

**ATOMS**

sets the atomic (starting) positions.

**GEOMETRY**

Controls the run type and strategy parameters, such as convergence thresholds and the maximum number of geometry steps to carry out.

**ATOMS and GEOMETRY**

These two keys together are sufficient for a straightforward Optimization, TransitionState search, IRC run or a Frequencies computation. (Of course, you also need to specify the FRAGMENTS key.

**GEOVAR**

May be used to impose constraints, for instance when only a subset of all coordinates should be optimized. GEOVAR may also be used in a LinearTransit run to define the LinearTransit parameters and their initial and final values.

Constraints and LinearTransit parameters may also be controlled within the ATOMS block if a MOPAC-style input format is used, see below.

**Runtype control and strategy parameters**

With the block key GEOMETRY you define the runtype and strategy parameters.

```
| GEOMETRY  | {RunType  |  { RunTypeData} } |
| RunType   |   (RunTypeData) |
| END |
```

**RunType**

Can be:
- SinglePoint or SP
- GeometryOptimization or GeoOpt or GO
- TransitionState or TS
- IntrinsicReactionCoordinate or IRC
- LinearTransit or LT
- Frequencies or FREQ

If omitted the run type is GeometryOptimization.

If the key GEOMETRY is not used at all the run type is SinglePoint.

The run type specification can be given as argument to the geometry key, or in the data block, but not both.

For some run types additional data may be given after the run type specification.

RunTypeData
(Optional) further specifications, depending on the run type. See the sections below.

Omission of the GEOMETRY key altogether effectuates a SinglePoint calculation. A straightforward optimization, with all features that can be set with geometry at their default values, is activated by supplying the key with an empty block:

```
| GEOMETRY
| END
```

More subkeys are available in the GEOMETRY block than just the run type specification. They are used to control strategy parameters such as convergence criteria. All subkeys are optional: default values take effect for those omitted. Some of the subkeys are only meaningful for certain run types. They will be ignored for other run types.

The initial approximation of the Hessian matrix, which may affect the number of optimization steps that are carried out to reach convergence, is not controlled by the key GEOMETRY, but by the key HESSDIAG, and/or by the key GEOVAR. See the section Initial Hessian.

### Atomic Coordinates

The input of (initial) atomic positions as Cartesian coordinates has been mentioned already in the minimal-input examples in Chapter 2.1. Alternatively they may be given in Z-matrix form.

```
| ATOMS           {Cartesian / Zmatrix / MOPAC} |
| {N} Atom Coords {F=Fragment}               |
| ...                                        |
| End                                        |
```

**Cartesian or Zmatrix or MOPAC**

Specifies the type of coordinates. Default (no specification) is Cartesian. Instead of Zmatrix you may also type internal.

MOPAC is a special variety: the subsequent records in the data block are MOPAC style Z-matrix input for the atomic system, see example below.
This is an optional integer by which you may number the atoms. The numbers should be 1, 2, 3, et cetera if any reference is made to them in other parts of input. The reason for this restriction is that ADF numbers the atoms internally according to their occurrence in the input file and it applies this internal numbering when any subsequent references are interpreted.

**Atom**

The name of an atom type. It must begin with the standard one- or two-character symbol for the chemical element: H, He, Li, and so on. Optionally it may be appended by .text, where text is any string (not containing delimiters). Examples: H, Mn.3, Cu.dz-new.

Dummy atoms may be useful in the construction of a Z-matrix, for instance to obtain a set of internal coordinates that reflect the symmetry of the molecule better. They may also be useful in a Z-matrix to avoid an ill-defined dihedral angle, which occurs when three (almost) co-linear atoms span either of the two planes that define the angle. In geometry optimizations this must absolutely be avoided if such internal coordinates are used as optimization parameters.

Dummy atoms are input with the chemical symbol XX. XX-type atoms can be inserted in the list of atoms like any other atom types. The name (XX) can have a suffix of the form .text. No fragment files must be supplied for dummies. There are no symmetry constraints on the positions of the dummies. The dummies serve only to set up the Z-matrix in a proper way.

**Coords**

This specifies the coordinates of the atom. If Cartesian coordinates are used the x, y, z values must be given. For Z-matrix coordinates you put first the three connection numbers, then the values of the bond length, bond angle and dihedral angle. Example:

```
Ge  2  1  5  2.1  95.3  24.8
```

defines that a Germanium atom is located with a distance 2.1Å from the second atom in the input list, that the angle (Ge-atom2-atom1) is 95.3 degrees and that the dihedral angle between the planes (Ge-atom2-atom1) and (atom2-atom1-atom5) is 24.8 degrees.

To avoid any confusion as regards the direction (sign) of the dihedral angle, here is the definition used in ADF: Let the connection numbers for an atom P refer to the atoms Q, R and S, in that order. Choose a local coordinate frame such that Q is at the origin, R on the positive z-axis and S in the xz–plane with a positive x-value. The three Z-matrix coordinates bond length, bond angle and dihedral angle of P are then precisely its spherical coordinates $r$, $\theta$, and $-\phi$: the distance to the origin, the angle that PQ makes with the positive z-axis (0..$\pi$) and the negative of the angle that the projection of PQ on the xy-plane makes with the positive x-axis ($0..2\pi$, or $-\pi..+\pi$).

The connection numbers and internal coordinate values of the first atom in a Z–matrix have no meaning. Similarly, the second atom requires only a bond-length specification and the third atom only a bond length and a bond angle. However, for each atom three connection numbers are read from input and interpreted, and you must therefore supply zeros for them if they don’t refer to any atoms. The corresponding meaningless Z-matrix coordinate values can be omitted. More in general: missing coordinate values are set to zero (also
for Cartesian coordinates input). Z-matrix values that are meaningless because they correspond to zero connec-
tion numbers are ignored, whatever their value is in the input file.

In a Z-matrix definition the three reference atoms, with respectively 3, 2, and 1 connection numbers equal to
zero, do not have to be the first three in the input list. The program will scan the list for any atom that has 3
connection numbers zero, then for one that has only a bond length specification, etc. If the Z-matrix is not
properly defined, for instance if more than one atom occurs with all three connection numbers equal to zero,
or when not every atom is somehow connected to all others, the program will abort.

F=Fragment

Specifies that the atom belongs to a particular fragment. The fragment name must be of the form
fragtype/n, where fragtype is the name of one of the types of fragments in the molecule. The integer
n, after the slash, counts the individual fragments of that type. The numbering suffix /n is not required if
there is only one fragment of that type.

When f=fragment is omitted altogether, the fragment type is taken to be the atom type that was specified
earlier on the same line. (The numbering /n is then added automatically by the program, by counting the
number of times that this single-atom fragment type occurs in the list of atoms).

MOPAC

The MOPAC style input requires that the records in the data block have the following format:
atomtype   distance idist angle iangle dihedral idehedral
The three internal coordinate values (distance, angle, dihedral) are each followed directly by the connection
number.

Atom type is not identical to chemical element: an atom type is defined by all characteristics of the basic atom
to which it in fact refers: the nuclear charge, the basis functions, the frozen core, the density functional and any
other features that were applied in generating that basic atom.

As mentioned before, the point group symmetry specified in input with a Schönfliess type symbol puts
restrictions on the orientation of the atomic system. Unless the input-specified symmetry equals the true
symmetry of the nuclear frame (in which case ADF will adjust the orientation of the molecule, if necessary), the
user must take care of this by supplying the Cartesian coordinates (in the appropriate orientation). If a subgroup
of the true nuclear symmetry is used and Z-matrix format is used for the coordinates, the program will place the
atoms in the standard Z-matrix frame: first atom at the origin, second on the positive x-axis, third in the xy-plane
with positive y-value.

Dummy atoms may be placed asymmetrically. If the atomic coordinates are input as Cartesians, any dummy
atoms are irrelevant. Their coordinates will be printed but otherwise they are ignored.

Input items are generally case insensitive. Exceptions are the names of files and directories. Since (to be
discussed below) the name of the fragment type as it is defined under ATOMS (explicitly with the f=option, or
implicitly as the name of the atom type) might also directly indicate the fragment file, the specification of
fragment types is in principle case-sensitive. Errors may occur if you are sloppy in this respect.
However, you must not give different fragment types names that differ only by case: at various places in the
program fragment type names are compared in a case-insensitive way.
**Mixed Cartesian and Z-matrix coordinates**

The key ATOMS can also be used to supply coordinates in a format that gives the values for the cartesian coordinates and the connection matrix, which defines a Z-matrix.

```
ATOMS    ZCart
{N}     Atom   Coords   {F=Fragment}
...
END
```

**ZCart**
Signals this particular format for the coordinates

**Coords**
As for Z-matrix input: three integers and three real values. The integers are the connection numbers that define the Z-matrix structure, but the reals are the *Cartesian* coordinates.

With ZCart input, the z-matrix is internally generated from the Cartesian coordinates and the connection numbers.

This feature is convenient when for instance Cartesian coordinates are easily available but you want to run a Geometry Optimization in *internal* coordinates, for which a Z-matrix structure is required. The ZCart option comes in handy also to satisfy symmetry-related orientation requirements when you basically wish to use Z-matrix coordinates.

With ZCart input the program defines the *type* of coordinates in the input file as *Cartesian*. This is significant in Geometry Optimizations, where the optimization variables are by default taken as the input coordinate type.

**Geometry Optimization**

Geometry Optimizations in ADF is based on a quasi Newton approach [6-8], using the Hessian for computing changes in the geometry so as to make the gradients vanish. The Hessian itself is initialized (for instance based on a force field) and updated in the process of optimization.

Several subkeys in the GEOMETRY block can be used for control of the Geometry Optimization procedure and related strategy parameters.

```
GEOMETRY
OPTIM     {Cartesian / Internal}   {All / Selected}
ITERATIONS Niter {Niter2}
HESSUPD    HessUpdate
CONVERGE   {E=TolE}   {Grad=tolG}   {Rad=TolR}   {Angle=tolA}
STEP       {Rad=MaxRadStep}   {Angle=MaxAngleStep}
END
```
**OPTIM**

Cartesian or Zmatrix (equivalently: internal) specifies the type of coordinates in which the minimization is carried out. By default the coordinate type is applied that was used in the ATOMS key for the input of the (initial) atomic positions. (Cartesian if ATOMS were input in ZC art format).

Cartesian optimization is allowed if the ATOMS were input in Z-matrix format, but no constraints (see the key GEOVAR) can then be used: all coordinates are optimized. An attempt to explicitly freeze variables may result in an error abort. Optimization in Z-matrix coordinates is not allowed if only Cartesian coordinates were supplied in ATOMS: the program does not construct a Z-matrix by itself. One should then use the Zcart format: give Cartesian coordinates and supply the structure of the Z-matrix. Again, in this case you cannot use constraints.

**Selected**

Only those coordinates are optimized that are defined with the key GEOVAR.

**All**

(The default value) means that in principle all atomic coordinates will be varied. The key GEOVAR may modify this in the sense that some of the coordinates can be kept frozen or can be forced to remain equal to some other coordinates.

**Niter**

The maximum number of geometry iterations allowed to locate the desired structure. The default is 30. This is a fairly large number. If the geometry has not converged (at least to a reasonable extent) within that many iterations you should sit down and consider the underlying cause rather than simply increase the allowed number of cycles and try again.

**Niter2**

An optional second parameter that plays only a role in a LinearTransit run, see the LT section. It must not be used in other runtypes.

**HessUpdate**

Specifies how the Hessian matrix is updated, using the gradient values of the current and the previous geometry.

Recognized values are:

(i) **BFGS**: Broyden-Fletcher-Goldfarb-Shanno
(ii) **MS**: Murtagh-Sargent
(iii) **DFP**: Davidon-Fletcher-Powell
(iv) **FS**: Fletcher switch
(v) **HOSHINO**: Hoshino

default: BFGS.

**Converge**

Convergence is monitored for three items: the energy, the Cartesian gradients and the estimated uncertainty in the (chosen type of optimization) coordinates. For the latter, lengths (Cartesian coordinates, bond-lengths) and angles (bond-, dihedral-) are considered separately.

Convergence criteria can be specified separately for each of these items:

**TolE**

The criterion for changes in the energy, in Hartrees. Default: 1e-3.

**TolG**

**TolR**

Refers to changes in the Cartesian coordinates or bond lengths, depending on in what coordinates you optimize, in angstrom. Default: 1e-2.

**TolA**

Refers to changes in bond- and dihedral angles, in degrees. This is only meaningful if optimization takes place in Z-matrix coordinates. Default: 0.5 degree.

If only a numerical value is supplied as argument for `CONVERGE`, rather than a specification by name, it is considered to apply to the gradients (only). The other aspects (energy and coordinates) retain their default settings then.

**Remarks:**

1. Molecules may differ very much in the stiffness around the energy minimum. Application of standard convergence thresholds without second thought is therefore not recommended. Strict criteria may require a large number of steps, a loose threshold may yield geometries that are far from the minimum as regards atom-atom distances, bond-angles etc. even when the total energy of the molecule might be very close to the value at the minimum. It is good practice to consider first what the objectives of the calculation are. The default settings in ADF are intended to be reasonable for most applications but inevitably situations may arise where they are inadequate.

2. The numerical integration precision parameter `accint` (see the key `INTEGRATION`) should match the required level of convergence in gradients. Gradients are computed as a combination of various integrals that are evaluated by numerical integration in ADF. The integral values have a limited precision: roughly speaking the `accint` value is the number of decimal digits in the value of the integrals that are correct. As soon as the gradients, which are supposedly zero at the exact energy minimum, are of the order of $10^{-\text{accint}}$ they will, in worst cases, become arbitrary and any attempt to continue convergence may not really improve things. You may even find that, due to the numerical-integration noise, the geometries start moving around in a random fashion, while the gradients vary more or less arbitrarily. As a general rule: set the `INTEGRATION` value higher (by at least 1.0) than the convergence level required for the gradients. Example: if gradients are to be converged to 1e-3, set `INTEGRATION` 4.5 (implying: higher by 1.5 than the gradients convergence level).

3. The convergence threshold for the coordinates (TolL, TolA) is not a reliable measure for the precision of the final coordinates. Usually it yields a reasonable estimate (order of magnitude), but to get accurate results one should tighten the criterion on the gradients, rather than on the steps (coordinates). The reason for this is that the program-estimated uncertainty in the coordinates is related to the used Hessian, which is updated during the optimization. Quite often it stays rather far from an accurate representation of the true Hessian. This does usually not prevent the program from converging nicely, but it does imply a possibly incorrect calculation of the uncertainty in the coordinates.

**Step**

Controls that changes in geometry from one cycle to another are not too large:

**MaxRadStep**

An upper bound on changes in Cartesian coordinates or bond lengths, as the case may be. Default: 0.3 angstrom when optimization is carried out in internal coordinates, 0.15 angstrom for Cartesian optimizations.
MaxAngleStep
Similarly this option limits changes in bond angles and dihedral angles. Default: 10 degrees.
Input for MaxRadStep, MaxAngleStep is in angstrom and degrees respectively, independently of the UNITS used for atomic coordinates input.
Note: Optimization of ring structures carried out in internal (Z-matrix) coordinates is sometimes tricky due to the ill-defined last segment of the ring. When problems arise, try Cartesian optimization or consider using smaller limits on the steps (in particular the angles) so as to prevent the program from breaking the ring beyond repair.

Transition State
A transition state (TS) search is very much like a minimization: the purpose is to find a stationary point on the energy surface, primarily by monitoring the energy gradients, which should vanish. The difference between a transition state and a (local) minimum is that at the transition state the Hessian has a negative eigenvalue. Because of the similarities between a minimization and a TS search most subkeys in GEOMETRY are applicable in both cases, see the Geometry Optimization section. However, practice shows that transition states are much harder to compute than a minimum. For a large part this is due to the much stronger anharmonicities that usually occur near the TS, which threaten to invalidate the quasi-Newton methods to find the stationary point. For this reason it is good advice to be more cautious in the optimization strategy when approaching a Transition State and for some subkeys the default settings are indeed different from those for a simple optimization. In addition, certain additional aspects have to be addressed.

```
GEOMETRY
  TransitionState (Mode=Mode) (NegHess=NegHess)
END
```

NegHess
The number of negative eigenvalues that the Hessian should have at the saddle point. In the current release it is a rather meaningless key, which should retain its default value (1).

Mode
Controls the first step from the starting geometry towards the saddle point: it specifies in which direction the energy is to be maximized while the optimization coordinates will otherwise be varied so as to minimize the energy. A positive value means that the eigenvector #mode of the (initial) Hessian will be taken for the maximization direction. This means: put all Hessian eigenvalues in ascending order, ignoring those that correspond to impossible movements (rigid rotations and translations, symmetry breaking) and then take the eigenvector of #mode in the remaining list. A negative value for mode instructs the program to take the eigenvector that makes the largest change of the abs(mode)-th atomic coordinate (counting only the coordinates that are allowed to be changed independently, in order as they occur in the input list of coordinates under ATOMS).
Default: mode=1. Generally the program performs best with this default: it will simply concentrates on the mode with the lowest eigenvalue, which should of course finally be the path over the transition state (negative eigenvalue!).
After the first geometry step, the subsequent steps will attempt to maximize along the eigenvector that resembles most (by overlap) the previous maximization direction until the Hessian is found to have a negative eigenvalue. At that point the program switches to that mode. As soon as the program has focused on the eigenvector with the lowest eigenvalue (mode=1) the overlap criterion to select the search direction is internally discarded and subsequently only the lowest eigenvector is taken. An input value mode=0 effectuates this immediately: the direction with the lowest eigenvalue will be the maximization direction for all iterations.

As mentioned before, the other subkeys have the same functionality as for minimizations, but different defaults or options may apply:

**HessianUpdate**
Different (fewer) options apply now:
(i) **Powell**: Powell
(ii) **BFGS**: Broyden-Fletcher-Goldfarb-Shanno
(iii) **DFP**: Davidon-Fletcher-Powell

**default**: Powell

**MaxRadStep**
Default: 0.2 angstrom for Z-matrix optimization, 0.1 angstrom for Cartesian optimization.

**MaxAngleStep**
Default: 5 degrees.

Note: in Transition State searches precision is often much more critical than in minimizations. One should set the Numerical Integration precision at a fair value (4.5 at least). The *default* (i.e. automatic) value is 5.0 in a Transition State search.

**Linear Transit**

In a Linear Transit (LT) run you define a number of atomic coordinates (at least one) to be the LT *parameters*: these get an initial and a final value. The LT is defined as the simultaneous linear change of these parameters from their initial to their final values. This is carried out in a number of equidistant steps. The total number of LT *points* is specified on input. At each LT point the remaining atomic coordinates – those that are not LT parameters – may or may not be optimized: the (final) structure and energy at each LT point are computed. A Linear Transit (LT) run is therefore just a sequence of (related) constrained Geometry Optimizations.

The LT scan may be used for instance to sketch an approximate path over the transition states between reactants and products. From this a reasonable guess for the Transition State can be obtained which may serve as starting point for a true transition state search for instance.

Whenever a **GEOMETRY** subkey is applicable in a Geometry Optimization, it will apply in a Linear Transit run in each of the optimizations that are carried out at the distinct Linear Transit points, and the same default values apply.

The runtype has to be specified. Additional specifications are optional.
GEOMETRY
    LinearTransit {NPoints}
END

NPoints
The number of LT points for which an optimization will be carried out.
If no value is supplied the default takes effect: 5.

There are a few obvious differences between a single optimization and a LT run. Most important is that the
coordinate(s) that describe the LT path, the LT parameters, cannot be optimized: at each of the LT points they
are frozen. This implies that technically speaking at each LT point a constrained optimization is carried out. One
of the consequences is that the ATOMS coordinate type – Cartesian or Z-matrix – must also be the optimization
coordinate type. The LT parameters themselves must be defined with the key GEOVAR, see below.

It is possible to freeze all coordinates so that the LT run is similar to a sequence of Single Point runs. However,
energy gradients will be computed at each step, so that more CPU time is spent at each LT point than for just a
Single Point calculation.

The number of LT points by which the path is traced is defined by the npoints argument to the subkey
LINEARTRANSIT. It is possible to execute only a subset of these points, usually with the purpose to complete
the calculation by using the restart facility of ADF. In this way you can break down a very large calculation into
several smaller ones, or have the opportunity to check how things have been going for the first few LT points
before deciding whether a continuation is useful. This may be achieved of course by simply defining different
start- and end-values for the LT parameters in a related series of calculations, but it is more comfortable to
specify the complete path once and just execute parts of it at a time. This is accomplished by giving a second
value to the ITERATIONS subkey in the geometry block.

... ITERATIONS Niter Niter2 ...

Niter
The first argument of the subkey ITERATIONS in the geometry block, controlling the maximum number of
iterations allowed to reach convergence, applies now for each LT point separately.

Niter2
The second argument specifies the maximum number of LT points to calculate in this run. If omitted (default)
the whole LT scan is completed. Doing only part of the scan may be combined with the restart feature, so that
the remainder can be done in a continuation run. See the RESTART key.
A too large value of LT points is automatically adjusted: no more LT points are computed than required
to complete the LT path as defined by the LINEARTRANSIT subkey. A negative or zero value is not accepted
and internally reset to one (1).
WARNING: if you use the QMMM functionality in combination with a Linear Transit, then only the coordinates of the true QM atoms can be used as LT parameters, no MM atoms must be involved in the LT parameter set.

**Intrinsic Reaction Coordinate**

The path of a chemical reaction can be traced from the Transition State to the products and/or to the reactants, using the Intrinsic Reaction Coordinate method (IRC) [9, 10]. The starting coordinates should be a fair approximation of the Transition State. The final values at the endpoint(s) – reactants, products – are computed. The IRC path is defined as the steepest-descent path from the Transition State down to the local energy minimum. The energy profile is obtained as well as length and curvature properties of the path, providing the basic quantities for an analysis of the reaction path. Additional properties along the path (dipole moment, atomic charges) are computed.

Technically speaking the path is computed by taking small steps along the path meanwhile optimizing all atomic coordinates orthogonal to it so that, like in a Linear Transit run, a sequence of constrained optimizations is carried out. The total number of steps along the path is not known in advance. The maximum number of such steps can be set in input. If the path is not completed in the run, a RESTART can be used to finish it. Each of the constrained optimizations in the run is treated as if it would be in a Linear Transit run: convergence thresholds, maximum numbers of optimization iterations et cetera are set with subkeys in the GEOMETRY block.

You can set the IRC runtype by typing it in the GEOMETRY block

```
GEOMETRY
  IRC   {Forward} {Backward} {N=Points} {Step=Step} {StepMax=StepMax}
    {StepMin=StepMin} {Start=Start}
END
```

**IRC**

The runtype

**INTRINSICREACTIONCOORDINATE** would also be recognized.

**Forward, Backward**

Specifies execution of the two possible paths from the Transition State to the adjacent local minima. By default both are computed. If **Forward** is specified only, the other path is turned off and similarly for **Backward**. For the definition of which of the two directions down from the Transition State to an adjacent minimum is ‘forward’ see below.

**Points**

The maximum number of IRC points computed in the run, for both paths together and including the initial (TS) central point (as far as applicable). Default 100.

**Step**

The (initial) step length when proceeding from one IRC point to another along the path. The difference between two geometries, to which the step quantity applies, is measured in mass-weighted coordinates. The
default value for step is $0.2 \text{ amu}^{1/2} \text{ bohr}$. Larger steps reduce, in principle, the required number of IRC points from the transition state to the minimum, but usually at the expense of more optimization steps at each of the points so the net gain in computation time may not be very large, or even negative. The default size is rather conservative and in many cases you may increase it to save a few steps. However, to some extent you can leave that to the program. When going from one point to the next, the program will increase or decrease the stepsize depending on whether or not the previous point to a large number of geometry cycles to converge. The adjusting algorithm also tends to be more cautious when the successive IRC points show more drastic changes in the atomic geometrical configuration. In all cases the IRC step sizes remain between pre-set maximum and minimum values, see the next items.

**StepMax**

The maximum step length that the program will select in the step-adjusting algorithm. Default: 1.0 or 10 times the initial step length, whichever is larger.

**StepMin**

The minimum step length that the program will select in the step-adjusting algorithm. Default: 0.03 or 0.3 times the initial step length, whichever is smaller.

**Start**

Defines how the initial direction of the path is chosen to move away from the Transition State. It does not imply whether the first step along this direction is taken positively or negatively. See for this aspect the section about Forward/Backward IRC paths.

The admissible values for start are:

- **GRAD**: compute the gradient and take that direction right from the start. Obviously, if we start perfectly at the Transition State this will be meaningless since the gradient vanishes there completely.
- **READ**: the initial path direction is read in with the key IRCSTART, see the section IRC Start Direction.
- **HESS N**: the initial path coincides with the n-th Hessian eigenvector (ordered by ascending eigenvalues); N must be an integer in the appropriate range.

The default (omission of any start specification at all) is the first Hessian eigenvector, presumably corresponding to the path over the Transition State (negative Hessian eigenvalue!).

**IRC start direction**

As mentioned above, the IRC path is initialized by a first step away from the Transition State. If perfect information is available this should be along the unique Hessian eigenvector with a negative eigenvalue. Therefore, it is preferable to supply (with a restart file) a good approximation of the Hessian at the Transition State. This can be computed in a Frequencies run. In many cases the automatic internally generated (force field based) Hessian will not severely disturb the procedure and may only require a few more initial search steps for the right direction to take, while saving a potentially expensive Frequencies calculation.

If you decide to use a precalculated Hessian, then usually the approximate Hessian resulting from a Transition State run will be good enough. The latter approach is more attractive of course since the TS run will usually be done anyway, as a preliminary to the IRC run, while an additional Frequencies run would be very demanding. At the other hand, Transition State runs often require a preceding Frequencies run. In such case, the Frequencies result file may be used both for the TS run and for the IRC run. The fact that the Frequencies run may have been performed not at the exact TS may affect slightly the adequacy for using it as a start-up for the IRC run, but this is likely not significant.
In some case you may want to specify the initial direction of the IRC path explicitly. This is done as follows:

```
IRCSSTART
  data
  data
  ...
END
```

**IRCSTART**

A block-type key. The data in the data block are values for all atomic coordinates (Cartesian or Z-matrix, as the case may be) that are not frozen and not (by GEOVAR) explicitly instructed to remain equal. All such coordinate data together define a direction vector in the space of all (free) coordinates, which then serves as the initial segment of the IRC path.

Note that only a direction vector is defined here: the size of the total vector plays no role. Furthermore, the initial step may be in the positive or negative direction along the so-defined initial path, see the section Forward / Backward IRC paths.

**Forward / Backward IRC paths**

Obviously there are two IRC paths down the transition state: Forward and Backward. We would have liked to chemically define forward and backward by determining (in advance!) which of the endpoints is reactants and which products. This is not well doable in practice. Therefore we define the directions in terms of the initial path vector: select simply the atomic coordinate with the largest (absolute) change in the initial vector and define Forward as the direction in which this coordinate increases and Backward as the direction in which it decreases.

**Optimization: Special Features**

**GEOVAR: constrained optimization, Linear (synchronous) Transit parameters**

The block key **GEOVAR** is used
- To put restrictions on the number of coordinates that are varied and
- To define Linear Transit parameters and assign them initial and final values.

**GEOVAR** can also be used to assign (initial) values to coordinates without other implications, but this feature is accidental.

In the input section of atomic coordinates (key **ATOMS**) identifiers (names) may be used rather than numerical values wherever coordinate values are expected: x, y, z in case of Cartesian coordinate input; r, θ, φ in case of internal coordinates. All such identifiers must then be specified under **GEOVAR** and assigned a value.
GEOVAR

Name  Data
....
END

Name
An identifier that can be used in place of a numerical value for one or more of the atomic coordinate values under ATOMS.

Data
Either of three formats:
1 A single value simply assigns the value to the corresponding atomic coordinate(s).
2 Two values (separated by a delimiter) imply that the corresponding atomic coordinate is a Linear Transit parameter. The two numbers are the initial and final values respectively for the Linear Transit path.
3 A single value followed by a letter F assigns the value to the corresponding atomic coordinates and specifies that these coordinates are frozen: they will not be optimized.

As regards the optimization of coordinates other than the frozen ones and the LT parameters, the meaning and effect of the input under GEOVAR depends on the subkey OPTIM in the GEOMETRY block:

If selected has been set, optimizations are carried out only for the coordinates that are referred to under GEOVAR (and that are not Linear Transit parameters or Frozen). All coordinates that were input as simple numerical data under ATOMS are kept frozen then.

Alternatively, if selected has not been set (: all, the default) all atomic coordinates are optimized (except the Linear Transit parameters and the explicitly frozen coordinates). In that case, each assignments under GEOVAR other than to freeze the coordinate or to define it as a Linear Transit parameter simply assigns an initial value to the pertaining coordinates. In this respect it is not different from typing the numerical value directly in the ATOMS block, except for the next aspect.

The same identifier may be used for two or more coordinates in ATOMS. If they are varied (i.e. if they are not frozen) they will forcibly be kept equal throughout the optimization so that they constitute only one degree of freedom. Don't use the same GEOVAR variable for coordinates that belong to atoms of different chemical types or to different types of coordinates (an angle and a bond length for instance). It is not sensible to do so and it will very probably lead to an error abort or to stupid results.

It is allowed to put as atomic coordinate under ATOMS minus a GEOVAR variable name, i.e. the name preceded directly by a minus sign (without a blank in between!). The coordinate will then be kept equal, but with opposite sign, to coordinates that are defined by the same variable without the minus sign. The initial (and final, in case of a LT run) value for that coordinate is the negative of the GEOVAR value.

**Constrained optimizations: coordinate types**

Restricted optimizations are performed by freezing certain coordinates, by explicitly referring to one and the same GEOVAR identifier for different coordinates, or by using the selected option. In addition, they are
implicit in each Linear Transit or IRC run. All restricted optimizations demand that the type of optimization variables (Cartesian or Z-matrix) equal the type of coordinates used in ATOMS. Zcart input under ATOMS is considered to be Cartesian in this respect.

If this is violated in a Linear Transit calculation the program will abort. If you apply the Frozen option under GEOVAR, while not using the same coordinate type for ATOMS as for optimization, an error will occur. If you refer to the same GEOVAR identifier for distinct coordinates while the ATOMS and the optimization types of variables do not match, the program will continue and assume that you only have assigned the same starting values to the pertaining coordinates. No equality constraints will be in effect then during the optimization.

**Symmetry versus constraints**

The symmetry of the atomic system defined by the input Schönfliess symbol is preserved during optimization. If the input information (which coordinates are kept frozen and which are optimized) conflicts with the symmetry, the latter will prevail and an error exit may occur. In the program the geometric step is first computed according to the user-specified constraints and then symmetrized. This symmetrized geometry update is applied, regardless whether this results in frozen coordinates being changed.

Input specifications that are in conflict with the point group symmetry may lead to an error abort.

**Z-matrix and symmetry**

If the structure of the Z-matrix does not reflect the symmetry of the molecule and constraints are applied the program may encounter algorithmic problems to match all demands. As a result some of the frozen coordinates may be found to change. Usually these changes are very small. To cure this: build the Z-matrix in a symmetric way.

**Symmetry in a Linear Transit**

In a Linear Transit run it is imperative that the complete Linear Transit path as defined by the parameters conforms to the specified symmetry. If such is not the case, an error will occur or possibly the program will continue but not produce correct results. Note that when no symmetry is specified in input, the initial geometry defines the specified symmetry.

**Summary of GEOVAR, OPTIM, and ATOMS**

For unconstrained optimization: don't use GEOVAR, apply OPTIM if Cartesian optimization is required while the data in the ATOMS block was in Z-matrix format or when Z-matrix optimization is required while the ATOMS input was in ZCart format. Provide the atomic coordinates (ATOMS) directly as numerical data.

For optimizations where only very few coordinates are frozen: use GEOVAR to set a few coordinates to frozen and/or to enforce equality of optimization coordinates whose values should remain equal. Don't use OPTIM: the type of optimization coordinates - Cartesian or internal - must be identical to what is used in the ATOMS input part because you're using constraints now. In the ATOMS section, use identifiers for the frozen coordinates and for those that should satisfy equality conditions; use numerical input for all other (optimization) coordinates.
For very limited optimization: turn on the selected option with \texttt{OPTIM} and assign with \texttt{GEOVAR} initial values to the coordinates that you want to optimize. In the \texttt{ATOMS} input use identifiers for these coordinates. The numerical input coordinates are kept frozen automatically now.

\textbf{Initial Hessian}

In a Geometry Optimization (or Transition State search) the Hessian matrix — second derivatives of the energy with respect to changes in coordinates — is updated while the program steps around in an attempt to find the (local) energy minimum. The quality of the initial Hessian may have a considerable impact on the required number of steps to reach geometric convergence.

By default the initial Hessian is read from a restart file — see the key \texttt{RESTART} — or constructed from a force field \cite{11} that is implemented in the program. In the latter case the user can modify the so-generated initial Hessian in four ways:

1. By setting all diagonal elements to some constant.
2. By defining three constants, one for distances (or Cartesian displacements, as the case may be), one for bond angles, and one for dihedral angles. All diagonal elements of the Hessian are adapted accordingly.
3. By supplying a list of diagonal values.
4. By giving diagonal-Hessian values for one or more specific coordinates.

For each element $i$ for which a diagonal Hessian value $H_{ii}$ is supplied the off-diagonal elements $H_{ij}$, (all $j \neq i$) are set to zero.

A combination of the above options is possible. The rules of how combinations are interpreted by the program are:

- The program first initializes the Hessian using the force field (or restart data).
- If a single constant (1) or three constants (2) are supplied, all diagonal elements are adjusted (and all off diagonal elements are set to zero).
- If a list of diagonal values is supplied (3), this overrides the first so many values of the diagonal. Such a list is not required to cover all diagonal elements. If the list is shorter than the dimension of the Hessian, i.e. the number of atomic coordinates, only the first so many elements will be adjusted.
- If any individual elements are supplied specifically (4), their values are replaced in the diagonal defined thus far.

All input values of the Hessian are in units of Hartree/bohr$^2$ for Cartesian coordinates and bond lengths. Hartree/radian$^2$ for bond angles and dihedral angles.

The first 3 options are controlled by the key \texttt{HESSDIAG}:

\begin{verbatim}
  HESSDIAG  {General}
  {    List
  END   }
\end{verbatim}
HESSDIAG

A general key: it has either an argument (General), or a data block (List). It is also possible to supply the argument and the data block, but this requires that the continuation symbol (&) is given after the argument, separated from the argument by at least one blank.

General

Must be either a single numerical value, or one or more named specifications of options, in the format optionname=value.

If a single numerical value is given, this value is assigned to all the options that are available. If the named-option format is applied, any named options that are not found get the value 1.0.

The options are: rad=radvalue to assign a value to all Hessian diagonal elements that refer to distance coordinates (bond length in case of Z-matrix coordinates, Cartesian coordinates otherwise),
ang=angvalue to assign a value to all elements that refer to bond angles, and finally dih=dihvalue for dihedral angles.

ang and dih are not significant in Cartesian optimizations.

List

A list of numerical values, which may expand over any number of lines. If \( n \) numbers are supplied, they are assigned to the first \( n \) diagonal elements of the Hessian. The remaining diagonal elements, if any, are not affected. The maximum number of Hessian diagonal elements equals the number of atomic coordinates.

The force field derived initial Hessian can be printed for inspection. Type in input:

```
HESSTEST
```

ADF will construct and print the initial Hessian and then abort.

**Hessian values for selected coordinates**

The diagonal elements for selected free coordinates can be given if these free variables are named in the GEOVAR block.

```
GEOVAR
  Varname Data H=HessValue
END
```

Varname, Data

The name of the variable and any data as discussed in the sections above: assignment of initial value, final value (in case of a Linear Transit run), or a Frozen specification.

HessValue

The value for the diagonal element of the Hessian associated with that variable. All atomic coordinates that are defined by this variable will get the HessValue as diagonal element in the initial force field.

Specification of a HessValue for a frozen coordinate or a Linear Transit parameter is meaningless.
**Frequencies**

Harmonic frequencies are computed in ADF by numerical differentiation of energy gradients in slightly displaced geometries [12, 13]. The computation of frequencies is activated by specifying it as the runtime in the GEOMETRY block, see the Geometry Optimization section above.

Most of the subkeys in the GEOMETRY block are meaningless for the calculation of frequencies. Indeed, a Frequencies calculation is not a variation on optimization, but rather a sequence of Single Point runs for the equilibrium geometry and a series of slightly different geometries. By comparison of the computed gradients the force constants and hence the frequencies are computed (in the harmonic approximation of the energy surface).

```
GEOMETRY
  Frequencies {Numdif=Numdif} {Disrad=drad} {Disang=dang}
  ITERATIONS Niter
END
```

**Numdif**

Must have the value 1 or 2 and specifies the type of numerical differentiation that is applied to compute the force constants from gradients in slightly displaced geometries: 1-point or 2-point numerical differentiation. In the former case the gradients of the displaced geometry are compared with the gradients at the input (equilibrium) geometry. In the latter case both a negative and a positive displacement are applied, yielding much more accurate results but at the expense of more computations. Default: numdif=2.

**dang and drad**

The displacements of the coordinates that will be varied. **Dang** applies to angles (bond and dihedral) in degrees and **drad** applies to Cartesian (x, y, z) coordinates and to bond lengths, in angstrom. Defaults: 1 degree and 0.01 angstrom.

**Niter**

In a calculation of frequencies it is the total number of (displaced) geometries for which gradients are computed. By default this is internally determined such that the calculation of frequencies can be completed. If you reduce it, the run will only partially build the matrix of force constants and a restart is required to complete the computation.

**WARNING:** you cannot combine a Frequencies calculation with the QMMM feature.

**Accuracy**

Accuracy is a crucial aspect in the computation of frequencies, in particular for modes with low frequencies: the gradients at the geometries displaced along that mode will hardly change – analytically – from their equilibrium values, so numerical integration noise may easily affect the reliability of the computed differences in gradients. It is worthwhile to consider carefully the size of the displacements. At one hand they should be small in order to suppress the effect of higher order (anharmonic) terms in the energy surface around the minimum, at the other hand they should be large enough to get significant differences in gradients so that these are computed reliably. High precision calculations where low frequency modes are involved may require high integration settings [14]. The default (i.e. automatic) value in a FREQUENCIES run is 6.0 (!). This may not be necessary in all cases, but it
turns out to be required quite often in order to get accurate results. The calculation of frequencies by evaluating a series of displaced geometries, as it is implemented in ADF, is very time-consuming. This is even more so in view of the high (default) integration precision. This means that you should be prepared for long calculations.

Using 2-point differentiation rather than 1-point differentiation implies two-sided displacements of the atoms. This doubles the computational effort but in the so-computed force constants all anharmonic terms of odd order are eliminated. Since in general the lowest anharmonicity is third order this eliminates the first anharmonicity. Again, this is a feature directed primarily at obtaining highly accurate and reliable results.

If a Frequencies calculation is carried out only to construct a good start-up Hessian for a TS search (see the RESTART key), accurate results are not crucial. The most important thing in such a situation is to get a fair guess for the negative eigenvalue and its associated mode, and to avoid spurious additional negative eigenvalues. We recommend to avoid the rather time-consuming standard Hessian-computing preparation run for a TS search and to lower the precision of the Frequencies run. A reasonable value should be 4.0.

**Cartesian versus Z-matrix displacements**

Cartesian displacements yield usually a higher accuracy than Z-matrix displacements because in the former case cancellation of numerical integration errors between the different geometries is (almost always) larger. If Z-matrix coordinates are used as the displacement variables, then make sure that no bond angles of 180 (or zero) degrees are among them. They will very probably be treated incorrectly. If your molecule has such bond angles, use dummies to redefine the coordinates or use Cartesian displacements.

**Fragments**

**Fragment files**

The TAPE21 result files from the ADF computations on the fragments that constitute a molecule completely characterize these fragments. The fragment TAPE21 files must be attached as fragment files. This is achieved with the key FRAGMENTS. See also the next section for the relation between Atom type, Fragment type and Fragment file names.

```
FRAGMENTS {Directory}
   FragType  FragFile
   FragType  FragFile
   ...
END
```

FragType
One of the fragment types defined under ATOMS, either explicitly \((f=fragtype/n)\) or implicitly (fragment type=atom type, if the \(f=\) option is not used).
The fragment file: the standard TAPE21 result file from the computation of that fragment. The file name must contain the complete path relative to Directory (the argument of the key). By default, when no Directory is specified, this is the local directory where the job runs. You may therefore omit the directory and give simple (local) file names if all the files are present in the working directory of the job.

Obviously, FragFile is case sensitive. However, FragType is also treated as case sensitive; see also the ATOMS key discussion (F= option). The reason is that there are shortcuts possible to the effect that the FragType name (in the ATOMS block) is immediately interpreted as the name of the fragment file.

The key fragments may be used any number of times in the input file. This is convenient if you employ a sizeable number of fragment files, with subsets located in different directories. You can then use the key separately for each directory, to avoid typing long path names for all the files. Fragtypes that occur in the FRAGMENTS block(s), but that are not referred by ATOMS are ignored. No fragment files must be specified for dummy atoms (XX).

It is allowed to use one and the same fragment file for different fragment types. Example:

<table>
<thead>
<tr>
<th>ATOMS</th>
</tr>
</thead>
<tbody>
<tr>
<td>C.1 x1 y1 z1</td>
</tr>
<tr>
<td>C.2 x2 y2 z2</td>
</tr>
<tr>
<td>...</td>
</tr>
<tr>
<td>END</td>
</tr>
<tr>
<td>FRAGMENTS</td>
</tr>
<tr>
<td>C.1 TAPE21.c</td>
</tr>
<tr>
<td>C.2 TAPE21.c</td>
</tr>
<tr>
<td>...</td>
</tr>
<tr>
<td>END</td>
</tr>
</tbody>
</table>

Two different atom types (and fragment types) C.1 and C.2 are defined. The properties of the two fragment types are now identical since they are characterized by the same fragment file, but from the program's point of view they are different and can therefore not be symmetry equivalent.

The reason you may want to specify different atom types will usually be related to analysis, in particular symmetry aspects. If you know in advance that the two atom types are not symmetry equivalent, or more generally, that they play a rather different role in the molecule, it can enhance clarity of printed output to assign different atom type names to them. However, see the notes below.

A fragment file must not be the result file of a spin-unrestricted calculation. When you try to use such a fragment file, the program will detect it and abort with an error message. If you want to analyze a molecule in terms of unrestricted fragments, you should use restricted fragment files and apply the key FRAGOCCUPATIONS.

Suppose that you have done a calculation on a molecule mol, in which you have defined two different atom types for atoms of the same chemical element. Suppose furthermore, that you want to use that molecule now as a fragment in a new calculation.
You list under **ATOMS** all atoms of the molecule and you specify which atoms belong to the various fragments, among which the molecular fragment **mol**. The program will then have a problem deciding which atoms in your system are associated with the different atom types in the fragment. Normally, ADF analyzes this by comparing the chemical elements. That is not sufficient here because one chemical element corresponds with more than one type of atom in the **mol fragment** type. In such a case it is imperative to use the same atom type names in your new calculation as you used in the generation of the fragment. These names are stored in the fragment file, and they are printed in the output file of the calculation of **mol**.

The names of three items may be related to each other, depending on how you specify input: the **atom type**, the **fragment type**, and the **fragment file**.

The atom type is defined in the data block to **ATOMS**.

The fragment type is defined also in the data block to **ATOMS**: with the *f* option. For records in the data block that don't have the *f* option, the fragment type name is by definition identical to the atom type name.

The fragment file is defined in the data block to **FRAGMENTS**, each record consisting of a fragment type name, followed by the fragment file. If a fragment type is not listed in the data block to **FRAGMENTS**, so that no fragment file name is specified, the fragment file is by definition identical to the fragment type name.

**QM/MM**

ADF supports the QM/MM method to handle large systems or environment effects by treating only part of the atoms quantum-mechanically and the other ones by molecular mechanics. Use of this feature is invoked by the QM/MM keyword (block type). The functionality and all details of the keyword, involving quite a few options and aspects, are described in the separate QM/MM manual.

**Model Hamiltonian**

**Density Functional**

The Density Functional, also called the exchange-and-correlation (XC) functional, consists of an LDA and a GGA part. LDA stands for the Local Density Approximation, which implies that the XC functional in each point in space depends only on the (spin) density in that same point. GGA stands for Generalized Gradient Approximation and is an addition to the LDA part, by including terms that depend on derivatives of the density. For both terms ADF supports a large number of the formulas advocated in the literature.

In principle you may specify different functionals to be used for the **potential**, which determines the self-consistent charge density, and for the **energy** expression that is used to evaluate the (XC part of the) energy of the charge density. To be consistent, one should generally apply the same functional to evaluate the potential and energy respectively. Two reasons, however, may lead one to do otherwise:

- The evaluation of the GGA part in the potential is rather time-consuming. The effect of the GGA term in the potential on the self-consistent charge density is often not very large. From the point of view of
computational efficiency it may, therefore, be attractive to solve the SCF equations at the LDA level (i.e. not including GGA terms in the potential), and to apply the full expression, including GGA terms, to the energy evaluation \textit{a posteriori}: post-SCF.

- A particular XC functional may have only an implementation for the potential, but not for the energy (or vice versa). This is a rather special case, intended primarily for fundamental research of Density Functional Theory, rather than for run-of-the-mill production runs.

The key that controls the Density Functional is \texttt{XC}, with sub keys \texttt{LDA} and \texttt{GGA} (or equivalently: \texttt{GRADIENTS}) to define the LDA and GGA parts of the functional. Either subkey is optional (need not be used) and may occur twice in the data block: if one wants to specify different functionals for potential and energy evaluations respectively, see above.

\begin{verbatim}
  XC
    {LDA (Apply) LDA (Stoll)}
    {GGA (Apply) GGA}
  END
\end{verbatim}

\textbf{Apply}

States whether the functional defined on the pertaining line will be used self-consistently (in the SCF-potential), or only post-SCF, i.e. to evaluate the XC energy corresponding to the charge density.

The value of \texttt{apply} must be \texttt{SCF} or \texttt{Energy}.

A value \texttt{postSCF} will also be accepted and is equivalent to \texttt{Energy}.
A value \texttt{Potential} will also be accepted and is equivalent to \texttt{SCF}.

For each record separately the default (if no \texttt{Apply} value is given in that record) is \texttt{SCF}.

For each of the two terms (LDA, GGA) in the functional: if no record with \texttt{Energy} specification is found in the data block, the evaluation of the XC energy will use the same functional as is applied for the potential.

\textbf{LDA}

Defines the LDA part of the XC functional and can be any of the following:

\texttt{Xonly}: The pure-exchange electron gas formula. Technically this is identical to the \texttt{Xalpha} form (see next) with a value 2/3 for the X-alpha parameter.

\texttt{Xalpha}: the scaled (parameterized) exchange-only formula. When this option is used you may (optionally) specify the X-alpha \textit{parameter} by typing a numerical value after the string \texttt{Xalpha} (separated by a blank).

If omitted this parameter takes the default value 0.7

\texttt{VWN}: the parameterization of electron gas data given by Vosko, Wilk and Nusair (ref [15], formula version V). Among the available LDA options this is the more advanced one, including correlation effects to a fair extent.

\textbf{Stoll}

For the VWN variety of the LDA form you may include Stoll's correction [16] by typing \texttt{Stoll} on the same line, after the main LDA specification. You must not use Stoll's correction in combination with the \texttt{Xonly} or the \texttt{Xalpha} form for the Local Density functional.

\textbf{GGA}

Specifies the GGA part of the XC Functional, in earlier times often called the “non-local” correction to the LDA part of the density functional. It uses derivatives (gradients) of the charge density. Separate choices can
be made for the GGA exchange correction and the GGA correlation correction respectively. Both specifications must be typed (if at all) on the same line, after the GGA subkey.

For the exchange part the options are:

- **Becke**: the gradient correction proposed in 1988 by Becke [17].
- **PW86x**: the correction advocated in 1986 by Perdew-Wang [18].
- **PW91x**: the exchange correction proposed in 1991 by Perdew-Wang [19].

For the correlation part the options are:

- **Perdew**: the correlation term presented in 1986 by Perdew [20].
- **PW91c**: the correlation correction of Perdew-Wang (1991), see [19].
- **LYP**: the Lee-Yang-Parr 1988 correlation correction, [21-23]

Some GGA options define the exchange and correlation parts in one stroke. These are:

- **PW91**: this is equivalent to \( pw91x + pw91c \) together.
- **Blyp**: this is equivalent to Becke (exchange) + LYP (correlation).
- **LB94**: this refers to the XC functional of Van Leeuwen and Baerends [24]. There are no separate entries for the Exchange and Correlation parts respectively of LB94.

The string **GGA** must contain not more than one of the exchange options and not more than one of the correlation options. If options are applied for both they must be separated by a blank or a comma.

**Defaults and special cases**

If the XC key is not used, the program will apply only the Local Density Approximation (no GGA terms). The chosen LDA form is then **VWN**.

If only a GGA part is specified, omitting the **LDA** sub key, the LDA part defaults to VWN, except when the LYP correlation correction is used: in that case the LDA default is **Xonly**: pure exchange.

The reason for this is that the **LYP** formulas assume the pure-exchange LDA form, while for instance the Perdew-86 correlation correction is a correction to a correlated LDA form. The precise form of this correlated LDA form assumed in the Perdew-86 correlation correction is not available as an option in ADF but the VWN formulas are fairly close to it.

Be aware that typing only the sub key **LDA**, without an argument, will activate the **VWN** form (also if **LYP** is specified in the GGA part).

The LB94 functional has only a **SCF** (=Potential) implementation, but no **Energy** counterpart. Therefore, LB94 must not be used together with the **Energy** specification for **Apply**. If LB94 is used for the **Potential** (=SCF), the GGA energy expression defaults to Becke (exchange part) + Perdew (correlation). This can be overruled by selecting another choice in the “**GGA Energy...**” specification.

The LB94 form is a density functional specifically devised to get the correct asymptotic behavior. This yields much better energies for the highest occupied molecular orbital (HOMO) and better excitation energies in a calculation of response properties (Time Dependent DFT). Energies for lower lying orbitals (sub-valence) should improve as well. The energy expression underlying the LB94 functional is very inaccurate. This does not affect the response properties but it does imply that the energy and its derivatives (gradients) should not be used because LB94-optimized geometries will be wrong, see for instance [25]. The application of the LB94 functional
in a runtype that involves the computation of energy gradients is disabled in ADF. You can override this internal check with the key ALLOW.

The LB94 formalism cannot be used in a Create run (due to an implementation limitation in the code). If you need the energy difference of a molecule with respect to LB94-atoms, you have to run the single-atom calculations with LB94 separately, using the same non-LB94 Create atoms as fragments as you did for the whole molecule. This will give you the required energy corrections.

**General remarks**

- The phrase non-local in the discussion of density functionals does not mean that non-local potentials are involved. The potentials are perfectly local, but when you go beyond LDA and include gradient corrections, the value of the density functional potential in a point \( r \) is evaluated not only from the local value of the charge density, but also from the gradient of the charge density.
- The Stoll formula is considered to be a correlation correction to the Local Density Approximation. It is conceptually not correct to use the Stoll correction *and* apply non-local gradient (GGA) corrections to the correlation. It is the user's responsibility, in general and also here, to avoid using options that are not solidly justified theoretically.
- It is questionable to apply gradient corrections to the *correlation*, while not doing so at the same time for the exchange. Therefore the program will check this and stop with an error message. This check can be overruled with the key ALLOW.
- The issue of the “best” density functional is a subject of extensive and widespread research. It is generally recognized that applying gradient corrections to the simplest Local Density Approximation usually gives better results for comparison with experimental data, especially as regards bond energies and the spectra computed from one-electron energies.
- The incorporation of gradient corrections during the SCF significantly increases the computing effort. In this respect it makes no difference which specific GGA formula is applied. The Energy (PostSCF) feature is therefore an alternative worthwhile considering: it saves a lot of time and the effects of this approximation are often small as regards the SCF solution, so the non-self-consistent aspect hardly shows up in the computed bond energy. In Geometry Optimizations, however, the Post-SCF option implies that the energy gradients are computed from the LDA energy expression and hence the resulting optimized geometry corresponds to the LDA functional. In such a case, including the GGA term may make a substantial difference to the computed equilibrium geometry.

**Relativistic effects**

<table>
<thead>
<tr>
<th>RELATIVISTIC {level} {formalism} {potential}</th>
</tr>
</thead>
</table>

**Level**

May be **None** (this suppresses the key, and is equivalent to not using the key at all), **Scalar** (default: scalar relativistic effects), or **SpinOrbit** (using double group symmetry).

**Formalism**

**Pauli** (default) or **ZORA** (**ZORA is RECOMMENDED!**)
Potential

Frozen (default) or Full

The key RELATIVISTIC instructs ADF to take relativistic effects into account. By default (omission of the key) this is suppressed.

Pauli

Specification of the Pauli formalism means that the first order relativistic corrections (the Pauli Hamiltonian) will be used [26-35]. In a scalar relativistic run ADF employs the single point group symmetry and only the so-called scalar relativistic corrections, Darwin and Mass-Velocity. The treatment is not strictly first-order, but is quasi-relativistic, in the sense that the first-order scalar relativistic Pauli Hamiltonian is diagonalized in the space of the non-relativistic solutions, i.e. in the non-relativistic basis set.

The quasi-relativistic approach improves results considerably over a first-order treatment. There are, however, theoretical deficiencies due to the singular behavior of the Pauli Hamiltonian at the nucleus. This would become manifest in a complete basis set but results are reasonable with the normally employed basis sets. However, this aspect implies that it is not recommended to apply this approach with an all-electron basis set for the heavy atoms, and for very heavy elements even a frozen core basis set often fails to give acceptable results.

ZORA

The ZORA approach gives generally better results than the Pauli formalism. For all-electron calculations, and in fact also for calculations on very heavy elements (Actinides), the Pauli method is absolutely unreliable. Therefore, with its formal introduction in ADF1999, the ZORA method is the recommended approach for relativistic calculations with ADF.

ZORA refers to the Zero Order Regular Approximation [36-40]. This formalism requires special basis sets, primarily to include much steeper core-like functions; applying the ZORA method with other, not-adapted basis sets, gives unreliable results. The ZORA basis sets can be found in the ADF database, in subdirectories under the $ADFHOME/atomicdata/ZORA directory.

The ZORA formalism can also be used in Geometry Optimizations. However, there is a slight mismatch between the energy expression and the potential in the ZORA approach, which has the effect that the geometry where the gradients are zero does not exactly coincide with the point of lowest energy. The differences are small, but not negligible, order of magnitude: 0.001 angstrom.

Spin-Orbit coupling

The Spin-Orbit option uses double-group symmetry. The symmetry-adapted orbitals are labeled by the quantum number $J$ rather than $L$ and any references in input to subspecies, such as a specification of occupation numbers, must refer to the double group labels.

Create runs must not use the Spin-Orbit formalism. The SFO analysis of Molecular Orbitals has not yet been implemented for a Spin-Orbit calculation. Gradient calculations for the Spin Orbit formalism have not yet been implemented either. Therefore, computations of harmonic frequencies calculation and geometry optimizations cannot employ this feature.
In a Spin-Orbit run each spatial orbital can allocate 2 electrons (times the dimension of the irreducible representation) as in a normal restricted calculation. However, contrary to the normal case these two electrons are not directly associated with spin-\( \alpha \) and spin-\( \beta \) so using the unrestricted feature in order to assign different numbers of electrons to \( \alpha \) and \( \beta \) spin respectively cannot be applied as such.

**Relativistic core potentials**

In all relativistic calculations -- scalar as well as spin-orbit -- the relativistic atomic core densities and the relativistic atomic potentials **must** be made available to ADF on a file specified with the key **COREPOTENTIALS**. This file must contain data for all atom types in the molecule, even for those atoms where relativistic aspects are expected to be negligible or that may not have a frozen core at all (such as Hydrogen). Excepted are any Ghost atoms (for instance for a BSSE calculation): these can not have any core potentials. This is tested by the program, internally, by looking at the nuclear charge and at the number of electrons belonging to an atom: if both numbers are zero, no (relativistic or other) core potential is allowed. Relativistic potentials can be generated with the auxiliary program DIRAC, see the UTILITIES document.

**Solvent effects: COSMO**

You can study chemistry in solution, as contrasted to the gas phase, with the implementation in ADF [41] of the Conductor like Screening Model (COSMO) of solvation [42-44]. The energy derivatives can also be calculated, so geometry optimization, harmonic frequencies, etcetera are available within this model.

The COSMO model is a dielectric model in which the solute molecule is embedded in a molecule-shaped cavity surrounded by a dielectric medium with given dielectric constant \( \varepsilon \). Energy-related terms are computed for a conductor first, then scaled by the function

\[
f(\varepsilon) = \frac{\varepsilon - 1}{\varepsilon + x}
\]

The empirical scaling factor \( x \) is specified in the input data block for the solvation key. The block key **SOLVATION** turns the solvation calculation on. In most cases default values are available for the involved parameters.

```plaintext
SOLVATION
  {SURF   Esurf}
  {DIV    {Ndiv=3}  {Min=0.5}  {OFAC=0.8}}
  {NOASS}
  {SOLV}
  {Eps=78.4}{Del=1.4}{Rad=1.4}{Emp=0.0}{Cav0=1.321}{Cav1=0.0067639}
  {RADII
    name1=value1
    name2=value2
    ...
  SUBEND }
```
Presence of the Solvation key block triggers the solvent calculation and does not require additional data. With subkeys you can customize various aspects of the model, for instance to specify the type of solute. None of the subkeys is obligatory. Follows a description of the subkeys.

**Surf**

Esurf must be Wsurf, Asurf, Esurf or Klamt. Four different cavity types are available.

Wsurf triggers the Van der Waals surface (VdW), which consists of the union of all atomic spheres.

Asurf gives the Solvent-Accessible-Surface (SAS). This is similar to VdW but consists of the path traced by the center of a spherical solvent molecule rolling about the VdW surface or, equivalently, a VdW surface created by atomic spheres to which the solvent radius has been added. These two surface types contain cusps at the intersection of spheres.

Esurf (the default) gives the Solvent-Excluding-Surface (SES), which consists of the path traced by the surface of a spherical solvent molecule rolling about the VdW surface. Primarily, this consists of the VdW surface but in the regions where the spheres would intersect, the concave part of the solvent sphere replaces the cusp.

These 3 surfaces are constructed with the GEPOL93 algorithm [45].

The SES surface is the default in ADF.

The fourth surface option is Klamt as described in [42]. It excludes the cusp regions also.

The actual construction of the surface involves a few technical parameters controlled with the subkey

**DIV**

Nddiv controls how fine the spheres that in fact describe the surface are partitioned in small surface triangles, each containing one point charge to represent the polarization of the cavity surface. Default Nddiv=3

Min specifies the size, in angstrom, of the smallest sphere that may be constructed by the SES surface. For VdW and SAS surfaces it has no meaning. Default Min=0.5

Ofac is a maximum allowed overlap of new created spheres, in the construction procedure. Default Ofac=0.8

**NOASS**

By default all new spheres that are created in the surface-construction are assigned to atoms, for the purpose of gradient computations (geometry optimization). Specifying the NOASS subkey turns this off. It has no argument.

**Solv**

Details attributes of the solvent. Eps specifies the dielectric constant (the default relates to water).

Rad specifies the radius of the (rigid sphere) solvent molecules, in angstrom.

Emp addresses the empirical scaling factor $x$ in the formula above.

Other options specify a linear parameterization of non-electrostatic terms as a function of surface area:

$$E_{\text{non-elst}} = f(\varepsilon)^* (CAV0 + CAV1* \text{area}) \quad (2.1.2)$$

63
In order to construct the surface you have to specify the atomic ("Van der Waals") radii. There are three ways of doing this. In the first method you append "R=value" to the atomic coordinates record, in the ATOMS key block. This would look like, for instance

\[
\begin{array}{cccccccc}
\text{C} & 1 & 2 & 3 & \text{CC} & \text{CCO} & \text{CCOH} & f=\text{C}.dz & R=2.0
\end{array}
\]

It assigns a radius of 2.0 to the Carbon atom.

In the second method you apply the same format, but specify a symbol (identifier) rather than a value

\[
\begin{array}{cccccccc}
\text{C} & 1 & 2 & 3 & \text{CC} & \text{CCO} & \text{CCOH} & f=\text{C}.dz & R=\text{C-sp3}
\end{array}
\]

The identifiers must be defined in the (optional) RADII subkey block in the SOLVATION data block (see next).

In the third method, you don’t modify the ATOMS block at all. In this case, the RADII subkey must be used and the “identifiers” in it must be exactly the atom type names in the ATOMS block.

**Radii**

This subkey is block type. Its data block (if the subkey is used) must terminate with a record SUBEND. In the RADII data block you give a list of identifiers and values

\[
\begin{array}{cccccccc}
\text{Solvation} & & & & & \\
& & & & & & & \\
& & & & & & & \\
\text{Radii} & & & & & & & \\
& & & & & & & \\
& & & & & & & \\
& & & & & & & \\
& & & & & & & \\
& & & & & & & \\
& & & & & & & \\
& & & & & & & \\
\text{Subend} & & & & & & & \\
& & & & & & & \\
& & & & & & & \\
& & & & & & & \\
\text{End} & & & & & & & \\
\end{array}
\]

The values are the radii of the atomic spheres, in the same units of length as used in the ATOMS block (angstrom or bohr). The names specify to which atoms these values apply. As discussed for the SOLV subkey this depends on the ATOMS block. If in the specification of atomic coordinates you have used the “R=” construct to assign radii, with identifiers rather than values for the R-value, these identifiers must be defined in the RADII sub block. If no “R=” construct was applied in the ATOMS block, you must use the atom type names as they occurred in the ATOMS data block. Referring to the example given in the SOLV subkey discussion, you might have

\[
\begin{array}{cccccccc}
& & & & & & & \\
& & & & & & & \\
& & & & & & & \\
& & & & & & & \\
& & & & & & & \\
& & & & & & & \\
& & & & & & & \\
& & & & & & & \\
\text{Radii} & & & & & & & \\
& & & & & & & \\
& & & & & & & \\
& & & & & & & \\
\text{Subend} & & & & & & & \\
& & & & & & & \\
& & & & & & & \\
& & & & & & & \\
\end{array}
\]

A simple atom type reference might look like

\[
\begin{array}{cccccccc}
& & & & & & & \\
& & & & & & & \\
& & & & & & & \\
& & & & & & & \\
\text{Radii} & & & & & & & \\
& & & & & & & \\
& & & & & & & \\
& & & & & & & \\
\text{Subend} & & & & & & & \\
& & & & & & & \\
\end{array}
\]

...
This concludes the discussion of the RADIUS subkey.

CHARGED
This addresses the determination of the (point) charges that model the cavity surface polarization. In COSMO calculations you compute the surface point charges \( q \) by solving the equation \( Aq = -f \), where \( f \) is the molecular potential at the location of the surface charges \( q \) and \( A \) is the self-interaction matrix of the charges. The number of charges can be substantial and the matrix \( A \) hence very large. A direct method, i.e. inversion of \( A \), may be very cumbersome or even impossible due to memory limitations, in which case you have to resort to an iterative method.

Meth specifies the equation-solving algorithm. Meth=INVER requests direct inversion. Meth=GAUS calls for the Gauss-Seidel iterative method. Meth=Jacobi activates another standard iterative procedure. The latter two methods require a positive-definite matrix (which may fail to be the case in an actual calculation) and can be used with a relaxation technique, controlled by the relaxation parameter OMEGA (1.0=no relaxation). Meth=CONJ (default) uses the preconditioned biconjugate gradient method. This is guaranteed to converge and does not require huge amounts of memory.

CONV and ITER are the convergence criterion and the maximum number of iterations for the iterative methods.

Some of the molecular electronic charge distribution may be located outside the cavity. This affects the assumptions underlying the COSMO equations. Specifying the CORR option to the CHARGED subkey constrains the computed solvent surface charges to add up to the negative of the molecular charge.

C-MATRIX
- How: For the potential \( f \) we need the Coulomb interaction between the charges \( q \) and the molecular electronic density (and nuclei). Three methods are available, specified by the first option to the C-Matrix subkey.
  a) EXACT: compute the straightforward Coulomb potential due to the charge \( q \) in each point of the molecular numerical integration grid and integrate against the electronic charge density. This is, in principle, exact but may have inaccuracies when the numerical integration points are very close to the positions of a charge \( q \). To remedy this, the point charges \( q \) can be ‘smeared out’ and represented by a disc, see the next subkey DISC.
  b) FIT: same as EXACT, but the \( q \)-potentials are now integrated not against the exact electronic charge density, but against the (much cheaper-to-compute) fitted density. The same DISC considerations apply.
  c) POT: evaluate the molecular potential at the position of the charge \( q \) and multiply against the \( q \)-strength. Since the molecular Coulomb potential is computed from the fit density, any difference in results between the FIT and the POT approach should be attributed to the DISC issue.

POT is the default, because it is faster, and is only inadequate if the fit density is very inaccurate, which would be a problem anyway.

- SCF: If you specify this option, the computation of the Coulomb interaction matrix (between electrons and surface charges) is carried out during the SCF procedure, but this turns out to hamper the SCF convergence behavior. Therefore: not recommended. IF you use it, the program will switch to one of the other 3 methods, as given by the “How” option, as soon as the SCF convergence error drops below TOL: (applies only to the SCF option, which is not recommended).
**DISC**

Applies only when the C-matrix method is EXACT or FIT. Note, however, that the default for the C-matrix method is POT, in which case the DISC subkey has no meaning. The DISC key lets the program replace the point charges \( q \) by a solid uniformly charged spherical surface disc whenever the numerical integration accuracy requires so, i.e. for those charges that are close to numerical integration points.

Options:
- **SC** defines a shrinking factor, by which the actual disc radius used is reduced from its ‘normal’ value: an inscribed disc in the triangular surface partitions that define the distribution of surface charges, see the subkey DIV.
- **LEG** gives the polynomial expansion order of the disc potentials. The Legendre expansion converges rapidly and the default should be adequate.
- **TOL** is a tolerance parameter to control the accuracy of the disc potential evaluations.

**SCF**

In COSMO calculations you can include the surface charges in the Fock operator self-consistently, i.e. by recomputing the charges \( q \) at every SCF cycle and include them in the equations, or in a perturbational manner, i.e. post-SCF. This is controlled with the first option. The When option must be either VAR or PERT, for variational and perturbational, respectively. Default is VAR.

The second (HOW) option applies only to the WHEN=VAR case and may affect the speed of SCF convergence. The COSMO calculation implies a considerable increase in CPU time! Values for HOW:
- **ALL**: This includes it in all SCF cycles (except for the first SCF cycle, which is gas-phase)
- **LAST**: This lets the program first converge the SCF completely without any solvent effects. Thereafter, the COSMO is turned on, hopefully converging in fewer cycles now, to compensate for the “double” SCF effort.
- **TOL=0.1** (or another value) is an in-between approach: converge the gas-phase SCF until the SCF error is below TOL, then turn on COSMO.

**LPRT**

This is a debug switch and triggers a lot more output related to the cavity construction etc.

**Electric Field: Homogeneous and Point Charges**

A homogeneous external electric field and/or the field due to point charges can be included in the Fock operator. Either can be applied only in a Single-Point calculation (or a Create run) because the energy derivatives that are computed in Geometry Optimizations do not take the fields into account.

```
EFIELD {ex ey ez}
  {x y z  q
   x y z  q
   ...
  }
END }
```

**EFIELD**

This *general* key can be used as a simple key or as a block key. The block form applies if no argument (ex, ey, ez) is given or when the argument is followed by the *continuation symbol* ( &).
Define a homogeneous electric field in atomic units: atomic Volts per bohr; the relation to SI units is: 1 a.u. = $5.14 \times 10^{11}$ V/m.

The units applied by adf for the interpretation of homogeneous field values are not affected by any units used for specifying atomic coordinates. By default no homogeneous E-field is included.

The Cartesian coordinates and strength of a point charge (in elementary charge units, +1 for a proton). Each point charge must be specified on a separate line in the data block. The Cartesian coordinates are in the units of length that was set by units (for interpreting atomic coordinates input). By default no point charges are included.

**Orientation of the fields**

When the atomic coordinates are input in Z-matrix format, the direction of the homogeneous field and the location of the point charges as specified in input are interpreted as referring to the standard Cartesian frame associated with Z-matrix input. The standard frame means: the first atom at the origin, the second on the positive x-axis, the third in the xy-plane with positive y.

If the program rotates (and translates, as the case might be) the atoms from the input frame – or the auto-generated frame in case of Z-matrix input – to some other frame, for instance to accommodate the internal ADF symmetry orientation requirements, the fields are transformed along with the atoms.

**Symmetry**

The homogeneous electric field and the point charge fields may polarize the electronic charge density. This must be accounted for in the point group symmetry. If symmetry is not specified in input, the program computes the symmetry from the nuclear frame and the fields.

**Bonding energy**

The bonding energy is computed as: the energy of the molecule in the field minus the energy of the constituent fragments in the same field. Of course, the fragments may not be polarized and hence not be self-consistent in this field. This depends on how the fragments themselves were computed.

**Polarizability and hyperpolarizability**

ADF supports a direct calculation of the (hyper) polarizability (see next section). The static (hyper) polarizabilities can also be computed by applying a small homogeneous field and comparing the results with the field-free data.

**Time-dependent DFT: Excitation Energies, (Hyper) Polarizabilities**

Excitation energies, frequency-dependent (hyper) polarizabilities, Van der Waals dispersion coefficients, higher multipole polarizabilities, Raman scattering intensities and depolarization ratios of closed-shell molecules are all available in ADF [46, 47] as applications of time-dependent DFT (TDDFT); see [48] for a review.
The input description for these properties is split in three parts: (a) general advice and remarks, (b) excitation energies, and (c) frequency-dependent (hyper) polarizabilities and related properties.

**General remarks on the use of the TDDFT Response and Excitation functionality**

**Symmetry**
As in calculations without TDDFT the symmetry is automatically detected from the input atomic coordinates and need not be specified, except in the following case: infinite symmetries cannot be handled in the current release (ATOM, C(lin), D(lin)). For such symmetries a subgroup with finite symmetry must be specified in the input. The usual orientation requirements apply.

If higher multipole polarizabilities are required, it may also be necessary to use a lower subgroup (the program will stop with an error message otherwise). For verification of results one can always compare to a NOSYM calculation.

**Closed-shell only**
The current implementation supports only closed-shell molecules. If occupation numbers other than 0 or 2 are used the program will detect this, (but only at a later stage of the calculation) and abort. All “RESPONSE” calculations must be spin-restricted.

**Atomic coordinates in a RAMAN calculation**
Atomic coordinate displacements in a RAMAN calculation must be Cartesian, not Z-matrix. Furthermore, the current implementation does not yet support constrained displacements, i.e. you must use all atomic coordinate displacements.

**Use of diffuse functions**
The properties described here may require diffuse functions to be added to the basis (and fit) sets. Poor results will be obtained if the user is unaware of this. As a general rule, diffuse functions are more important for smaller than for larger molecules, more important for hyperpolarizabilities than for normal polarizabilities, more important for high-lying excitation energies (Rydberg states) than for low-lying excitations, more important for higher multipole polarizabilities than for dipole polarizabilities. The user should know when diffuse functions are required and when they are not: the program will not check anything in this respect. For example, in a study on low-lying excitation energies of a large molecule, diffuse functions will usually have little effect, whereas a hyperpolarizability calculation on a small molecule is pointless unless diffuse functions are included. Diffuse basis sets are included in the database (in the Vdiff/ directory of the database), but only for a few atoms. For some other atoms diffuse basis sets may be available at the web site http://tc.chem.vu.nl/~veisberg. For other atoms, the user will have to add diffuse basis and fit functions to the existing data base sets. It is not necessary to start from basis V as was done for the basis sets just mentioned.

All this means, of course, that usage of the TDDFT functionality is still somewhat experimental. It may be expected that more extensive basis sets will come available in the future, when usage and experience increase.

**Linear dependency in basis**
If large diffuse basis sets are used, or if diffuse functions are used for atoms that are not far apart the calculation may suffer from numerical problems because of (near-) linear dependencies in the basis set. The user should be aware of this danger and use the DEPENDENCY key to check and solve this.

**The TAILS input keyword**
For reasons of numerical robustness and safety rather strict defaults apply for the neglect of tails of basis and fit functions (see the key TAILS) in a Response or Excitation calculation. This may result in longer CPU
times than needed for non-TDDFT runs, in particular for larger molecules. Possibly this precaution is not necessary, but we have not yet tested this sufficiently to relax the tightened defaults.

**Relativistic effects**

The Response and Excitations options can be combined with scalar relativistic options (ZORA or Pauli). The one-electron relativistic orbitals and orbital energies are then used as input for the property calculation. Spin orbit effects have not yet been incorporated in this part of the code. In case of a ZORA calculation, the so-called “scaled” orbital energies are used as default.

**Choice of XC potential**

For properties that depend strongly on the outer region of the molecule (high-lying excitation energies, (hyper) polarizabilities), it may be important to use a XC potential with correct asymptotic behavior (approaching \(-1/r\) as \(r\) tends to infinity). In ADF the LB94 potential has been implemented for this purpose [24].

With this particular XC functional, the XC potential is computed from the exact charge density for reasons of stability and robustness (whereas for other functions the (cheaper) fit density is used). This implies that computation times may be longer. Another ‘side effect’ is that, since there is no energy expression corresponding to the LB94 potential, the final (bonding) energy of a LB94 calculation uses another GGA and hence the energy result is not (exactly) consistent with the SCF procedure. Note, finally, that the LB94 potential is not suitable for geometry optimizations because it is rather inaccurate in the bonding region, see the discussion of the XC input key.

Applications with the LB94 potential to response calculations with ADF can be found in [49] (polarizabilities), [50-52] (hyperpolarizabilities), [53] (high-lying excitation energies), [54] (multipole polarizabilities and dispersion coefficients)

**Accuracy check list**

As mentioned before, the TDDFT module is relatively new and not extensively tested for a wide range of applications. Therefore, we strongly recommend the user to build experience about aspects that may affect the accuracy of TDDFT results. In particular we advise to ‘experiment’ with

- Varying integration accuracy
- Varying the SCF convergence
- Varying the ORTHONORMALITY and TOLERANCE values in an EXCITATION calculation
- Varying the TAILS parameters
- Using diffuse functions
- Using the DEPENDENCY key
- Applying the ZORA relativistic corrections for molecules containing heavy nuclei
- Using an asymptotically correct XC potential such as LB94

**Excitation Input**

You can perform a calculation of singlet-singlet and singlet-triplet excitation energies of a closed-shell molecule by supplying in the input file the block key **EXCITATION**

```
EXCITATION
    { OPTION }
    { OPTION }
    ...
```
Several options can be addressed with subkeys in the data block. This functionality is based on TDDFT and consequently has a different theoretical foundation than the SCF techniques described elsewhere in this User’s Guide. Two possible ways are available to solve the eigenvalue equation from which the excitation energies and oscillator strengths are obtained, of which the iterative Davidson procedure is the default. In this case, the program needs to know how many excitation energies are needed per irrep, what accuracy is required, and what type of excitation energies are required (singlet-singlet or singlet-triplet). Suitable defaults have been defined for all of these. Each of these points is discussed below.

**Exact diagonalization vs. iterative Davidson procedure**

The most straightforward procedure is a direct diagonalization of the matrix from which the excitation energies and oscillator strengths are obtained. Since the matrix may become very large, this option is possible only for very small molecules. It can be activated by specifying the word `EXACT` as one of the subkeys in the `EXCITATIONS` data block. The default is the iterative Davidson method. A few of the lowest excitation energies and oscillator strengths are then found within an error tolerance. An advantage of the `EXACT` option is that additional information is produced, such as the Cauchy coefficients that determine the average dipole polarizability.

**Singlet versus triplet**

By default, the singlet-singlet and singlet-triplet excitation energies are both calculated. The singlets are handled first, then the corresponding triplet excitation energies. One can skip one of these two parts of the calculation by specifying either `ONLYSING` or `ONLYTRIP` as a subkey in the data block.

**Which excitation energies and how many?**

The user can specify how many excitation energies per irrep should be calculated. If no pertaining input is available the program determines these numbers from the smallest differences between occupied and virtual Kohn-Sham orbital energies. By default it looks at the 10 lowest orbital energy differences. This number can be modified, by specifying inside the `EXCITATION` block key, for example:

```plaintext
LOWEST 30
```

One should be aware that this procedure does not guarantee that the lowest 10 (or 30) excitation energies will actually be found, since the orbital energy difference approximation to the excitation energy is rather crude. However, if the program decides on the basis of this procedure to calculate 4 excitation energies in a certain irreducible representation, these 4 excitation energies are certainly the lowest in that particular irrep.

The user has more control when the number of excitations per irrep is explicitly specified within the `EXCITATION` block key by the `DAVIDSON` subkey:

```plaintext
DAVIDSON &
  E'' 5
  T1.U 2
SUBEND
```
The Davidson subkey is a general (simple or block type) subkey. For usage as block type it must be followed by the continuation code ( &). Its data block may contain any number of records and must end with a record SUBEND. In the subkey data block a list of irreps, followed by the number of requested excitation energies is specified. Note that the irrep name may not be identical to the usual ADF name. For example E'' is called EEE in ADF. The Excitation code will skip an irrep of the label is not recognized. For multidimensional irreps, only the first column is treated, because the other would produce identical output. This implies that the oscillator strengths for E-irreps have to be multiplied by 2 and the oscillator strengths for T-irreps by 3.

The Exact subkey, mentioned already above, can also be used as a block type subkey to treat only a few irreps instead of all. The number of excitation energies does not have to be specified then.

Accuracy and other technical parameters

A summary of technical parameters with their defaults is:

<table>
<thead>
<tr>
<th>EXCITATION</th>
</tr>
</thead>
<tbody>
<tr>
<td>VECTORS</td>
</tr>
<tr>
<td>TOLERANCE</td>
</tr>
<tr>
<td>ORTHONORMALITY</td>
</tr>
<tr>
<td>ITERATIONS</td>
</tr>
<tr>
<td>END</td>
</tr>
</tbody>
</table>

VECTORS: the maximum number of trial vectors in the Davidson algorithm for which space is allocated. If this number is small less memory will be needed, but the trial vector space is smaller and has to be collapsed more often, at the expense of CPU time. The default if usually adequate.

TOLERANCE: specifies the error tolerance in the square of the excitation energies in hartree units. The default is probably acceptable but we recommend that you verify the results against a stricter default (e.g. 1e-8) for at least a few cases.

ORTHONORMALITY: the Davidson algorithm orthonormalizes its trial vectors. Increasing the default orthonormality criterion increases the CPU time somewhat, but is another useful check on the reliability of the results.

ITERATIONS: the maximum number of attempts within which the Davidson algorithm has to converge. The default appears to be adequate in most cases.

Applications of the Excitation feature in ADF

It may be useful to consult the following (early) applications of the Excitation feature in ADF:

1) For excitation energies based on exact XC potentials: [55]
2) Calculations on Free Base Porphin: [56]
3) Calculations on MnO₄⁻, Ni(CO)₄ and Mn₂(CO)₁₀⁻: [57]
4) Calculations on M(CO)₅ (M=Cr, Mo, W), using the ZORA relativistic approach: [58]
**Input description for the Response functionality**

The calculation of frequency-dependent (hyper)polarizabilities and related properties is activated with the block key **RESPONSE**

```
RESPONSE
END
```

In this example only the $zz$ component of the dipole polarizability tensor is calculated, at zero frequency. The orientation of the molecule is therefore crucial. Be aware that the program may modify the orientation of the molecule if the input coordinates do not agree with Asymmetry conventions in ADF!

(This calculation could equivalently be done through a finite field method).

The impact of various approximations on the quality of computed polarizabilities has been studied in, for instance, Refs. [49, 55, 59]. If you are new to this application field, we strongly recommend that you study a few general references first, in particular when you consider hyperpolarizability calculations. These have many pitfalls, technically (which basis sets to use, application of the DEPENDENCY key) and theoretically (how do theoretical tensor components relate to experimental quantities, different conventions used). Please, take a good look both at ADF-specific references [50-52, 60] and at general references related to this subject: Refs. [61-63], the entire issues of Chem.Rev.94, the ACS Symposium Series #628, and further references in the ADF-specific references.

Let’s have a look at the available subkeys in the **RESPONSE** data block. (Not all of them should be used at the same time!)

```
RESPONSE
    ALLCOMPONENTS
    HYPERPOL LaserFreq
    DYNAHYP
    NFREQ Nfreq
    FREQBEG FirstFreq {eV / Hartree / Angstrom}
    FREQEND LastFreq {eV / Hartree / Angstrom}
    ALLTENSOR
    DYNAHYP
    QUADRUPOLE
    OCTUPIOLE
    VANDERWAALS NvanDerWaals
    RAMAN
END
```

**Entire tensor or only one component**

You specify the **ALLCOMPONENTS** subkey to get the entire polarizability tensor, instead of just the $zz$ component.
Frequencies or wavelengths

Instead of performing the calculation at zero frequency (which results in the static polarizability), one can specify an even-spaced sequence of frequencies, using the subkeys \texttt{NFREQ}, \texttt{FRQBEGIN}, AND \texttt{FRQEND} with obvious meaning. The (first and last) frequency values are by default in eV. This can be changed into Hartree units (a.u.) or in wavelengths (angstroms) by typing the corresponding word after the frequency value.

Hypermultipolarizabilities

The first hyperpolarizability tensor $\beta$ is calculated if the subkey \texttt{HYPERPOL} is present with a specification of the laser frequency (in hartree units). If also the subkey \texttt{ALLCOMPONENTS} is specified, all components of the hyperpolarizability tensor will be obtained.

As mentioned before, by default only the static dipole hyperpolarizability tensor is computed. If one is interested in the frequency-dependent hyperpolarizability, the input could look like:

\begin{verbatim}
RESPONSE
   ALLCOMPONENTS
   HYPERPOL 0.01
   DYNAHYP
END
\end{verbatim}

The subkey \texttt{DYNAHYP} has to be added and the main frequency $\omega$ has to be specified in Hartrees after the subkey \texttt{HYPERPOL}. In the output all nonzero components of the tensors governing the static first hyperpolarizability, second harmonic generation, electro-optic pockels effect, and optical rectification are printed.

Note: Second hyperpolarizabilities are currently not available analytically. Some can however be obtained by calculating the first hyperpolarizability in a finite field.

The effect of using different DFT functionals (LDA, LB94, BLYP) on hyperpolarizabilities in small molecules has been investigated in [52].

Higher multipole polarizabilities

Instead of just calculating the dipole-dipole polarizability, one may address the dipole-quadrupole, quadrupole-quadrupole, dipole-octupole, quadrupole-octupole, and octupole-octupole polarizability tensors. These can all be calculated in a single run, using the subkey \texttt{ALLTENSOR}. If only quadrupole-quadrupole or octupole-octupole tensors are needed, the subkey \texttt{QUADRUPOLE} or \texttt{OCTUPOLE} should be used.

Accuracy and convergence

\begin{verbatim}
RESPONSE
   ERRALF 1e-6
   ERABSX 1e-6
   ERRTMX 1e-6
END
\end{verbatim}

The subkeys \texttt{ERRALF}, \texttt{ERABSX}, \texttt{ERRTMX} determine the convergence criteria for a polarizability calculation. The strict defaults are shown. It is rarely necessary to change the defaults, as these are rather strict but do not lead to many iterations.
Dispersion coefficients

Simple dispersion coefficients (the dipole-dipole interaction between two identical molecules, the $C_6$ coefficient) are calculated in a single ADF calculation. General dispersion coefficients are obtained with the auxiliary program DISPER, which uses two output files (file named TENSOR) of two separate ADF runs as input. See the Utilities and the Examples documents.

To get the dispersion coefficients one has to calculate polarizabilities at imaginary frequencies between 0 and infinity. The ADF program chooses the frequencies itself. The user has to specify the number of frequencies, which in a sense defines the level of accuracy, as an argument to the subkey `VANDERWAALS`.

```plaintext
RESPONSE
   ALLCOMPONENTS
   VANDERWAALS 10
END
```

Ten frequencies is reasonable. In the example only dipole-dipole interactions are considered. If `ALLTENSOR` is specified, higher dispersion coefficients are also calculated. This ADF calculation generates a file with name TENSOR, which contains the results of multipole polarizabilities at imaginary frequencies. This TENSOR file has to be saved. Similarly, the TENSOR file for the second monomer has to be saved. The files have to be renamed to files ‘tensorA’ and ‘tensorB’ (case sensitive) respectively. Then the program DISPER has to be called in the same directory where the ‘tensorA’ and ‘tensorB’ files are located. DISPER needs no further input. See the Utilities document.

Raman scattering

Raman scattering intensities and depolarization ratios for all molecular vibrations at a certain laser frequency can be calculated in a single run. The run type must be Frequencies, which is arranged with the `GEOMETRY` key.

```plaintext
GEOMETRY
   FREQUENCIES
END
```

The Response key is used to specify that Raman intensities are computed.

```plaintext
RESPONSE
   RAMAN
END
```

In this example the static Raman scattering is calculated ($\omega = 0$). This type of calculation is very similar to an IR intensity calculation. In fact, all IR output is automatically generated as well. At all distorted geometries the dipole polarizability tensor is calculated. This is very time-consuming and is only feasible for small molecules.

There are a few caveats:
- Numerical integration accuracy must be high
A calculation in which only a subset of the atoms is displaced is not possible for Raman calculations.

For good results, a well converged (with the same basis and functional) equilibrium geometry must be used. Because of this last point, it is wise to always start the RAMAN calculation with a TAPE13 restart file from a previous geometry optimization with the same basis, accuracy parameters, and density functional.

**ESR**

Electron Spin Resonance properties are accessible with the keywords ESR and QTENS.

ESR is a block-type keyword that invokes calculation of the g-tensor [64] as well as the Nuclear Magnetic Dipole Hyperfine interaction (A-tensor) [65].

QTENS gives you the Nuclear Electric Quadrupole Hyperfine interaction (Q-tensor) [66]. The latter is directly related to the Electric Field Gradient: the field gradient tensor elements equal the Q-tensor elements divided by the nuclear quadrupole moment, which is calculated and printed next to the Q-tensor.

**ESR**

**End**

ESR is a block-type key, although it has not (yet) any data records to specify options or parameters. You can use the key in two situations:

1. In a Spin-Orbit relativistic calculation the program will compute the G-tensor. The calculation must be spin-restricted and there must be exactly one unpaired electron (Kramer’s Pair).
2. If the Spin-Orbit feature is turned off, the calculation must be spin-unrestricted and apply to an open-shell system. The program will compute the Nuclear Magnetic Dipole Hyperfine interaction (A-tensor).

In case (1), the program will also calculate and print the Nuclear Magnetic Dipole Hyperfine interaction, but the terms due to the spin-polarization density at the nucleus are absent. Furthermore, if there is more than one unpaired electron, the computed results will simply be incorrect, without any warning from the program.

**QTENS**

This key activates the computation of the Nuclear Electric Quadrupole Hyperfine interaction. It can be applied to open-shell and to closed-shell systems.

For the computation of the A-tensor, the Nuclear Magnetic Dipole Hyperfine interaction, an accurate evaluation of the spin-polarization density at the nucleus is important. This is best achieved in an all-electron calculation, avoiding any frozen core approximation.
Electronic Configuration

The next few keys can be used to specify the electronic configuration. If you don’t specify any such keys, certain defaults will apply. In principle, the program will (by default) attempt to find the lowest-energy spin-restricted (one-determinant) state. If SCF convergence is problematic the program may wind up at an excited state, by which (in this context) we mean a one-determinant state with a higher energy than some other one-determinant state with the same net spin polarization. In worse cases the program may fail to converge to any state at all. It is good practice to always verify which configuration you actually have computed.

When you specify a particular configuration and/or net charge and/or net spin-polarization of the system, the program will try to compute accordingly, even if the data have no physical or chemical meaning. The program has no knowledge about the existence of materials and will simply try to carry out what you tell it to do.

Spin: restricted vs. unrestricted

UNRESTRICTED

Specifies that spin-α and spin-β MOs may be spatially different and may have different occupation numbers. The default (absence of the key) is spin-restricted. The key has no argument.

The unrestricted mode roughly doubles the computational effort. The actual numbers of spin-α and spin-β electrons respectively are controlled by the keys CHARGE and OCCUPATIONS. Note carefully, that using only the keyword UNRESTRICTED, without either CHARGE or OCCUPATIONS (or both) would not result in any spin polarization. This implies that you would effectively perform a spin-restricted calculation, but with increased computational effort. Therefore, the program will check that in an unrestricted calculation at least one of the keys CHARGE and OCCUPATIONS is applied.

The unrestricted feature is equivalent with, in ab-initio terminology, (Spin-)Unrestricted-Hartree-Fock (UHF); the N-particle wavefunction is a single determinant and not necessarily an eigenfunction of the spin operator S^2. A restricted calculation implies that the (spatial) orbitals and the occupation numbers are identical for spin-α and spin-β.

The Fock operator, both in an unrestricted and in a restricted run, commutes with the spin operator S_z, but not (unless accidentally) with S^2. The obtained one-determinant wave function may for instance be a mixture of a singlet and a triplet state.

The expectation value of S^2 is not computed in ADF.

Note: implementation of an evaluation of S^2 is not quite trivial. DFT is essentially a one-particle formalism, so the S-operator for the N-particle system has to be written out in single-particle operators [67].

Molecules that have been calculated using the unrestricted formalism cannot be employed as fragments. ADF will abort when you attach the TAPE21 result file from an unrestricted calculation as a fragment file.

A fair approximation to a computation with unrestricted fragments can be achieved with the key FRAGOCCUPATIONS. See also the Examples.
Net Charge and Spin polarization

The net charge of the molecule and the net spin polarization can be controlled with the key `CHARGE`.

\[ \text{CHARGE} \{ \text{NetQ} \ (\alpha \beta) \} \]

NetQ
- The net total charge of the molecule
- The net total spin polarization: the number of spin-\(\alpha\) electrons in excess of spin-\(\beta\) electrons. Specification is only meaningful in a spin-unrestricted calculation.

If the key is used, the first value in the argument is assigned to \text{netQ}, the net total charge, and the second to \(\alpha \beta\). If the key is not used at all, default values apply. The default for the net total charge is the \text{sum of fragment charges: not necessarily neutral!!} The fragment charges are the net total charges that were used in the fragment runs; this information is stored in the fragment files.

The default spin polarization is zero.

An unrestricted calculation with \(\alpha \beta=0\) (for instance by not specifying \text{CHARGE} at all) is in fact a restricted run: it should give exactly the same as the restricted calculation, but it will use more CPU time.

Orbital occupations: electronic configuration, excited states

With the key `OCCUPATIONS` you can specify in detail the assignment of electrons to MOs

\[ \text{OCCUPATIONS Options} \{ \text{irrep orbitalnumbers} \text{irrep orbitalnumbers} \ldots \text{END} \} \]

\text{OCCUPATIONS} is a general key: it has an argument or a data block. If you want to use both, the continuation code ( &) must be appended at the end of the argument.

Options
- May contain one or more of the following:
  - \text{Keeporbitals=NKeep}
    - Until SCF cycle \text{Nkeep} electrons are assigned to MOs according to the Aufbau principle, using at each cycle the then current orbital energies of the MOs. Thereafter the \text{KeepOrbitals} feature is applied. As soon as this is activated the program will on successive SCF cycles assign electrons to the MOs that maximally resemble – in spatial form – those that were occupied in a “reference cycle number”. The default for \text{Nkeep} is 20, except:
    a) When orbital occupations for MOs are specified explicitly in the data block of the \text{OCCUPATIONS} key,
these apply throughout.
b) In a Create run fixed occupations are derived from a database in the program.
c) When electron smearing is explicitly turned on by the user (see the `smearq` option below) $N_{\text{keep}}$ is by default 1,000,000 so the program will ‘never’ compare the spatial forms of MOs to determine the occupation numbers.

The “reference cycle number” is by default the previous cycle, which will suppress jumps in the spatial occupations during the SCF development while at the other hand allowing the system to let the more-or-less-frozen configuration relax to self-consistency.

**Freeze**

Occurrence of this word in the option list specifies that the “reference cycle number” will be the cycle number on which the KeepOrbitals feature is activated: during all subsequent SCF cycles the program will assign electrons to MOs that resemble the MOs of that specific SCF cycle. This may be used when the MOs of that cycle are already reasonably close to the final ones, and it will suppress unwanted step-by-step charge-transfers from occupied to empty orbitals that are very close in energy. By default this option is not active.

**Smearq=Smear**

Smear is half the energy width (in hartrees) over which electrons are smeared out over orbitals that lie around the fermi level and that are close in energy. Smearing is a trick that may help when the SCF has problems converging. One should be well aware that the physical meaning of a result obtained with smeared occupations is unclear (to express it mildly). It may be useful to get over a hurdle in a geometry optimization.

By default the initial smear parameter is zero (i.e.: smearing is not applied). It is turned on automatically by the program when SCF convergence is found to be problematic, but only in an optimization-type application (simple optimization, linear transit, transition state) when the geometry is not yet converged.

You can rigorously prohibit any smearing by specifying it explicitly with value zero. More generally: specifying the smear parameter makes the program to apply it always, but always with the input-specified value.

**irrep**

The name of one of the irreducible representations (not a subspecies) of the point group of the system. See the Appendix for the irrep names as they are used in ADF.

**orbitalnumbers**

A series of one or more numbers: the occupation numbers for the one-electron valence orbitals in that irrep. The orbitals are ordered according to their energy eigenvalue; higher states than those listed get an occupation number zero.

For degenerate representations such as the 2-dimensional E-representations or the 3-dimensional T-representations, you must give the total occupation, i.e. the sum over the partner representations; ADF assigns each partner an occupation equal to the appropriate fraction of what appears here.

In an unrestricted calculation, two sequences of numbers must be specified for each irrep; the sequences are separated by a double slash (`//`). The first set of numbers is assigned to the spin-$\alpha$ orbitals, the second set to the spin-$\beta$ orbitals.

Notes about the **OCCUPATIONS** data block:

- If the block form of **OCCUPATIONS** is used all valence electrons that you want to use in the calculation must explicitly be assigned to MOs: any MOs that are not mentioned in the data block will be empty (except the frozen core orbitals).
  In this context the concept *valence electrons* and hence *valence orbitals* is not necessarily identical to what
you may normally assume to be the valence space of an atom or molecule. The meaning of \textit{valence} is here strictly defined as whatever electrons are outside the frozen core. It depends therefore on the level of frozen core approximation applied in the calculation. This traces back to the Create runs in which the basic atoms were generated that are now used to build the molecule.

- When for some \textit{irrep} there is a rather long list of occupation numbers, corresponding to \textit{consecutive} \textit{fully occupied} states, you can combine these numbers and enter their sum instead: ADF knows the maximum occupation for an \textit{irrep}, and when you put a larger number the program will split it up. For instance, if you give for the \textit{p}-representation (in a single atom calculation):

\begin{verbatim}
P 17 3
\end{verbatim}

ADF will interpret this as

\begin{verbatim}
P 6 6 5 3
\end{verbatim}

i.e. the occupation number 17 is interpreted as denoting two fully occupied \textit{p}-shells and the remaining five electrons in the next higher shell.

This example also illustrates how to specify an excited state: here we have defined a hole in the third \textit{p}-shell.

- Fractional occupation numbers in input are allowed. For a discussion of the interpretation of fractional occupation numbers see [68].

The program even allows you (technically) to use a non-integer total number of electrons, whatever the physical meaning of such a calculation is.

- The data block of \textit{OCCUPATIONS} is not parsed (see the section Interpretation of Input below). The program does not replace expressions by their value and it does not recognize constants or functions defined with the \textit{DEFINE} key.

- In a Frequencies run the symmetry used internally in the program is NOSYM, irrespective of any Schönfliess symbol in the input file. As a consequence the program will recognize only the A representation (the only \textit{irrep} in NOSYM), but not the representations belonging to the input point group symmetry. (The symmetry in the equilibrium geometry, defined by the input Schönfliess symbol, is used to enhance efficiency and stability in the construction of the matrix of Force constants).

Notes about the \textit{OCCUPATIONS} options:

- If occupation numbers are explicitly defined (the block form is used), the \textit{Smearq} option cannot be used.

- The aufbau principle does not determine or adjust the distribution of electrons over spin-\textit{α} versus spin-\textit{β} in an unrestricted calculation. This aspect is controlled by the key \textit{CHARGE} and by any explicit occupations in the data block of \textit{OCCUPATIONS}.

- When occupation numbers are not specified and no Smearing is specified either, the program will turn on smearing automatically when the SCF has serious convergence problems, in an attempt to overcome those problems, but only in a geometry optimization (including transition state, linear transit, etc.). If such happens the program restores the original situation (no smearing) at the start of each new SCF. In automatic smearing the smear parameter is initiated at 0.01 hartree and may be varied (by the program) between 0.001 and 0.1 hartree.

The automatic use of smearing by the program can be prohibited by explicitly setting the smear option with value zero (\textit{Smearq}=0).

- Smearing cannot be used in combination with the \textit{KEEPORBITALS} option. This option therefore also turns of \textit{automatic} smearing in troublesome SCF’s during an optimization.
**CHARGE vs. OCCUPATIONS**

The contents of the data block of OCCUPATIONS, if used, defines the total number of valence electrons and hence the net total charge. In an unrestricted run it also defines the net spin polarization. If the key CHARGE is also used, the program will check that both specifications are consistent.

We strongly recommend to employ this and always specify the net total charge and spin polarization with CHARGE whenever explicit occupation numbers are supplied with OCCUPATIONS, to that the program will check that your occupation numbers result in the total charge and spin polarization that you have in mind.

**Create mode**

In Create mode occupation numbers are predefined (see Appendix 2 Elements of the Periodic Table), and these are applied unless you specify occupations in input yourself. Conceivably this may result in a non-aufbau configuration. In Create mode the program always operates as if OCCUPATIONS were set in input.

**Frozen core vs. pseudopotentials**

Pseudopotentials are not supported. The frozen core approximation is automatic in a normal (Fragment mode) calculation and is defined by the basic atomic fragments. The data file used in the Create run specifies the frozen core for the atom, which is then used in all molecules that incorporate that atomic fragment.

**Multiplet States**

Calculations with ADF yield results for one-determinant electronic states, which are not always the ‘true’ states of the molecule. The evaluation of the correct multiplet energies is not trivial in this approach; see the Theory document. The point is to evaluate a specific multiplet state as a linear combination of selected one-determinant functions, each computed in the field of the so-called Average-of-Configuration (AOC). Typically, in an open shell system, the AOC is the spin-restricted system in which all orbitals in the open shell are degenerate and equally occupied. The AOC serves then as a fragment for the subsequent calculations, in which the different open shell orbitals are occupied differently by specifying the appropriate occupation numbers as explained below.

**Important:** in these follow-up calculations it is imperative that the results are obtained in the AOC field: no SCF convergence must be carried out, because we only want to assign the electrons differently, while keeping exactly the AOC orbitals. To achieve this, the follow-up calculations must use the keyword SCF, and the subkey ITERATIONS must be set to 0.

Since ADF requires that the point-group symmetry matches not only to the nuclear frame but also to the electronic charge density and MO occupations, these calculations must run in a lower pointgroup symmetry. Often you will also want to run the modified calculations spin-unrestricted. For an example, see the set of sample runs that come with the package and the discussion in the Examples document.
The calculation of the one-determinant states based on the AOC reference state is controlled with the key
SLATERDETERMINANTS:. It is a general key; it can be used as a simple key and requires an argument then. It can also be used as a block key. For this particular key it is not correct to specify an argument and a data block.

### SLATERDETERMINANTS file

When used as a simple key, the argument must be a file (including the path). The file must be an ASCII file containing data in the same format as you would supply in the data block when using the key as block type key, see below. All information on the file until the *eof* must be suitable for the data block, but no record 'END' on the file must be specified: only the contents of the data block.

The block format:

```
SLATERDETERMINANTS

    TITLE1
         irrep   occups
         irrep   occups
         ....
     SUBEND

    TITLE2
         irrep   occups
         ....
     SUBEND

    TITLE3
         ....
     SUBEND

    ....

END
```

Each "TITLE" functions as a subkey, but is otherwise an arbitrary string to label the resulting one-determinant calculation. Each such subkey block contains the occupation numbers for a single one-determinant calculation. It is necessary that the calculation uses the reference AOC run as its only fragment file. The occupations in the subkey blocks must be re-arrangements of the AOC open-shell electrons. In the SLATERDETERMINANTS calculation you must explicitly specify the pointgroup symmetry in which you want to run; this must be a lower symmetry than the AOC one, otherwise you couldn’t rearrange the open shell electrons. See the Theory document. An sample run is included in Examples document.

Each “irrep   occups” record specifies the occupations for the indicated irrep in the usual way (see for instance the OCCUPATIONS key). The irrep labels must correspond to the (lower) point group symmetry used in the SLATERDETERMINANTS calculation. Note that in an unrestricted calculations, occupations numbers must be given for both spins, using the double slash (//) to separate the occupations for spin-alpha and spin-beta.

In this setup, the program will for each of the subkey blocks under the SLATERDETERMINANTS key execute an SCF calculation with only one cycle, i.e. no convergence, where the start-up field is the fragment field, i.e. the
AOC field. So all one-determinant states in this calculation are evaluated in the AOC field. The resulting energies for the distinctly computed one-determinant states can then be combined to the desired multiplet values, corresponding to how the multiplet states are combinations of the one-determinant states.

**Precision and Self-Consistency**

The precision of a calculation is determined by
a) The function sets (basis sets, levels of frozen core approximation, and fit sets for the computation of the Coulomb potential)
b) Numerical integration settings
c) Convergence criteria (for the SCF procedure and the geometry optimization)
d) A few more items that are rather technical and usually irrelevant (these are not discussed here).
The fragments you attach determine, through the fragment files, the function sets. Since each fragment traces back to one or more Create runs, the employed data base files in the Create runs determine the finally employed function sets.
For convergence of the geometry optimization see the key GEOMETRY.
In this part we examine numerical integration and the SCF procedure.

**Numerical Integration**

Many integrals in ADF are evaluated by numerical integration: Fock matrix elements, several terms in the (bonding) energy, gradients in geometry optimization, and so on. A sophisticated numerical integration procedure is used [69, 70]. It requires only one input parameter which determines the precision of numerical integrals and derives from that the number of integration points.

| INTEGRATION accint |

accint

A positive real number: the numerical integration scheme generates points and weights such that a large number of representative test integrals are evaluated with an accuracy of accint significant digits. The default for accint depends on the runtype: 4.0 for Single Point runs and simple Geometry Optimizations, including Linear Transits; 5.0 for Transition State searches; 6.0 for the computation of Frequencies; 10.0 in Create runs.

The number of integration points varies strongly with accint, and this determines to a large extent the computational effort. Decreasing accint from 4.0 to 3.0 for instance roughly halves the number of points (this depends somewhat on the molecule).
The defaults should yield good precision for the very large majority of applications. Lower values (3.0 or even 2.0) can be used if precision is not crucial and the purpose is to get an impression. We recommend that you experiment for yourself to get a feel for how results may vary in quality and computing time.
The default in Create mode is very large: 10.0. This is computationally no problem thanks to the simplicity of the single atom case, in particular due to the high symmetry. There is no reason to override the default INTEGRATION settings when creating basic atoms.

**Frequencies**

The computation of *frequencies* should always be carried out with accint at least 4.0 to get results that make sense; many molecules, in particular those than contain metals, require higher precision, which is why the default is 6.0. The reason is that frequencies are computed by numerical differentiation of gradients computed in slightly displaced geometries. Obviously the *noise* in the gradients due to numerical integration errors should be small compared to the difference in gradients across the different geometries and since the latter are very close, a high numerical accuracy is required [71-73].

**Self-adapting precision during optimizations**

In Geometry Optimizations and Transition State searches, the gradients at the initial geometry may be quite large and in such case there is no need to apply the same high integration precision as may be required close to convergence when the gradients become very small. Therefore, ADF tries to reduce the numerical integration accuracy during optimization to save time, and use the user-specified precision only in the final stage of the optimization.

The program starts with an initial value accfirst, to get an assessment of the (local) gradients. Subsequently it adjusts it according to the progress towards convergence. All values are kept between a lower and an upper bound: accmin and accmax respectively.

All three parameters, accfirst, accmin, accmax, are controlled by the key **INTEGRATION**.

```
| INTEGRATION  accl  {acc2  {acc3}} |
```

The simplest application, discussed above, specifies one value (accl). This defines then both the upper bound (accmax) and the first value (accfirst). The lower bound accmin is by default 3.0 (adjusted internally to accl, if that is lower); 3.5 in Transition State searches.

If **two** values are supplied, the smallest is taken for the lower bound, the larger for the upper bound. The value for the first cycle equals the upper bound.

If **three** values are supplied, the smallest is the lower bound, the largest the upper bound, the remaining value is used to start with.

**More integration options**

We’ve now only explained the normal, simple application of the INTEGRATION key, which we hope and expect is adequate for all your computations. In chapter 2.3 “More Options” additional details will be discussed. The distribution of points over space is internally regulated by quite a few parameters. Each of these parameters can be controlled in input. By default they depend on one another, and all of them depend on the main parameter accint. Advanced users may wish to experiment and override the default relations between the parameters.
You may also have rather non-standard applications where the default relations are less adequate. A thorough understanding of the underlying method is required to make a sensible choice for all parameters [70, 74].

**SCF**

The SCF procedure is regulated with keys that set the maximum number of iterations, the convergence criterion, and various items that control the iterative update method. Molecules may display wildly different SCF-iteration behavior, ranging from easy and rapid convergence to troublesome oscillations. We expect that the default settings take care of most cases, but one should realize that this is a difficult and tricky subject. The user has a few (main) options to adapt the procedure to the situation at hand: simple damping or the DIIS procedure (Direct Inversion in the Iterative Subspace). Either of them can be combined with Level-Shifting.

At each cycle the density is computed as a summation of occupied orbitals (squared); the new density defines the new potential from which the orbitals are re-computed, et cetera, until convergence is reached. To speed-up convergence and to avoid non-convergent oscillatory behavior the values at the next iteration are constructed as a mixture of the computed new data and those used at the cycles before. This may involve only the previous cycle and is then called *damping*. Alternatively the DIIS procedure can be invoked, which is a generalization of damping to include more previous iterations.

Subkeys in the data block of the master key SCF control the aspects mentioned above. Each subkey is optional. Omission means the application of default values. Omission of the SCF key altogether implies defaults for all subkeys.

```
SCF
  ITERATIONS   Niter
  CONVERGE     scfcnv { sconv2 }
  MIXING       mix
                DIIS   {N=n} {OK=ok} {CX=cx} {CXX=cxx} {BFAC=bfac} {cyc=cyc}
                LSHIFT vshift   {Err=shift_err}   {Cyc=shift_cyc}
END
```

**Niter**
The maximum number of SCF cycles allowed. In case of Geometry Optimizations it applies separately to each of the SCF procedures that are executed. Default is 50 (in Create mode: 100). The program executes at least one cycle, even if you specify a non-positive number.

**SCFcnv**
The criterion to stop the SCF updates. The tested error is the commutator of the Fock matrix and the P-matrix (=density matrix in the representation of the basis functions) from which the F-matrix was obtained. This commutator is zero when absolute self-consistency is reached. Convergence is considered reached when the maximum element falls below scfcnv and the norm of the matrix below 10*scfcnv. The default is 1e-6 (in Create mode: 1e-8).

**sconv2**
A second criterion which plays a role a) during geometry optimizations and b) when the SCF procedure has difficulty converging.
During an optimization the SCF convergence criterion is relaxed as long as the geometry has not yet converged. At the start-up geometry, and at the final geometry, the normal criterion (\texttt{sfcnv}) is applied, at intermediate cycles the criterion is adjusted depending on how far the geometry has converged. \texttt{sconv2} defines a minimum criterion: the actual criterion in effect will not be less than \texttt{sconv2}.

When in any SCF procedure the currently applicable criterion does not seem to be achievable, the program stops the SCF. When the secondary criterion (\texttt{sconv2}) has been met, only a warning is issued and the program continues normally. If the secondary criterion was not met either, the program terminates any further geometry optimizations, frequency steps, etc. You can prevent the program from terminating in such a case with the key \texttt{ALLOW}.

The default for \texttt{sconv2} is \texttt{1e-3}.

\textbf{Mix}

The relative weight of the new potential, as computed from the occupied orbitals, to be mixed with the potential that was used in the previous cycle, to define the potential for the next. Mixing is used only if, and as long as, the DIIS procedure (see below) is not operational. Default: 0.2.

For problematic systems that require strong damping, one should decrease the mix-parameter.

\textbf{DIIS}

The \texttt{DIIS} subkey specification(s) can be given to control the DIIS procedure. Each of these specifications is optional. Simple damping will be used during the first few cycles, until the DIIS procedure becomes operational. Two conditions must be satisfied for this: 1) at least two iterations must have been done anyway (to build up sufficient information for the DIIS to work at all) and 2) the error must be small enough; see however the \texttt{cyc} option below.

There have been claims in the literature that the DIIS should not be used until fair convergence has been reached. Our experience thus far does not indicate that this should be taken too seriously, except in special situations. To allow the user complete control, the start-up criteria can be set in input.

\textbf{N}

The number of expansion vectors used in the DIIS. The number of previous cycles taken into the linear combination is then \texttt{N}-1 (the new computed potential is also involved in the linear combination.

Default \texttt{N}=10. An input value smaller than 2 disables the DIIS.

\textbf{OK}

The DIIS starting criterion. The DIIS procedure is not invoked until a) the maximum commutator element is smaller than \texttt{OK} (default: 0.5) or b) a certain number of SCF cycles has been executed.

\textbf{Cyc}

The SCF cycle no. at which the DIIS will start irrespective of the OK value above. Default: 5.

\textbf{Cx}

An upper bound on linear combination coefficients as applied in the DIIS. As soon as any coefficient exceeds \texttt{Cx}, all information about older cycles but the last two is discarded and the DIIS starts again to accumulate info from the current cycle on. The computed linear combination, with the large coefficient(s), is used for the next iteration, however. Default=5.0

\textbf{Cxx}

A second upper bound on the coefficients (should in principle be bigger than \texttt{Cx}). When a coefficient exceeds \texttt{Cxx}, the computed linear combination is not used for the next cycle, but simple damping is applied.

\textbf{Bfac}

A factor to bias the DIIS combination vector in favour of the new computed potential. Default=0 (no bias). A sensible alternative value, advocated in [75] is 0.2.
VShift
The level shifting parameter. The diagonal elements of the Fock matrix, in the representation of the orbitals of the previous iteration, are raised by $v_{\text{shift}}$ hartree energy units for the virtual orbitals. This may help to solve convergence problems when during the SCF iterations charge is sloshing back and forth between different orbitals that are close in energy and all located around the Fermi level.

Shift_err
Specifies that level shifting will be turned off by the program as soon as the SCF error drops below a threshold; default value: 1e-2

Shift_cyc
Specifies that level shifting is not turned on before the given SCF cycle number (for the start-up geometry); default value: 1.

Note1: very strong damping, i.e. a very small value of mix such as 1e-3, may not combine very well with the DIIS procedure. The reason is that with strong damping successive SCF cycles tend to be very similar and the vectors building up the DIIS space become linearly dependent. We recommend in difficult cases either to use a not too strong damping ($\text{mix}=0.03$) or to use strong damping while the DIIS is disabled (by setting $n=0$ for the DIIS subkey) during a limited number of SCF iterations, and then restart with DIIS activated and less stringent damping.

Note2: Another feature, electron smearing, may be used to overcome convergence difficulties. The idea is to distribute electron occupations fractionally over a few states around the Fermi level, by a pseudo-thermal distribution function. This aspect is controlled with the \texttt{SMEAR} option to the \texttt{OCCUPATIONS} key. One should be aware that the applied distribution of occupations is not really an approximation to the finite-temperature case. In fact, the results are unphysical and one should not use the results as a meaningful outcome. The smearing trick is only to be used to overcome convergence difficulties. Having reached convergence with it, one should typically do a follow-up restart calculation without smearing, using the converged outcomes to hopefully get the thing to converge properly. A typical ‘allowed’ application is the usage of smearing during geometry optimizations, because the intermediate geometries are not relevant anyway and only a step towards the final results. By default, the program does not apply any smearing unless during a geometry optimization. See the \texttt{OCCUPATIONS} key for more details.

**Interpretation of Input**

ADF has two special keys that regulate the specification and interpretation of numerical data in input. These keys, and related aspects, are convenient for the formatting of input.

The position of the interpretation keys in the input file is significant! Therefore, to avoid problems and misunderstandings: before supplying any numerical data, specify first (if at all) the keys \texttt{UNITS} and \texttt{DEFINE} (see below).
**Units of length and angle**

Geometric lengths and angles are in units defined by:

```
UNITS
  LENGTH  Angstrom / Bohr
  ANGLE   Degree / Radian
END
```

Angstrom and Bohr, respectively Degree and Radian, are recognized strings. Each of the subkeys is optional, as is the key UNITS itself. Defaults: Angstrom for lengths, and Degree for angles.

The position of the key UNITS in input is significant as regards the evaluation of expressions (see the paragraph on constants and functions below). In other respects its position plays no role. To avoid mistakes one should place UNITS as early as possible in input (if at all).

**Expressions**

ADF supports the use of arithmetic expressions, functions, and constants to represent numerical data. This can be convenient for the input of, for instance, atomic positions when these would most easily be represented in terms of 1/3, sin(360/5), et cetera. Using expressions and functions is easier, avoids the tedious typing of long decimal expansions and solves the question of precision (how many digits should be supplied?).

The standard arithmetic operands in Fortran (+ - * / **) can be applied in expressions, together with parentheses where suitable.

Blanks are allowed and ignored, but they are interpreted as separators, i.e. as denoting the end of an expression, whenever the part until the blank can be evaluated as a correct expression. For instance 3 * 4 will be interpreted as 12, but 3 * 4 will be interpreted as 3, followed by a character *, followed in turn by the number 4.

All numbers and results are interpreted and handled as being of type real, but whenever the result is a whole number (allowing for very small round-off) it will be recognized and accepted as an integer when such data is required.

**Constants and functions**

The user may define constants and functions in the input file, and apply them subsequently in expressions. The input file is read sequentially and constants and functions must be defined before they can be used. The argument list of a function must be enclosed in parentheses and the arguments, if more than one, separated by commas.

The following functions are predefined in ADF and can be used directly in input: sin, cos, tan, asin, acos, atan, exp, log, sqrt, nint. Each of them has one argument. log is the natural logarithm (base e).
No constants are predefined.

The angular argument to the trigonometric functions \( \cos, \sin, \tan \) is in the unit for angles as defined by \texttt{UNITS}, \textit{provided the unit has been set before it is applied}. For the result of the inverse trigonometric functions the same holds.

Constants and functions can be defined with the block key \texttt{DEFINE}:

\begin{verbatim}
DEFINE
  angle=54
  ab = sin(angle/3)
  s13 = 14*sqrt(2)
  func(x,y,z) = x*ab+y**2-y*z
END
\end{verbatim}

The constants \( \texttt{angle} \), \( \texttt{ab} \), and \( \texttt{s13} \) are defined together with a function \( \texttt{func} \), using the predefined functions \( \texttt{sin} \) and \( \texttt{sqrt} \). These can then be applied to assign values elsewhere in input.

In the example above, the constant \( \texttt{angle} \) is used in the definition of \( \texttt{ab} \), and \( \texttt{ab} \) is used in turn to define \( \texttt{func} \); these constructions are allowed because \( \texttt{angle} \) is defined before \( \texttt{ab} \), and \( \texttt{ab} \) is defined before \( \texttt{func} \).

The replacement of constants, functions, and other expressions by their numerical values may considerably increase the length of the input record, in particular when real values are being generated (by the parser) in the standard format \texttt{E22.14}. Take care that the resulting record does not exceed 80 characters. The program will abort or may run into an error if this is violated.

The input-reading routine applies the constants and functions wherever it is allowed to do so. To prevent any unwanted replacements in the input file you should avoid very short identifiers for constants and functions.

Warning example:

\begin{verbatim}
DEFINE
  A=3.18
  C=4.12
END
...
ATOMS
  C  0.00  1.05 -3.22
...
\end{verbatim}

The program will apply the definition of the variable \( \texttt{C} \) and read:
Define
A=3.18
C=4.12
End
...
Atoms
4.12 0.00 1.05 -3.22
...

Avoid single-character identifiers!

Strings
Quotes can be used to designate strings, i.e. (parts of) records which are not to be parsed for expressions, but which should be taken as they are. The quotes themselves are ignored, i.e. removed by the parser. Two consecutive quotes inside a string are interpreted to denote the (single) quote character as a part of the string.

Where does parsing apply?
Replacing pre-defined variables and expressions by their value is applied only to keys that carry numerical data. For example: Atoms, Define, Units. However, it is not applied to keys that carry electronic occupation numbers.
Note that when parsing applies to a given key the whole record of the key (key + argument) and its data block are parsed. The parsing then applies to all items, even those that in themselves have no numerical meaning (for instance, the atom type names in the Atoms data block are scanned and must of course then not be ‘defined’ as identifiers with a numerical value.

Constants vs. geometric parameters
Note carefully the difference between constants defined with Define and identifiers that are used for atomic coordinates in the data blocks of Atoms and GeoVar. Constants defined under Define are merely symbols for, and exactly equivalent to, certain numerical values, whereas the coordinate identifiers carry implications such as the distinction between frozen and optimization coordinates. Constants affect only the input after their definition and the location of their definition in the input file is significant. Geometric identifiers only relate to the data blocks of Atoms and GeoVar respectively and the relative order in which the keys Atoms and GeoVar occur is irrelevant.
Restarts

Check-point file

When an ADF calculation terminates abnormally – not controlled by the program itself, for instance after a core dump due to some bug – there will usually be a file TAPE13, which serves as a checkpoint file. TAPE13 can be used to restart the calculation at a point not too far before the fatal condition occurred. It contains only data for the restart, but none of the special analysis data on TAPE21 that would be useful for analysis, to serve as fragment file, etc.

TAPE13 is upgraded during the calculation but discarded upon normal termination, namely when all relevant information has been saved on TAPE21. At that point all info that would have been on TAPE13 is present on TAPE21. If you wish to keep TAPE13 anyway – for instance because you plan a restart after normal termination and don’t intend to keep the substantially bigger TAPE21 – you must use the SAVE key.

Upon normal (i.e. program-controlled) termination of a calculation, the TAPE21 result file can be used for restart purposes. When a crash occurs, however, chances are that TAPE21 has not correctly been closed and that its data structure is inconsistent: during the calculation large portions of TAPE21 are kept in memory rather than on file, and only at the point of final termination, all data is flushed to file.

General remarks

In all restart calculations a normal input file must be supplied (you can, for instance, simply take the original one), with a specification of the restart file added: the restart file does not replace the input file. From the program’s point of view, it first reads the ‘normal’ input file and then inspects whether a restart file is present to replace some of the information read from input.

The concept of restarts in ADF is rather simple and primarily directed at increasing computational efficiency by providing cost-expensive data. The continuation run is to a large extent independent from the one that generated the restart file. The runtype, the choice of density-functional and other features in the Hamiltonian, precision of numerical integration, thresholds on convergence, etcetera are all determined solely from the input file for the new run: no such data is read from the restart file. Most input items should, therefore, be supplied in the restart run again, even if it is a direct continuation of a previous calculation: omission implies using the standard defaults, which are not necessarily the settings of the calculation that generated the restart file.

Even the key ATOMS with the list of atomic coordinates must be supplied again: the program needs the information herein to deduce what fragments are used, which coordinates are free or frozen respectively in an optimization, etc. The coordinate values may be supplied with the restart file and these will then overwrite those specified in the input file.

Obviously, the two runs cannot be completely unrelated. To let the restart data make sense the runs should correspond to the same molecule (i.e. its general definition in terms of fragment building blocks). The program does not check all aspects related to this and certain abuses will therefore survive the internal tests, but will surely lead to some error later on: it is the user’s responsibility to ensure that the restart data match the calculation one has in mind.
Interdependencies between data read from the restart file (rather than from input or fragment files) and other
items imply that some input keys and some options to specific keys may be inaccessible when restart data are
provided. In most cases supplying such inaccessible input options will simply be ignored; in some cases a
warning is issued or an error abort occurs.

A restart file supplies data from a previous run that might be useful in the current one. The applications are
( combinations are possible):
- Get a better start in the (first) SCF procedure by providing the electronic charge density (in the form of
fit coefficients) from the preceding run,
- Continue an optimization by supplying the latest geometry (coordinates) from a previous run via the
restart file (rather than typing them in),
- Get faster geometry convergence by supplying a Hessian,
- Breaking large jobs (Linear Transits, Frequencies) in smaller ones, each time doing a part and passing
this on to the continuation run.

WARNING. The SCF and optimization procedures use history to improve convergence behavior. Most of such
history information is not stored on a restart file. As a consequence, a restart may not continue exactly as the
original run would have done if it hadn't terminated. In a SCF restart, for instance, the DIIS procedure has to
rebuild the information. The same holds for geometry optimizations, although history plays usually not a very
big role there.

The RESTART key

The name of the restart file must be provided with the key RESTART (see below). A list of data items is read
from the file (if present on the file and only as far as significant for the new run) and used unless their usage is
explicitly suppressed by the user.

```
RESTART       restartfile { &
  optionlist
  optionlist
  ...
  END}
```

RESTART
This general key can be used as a simple key – to supply the name of the restart file – or as a block key. In
the latter case the continuation code (&) must be applied to tell the program that a data block follows.

restartfile
The name of a file with restart data. The path (absolute or relative) to the file must be included if the file is not
local to the directory where the calculation executes. In most cases it will be a TAPE21 file from an ADF
calculation, but this is not necessary. It may be any file – constructed by the user for instance – provided it
has the right structure. It must be a KF file and the data to be used must be stored in sections and under
variable names as defined below, which is exactly how such data are generated by a normal adf run on
TAPE21 or on the checkpoint file TAPE13.
Note: the filename must not be one of the standard filenames used internally by the program, such as TAPE21, TAPE13 etc. Generally: don't use a name like TAPENN where nn is a two-digit number.

optionlist

A list of options, separated by blanks or commas. The following options are applicable:

NOSCF
Do not use any fit coefficients from the restart file as a first approximation to the (fitted) scf density for the new calculation. Instead, the sum-of-fragments density will be used, as in a non-restart run.

NOGEO
Do not use the geometry – Cartesian, Z-matrix, etc. coordinates – from the restart file.

NOHES
Do not use any Hessian from the restart file.

Note: in the continuation of a Linear Transit, IRC or Frequencies run, geometric data are read from the restart file and will be used: the option NOGEO is ignored. In a continued Frequencies run the input coordinates (key ATOMS) must be correct (i.e. the equilibrium geometry). In a continued LT or IRC run, the input coordinate values from ATOMS are ignored (but they must be supplied to give the program a preliminary count of atoms and fragments involved).

Structure of the restart file

All data that may be retrieved from the restart file must be stored in a specific location on the restart file. If you're simply using a TAPE21 result file or a TAPE13 checkpoint file you don't need to bother about this: ADF has put all data in the right place; the following discussion is primarily for those who want to manipulate the restart file or even construct one themselves.

Since the restart file must be a KF file, the location of the data is of the form Section%Variable, specifying the section and the variable name. The section and variable names are case sensitive. See the UTILITIES document for general information about KF files.

If the specified variable is not present in the specified section on the restart file – or if there is no such section at all – the data is not used, usually without an error message. In some cases a few global tests are carried out on the retrieved data; if they fail the tests the data are not used and a warning – in some cases an error abort – may be issued by the program.

KF files are binary files and so are the TAPE21 result file, the TAPE13 checkpoint file and generally any restart files. If you wish to edit and modify the contents, or just inspect them, the standard KF utilities can be used. Apply pkf to get a survey of the sections and variables on the file, dmpkf to get a complete ASCII version of the file and udmpkf to transform an ASCII version – presumably edited and modified – back into binary format. See the UTILITIES document.

Data on the restart file

Follows a survey of all data items that the program may search for on the restart file.
**SCF data**

**Fit%coef**

The fit-expansion of the charge density to be used as start-up for the next SCF. Without these restart fit data the first SCF will start from the (fitted) sum-of-fragments charge density.

**Fit%coef_FreqCenter**

Only in a Frequencies run: the fit-expansion of the SCF-converged equilibrium geometry. It usually helps to get a somewhat better start-up of the SCF in displaced geometries.

If the NOSCF option is used to the RESTART key, any Fit%coef_? data on the restart file are ignored.

**Coordinates**

**Geometry%xyz**

Cartesian atomic coordinates. The option NOGEO suppresses using such data. In a Frequencies or continued Linear Transit run, they may be read but will be ignored (i.e. replaced by other coordinates data from the restart file).

In most applications, when coordinates are read (and used) from the restart file, only Cartesian coordinates are retrieved and the corresponding Z-matrix values are computed from them, using the Z-matrix structure defined in the ATOMS data block. This is one of the reasons why the ATOMS key must be used even when the atomic coordinates are supplied on the restart file.

**Hessian**

**GeoOpt%Hessian_CART**
**GeoOpt%Hessian inverted_CART**
**GeoOpt%Hessian_ZMAT**
**GeoOpt%Hessian inverted_ZMAT**

All these four varieties are searched for if the new run searches for a restart Hessian matrix at all, that is: in an optimization, Linear Transit or Transition State search. As the names should suggest these variables stand for the Hessian, respectively the inverse of the Hessian in Cartesian or Z-matrix coordinates.

In all cases the full square matrix must be present, with dimension the number of atomic coordinates, 3 times the number of atoms. This holds also for Z-matrix coordinates. The 6 dummy coordinates play no role, the corresponding matrix elements in the Hessian should be zero.

If a Hessian is searched for on the restart file, all four possibilities above are tried and the first one found is used, the other ones being ignored. The order in which they are tried is:

If the current run uses Cartesian coordinates as optimization variables, then first the two CART varieties are tried, and vice versa for Z-matrix optimization.

In a minimization (simple optimization or Linear Transit) first the inverted variety is tried; in a Transition State search the normal (not inverted) Hessian is looked for first.
Note: If a Z-matrix Hessian is retrieved from the restart file the program will use the underlying Z-matrix structure to derive a Cartesian Hessian from it. In such case the restart file must also contain:

GeoOpt%kmatrix

The Z-matrix structure (references to the atoms in this matrix assume the ordering of atoms as used internally by the program).

Note: the kmatrix on the file need not be identical to the kmatrix used in the current calculation. In fact, the current calculation may not even have a Z-matrix structure.

**Transition State**

In a continued TS run the program retrieves, apart from general geometry optimization data such as the Hessian – see above – only the latest TS search vector: the eigenvector of the (approximate) Hessian that points to the Transition State. All other TS-specific data are input-determined with corresponding defaults. The TS search vector is stored in:

TS%mode to follow

A list of atomic coordinates (Cartesian or Z-matrix, depending on the type of optimization variables used. The underlying list of atoms has the atoms not necessarily in the order in which they have been given in input: rather they are grouped together by atom type.

**Linear Transit**

In a continued Linear Transit (LT) calculation the continuation run proceeds from where the previous run stopped. The total number of points by which the transit is scanned, the current point (its index and the Cartesian coordinates), the accumulated results of completed points on the transit etc. are copied from the restart file. If the restart file contains a section LT, then all relevant data must be present on it and correct (i.e. matching those of the current run: same number of LT parameters, and of course the same molecule.

LT%nr of points

The number of points by which the LT is scanned; this is identical to the Fortran variable ltimax in the code. The value on the restart file applies in the calculations and overwrites any input/default value (see the subkey LINEARTRANSIT of the GEOMETRY block)

lt%current point

Index of the current LT scan point. This is where the program will continue. In a non-restart LT run, this index initializes at 1.

LT%Energies

An array with energy values, one for each LT point. When the LT run is completed, this array allows you to map out the energy along the LT path. The values for the completed LT points are stored on the restart file. This size of the array on the restart file must (at least) be the total number of points on the complete path.
LT%Parameters
Initial and final values for the LT parameters, which describe roughly the path (all other coordinates may be optimized at each point, depending on other input keys). The values from the restart file overwrite input values. The input values should be supplied, however, as if it were a non-restart run.

LT%atmcrd
ZMAT if a Z-matrix structure is available for the molecule, CART otherwise. This is used to control printing of results. It does not define the type of optimization variables; see the next item.

LT%geocrd
ZMAT or CART: the type of optimization variables. This defines in which type of coordinates the LT parameters are defined and any optimization of other coordinates takes place.

LT%xyz
Cartesian coordinates for all LT points: 3*atoms*LTpoints. The size of the array must conform to this. Only the values of the completed LT points and those of the current point are relevant. Those of the current LT point are used as initial coordinates to start the current run.

LT%zmatrix
Same for the Z-matrix coordinates. They should match the Cartesian coordinates for the completed LT points (this is not checked). Those for the current LT point will be recomputed from the current Cartesian coordinates.

IRC
In a continued Intrinsic Reaction Coordinate (IRC) calculation, the continuation run processes the path(s) as specified in input. Any info for such path(s) on the restart file will then be used to continue from there. If the restart file contains the relevant IRC sections, see below, then all relevant data must be present on it and correct (i.e. matching those of the current run).

The sections on file pertaining to the IRC are:
IRC: this section contains information about the central (TS) point, which variables are optimized in each of the IRC points, the connection matrix defining the z-matrix structure, etc.
IRC_Forward and IRC_Backward: these sections contain the data of the two paths from the Transition State down to the two adjacent local energy minima: for each point the distance from the previous point and the local curvature and molecular properties such as energy, atomic charges and dipole moment.

LT%nr of points
The number of points by which the LT is scanned; this is identical to the Fortran variable _ltimax in the code. The value on the restart file applies in the calculations and overwrites any input/default value (see the subkey LINEARTRANSIT of the GEOMETRY block)

LT%current point
Index of the current LT scan point. This is where the program will continue. In a non-restart LT run, this index initializes at 1.

LT%Energies
An array with energy values, one for each LT point. When the LT run is completed, this array allows you to map out the energy along the LT path. The values for the completed LT points are stored on the restart file. This size of the array on the restart file must (at least) be the total nr of points on the complete path.
**lt%Parameters**

Initial and final values for the LT parameters, which describe roughly the path (all other coordinates may be optimized at each point, depending on other input keys). The values from the restart file overwrite input values. The input values should be supplied, however, as if it were a non-restart run.

**lt%atmcrd**

ZMAT if a Z-matrix structure is available for the molecule, CART otherwise. This is used to control printing of results. It does not define the type of optimization variables: see the next item.

**lt%geocrd**

ZMAT or CART: the type of optimization variables. This defines in which type of coordinates the LT parameters are defined and any optimization of other coordinates takes place.

**lt%xyz**

Cartesian coordinates for all LT points: 3*atoms*LT points. The size of the array must conform to this. Only the values of the completed LT points and those of the current point are relevant. Those of the current LT point are used as initial coordinates to start the current run.

**LT%zmatrix**

Same for the Z-matrix coordinates. They should match the Cartesian coordinates for the completed LT points (this is not checked). Those for the current LT point will be recomputed from the current Cartesian coordinates.

---

**Frequencies**

In the continuation of a Frequencies calculation all Frequencies-related data are retrieved from the section Freq on the restart file. (SCF fit data are, as always, retrieved from the section Fit). A fairly large number of items will be read and must all be present (if a section Freq is present in a restart file supplied to a Frequencies run). Technical parameters such as the type of numerical differentiation, size of displacements etc. are read from the restart file. Any input specifications are ignored.

**Freq%kountf**

Counter of number of geometries completed. In a non-restart run this is initialized at zero; in a restart it is read from the file.

**Freq%nraman**

Flag for RAMAN calculations

**Freq%numdif**

1 or 2: defines numerical differentiation used to compute the force constants from the gradients in slightly displaced geometries (by 1-point or 2-point differentiation).

**Freq%disrad**

Size of displacement for cartesian or bond-length displacements.

**Freq%disang**

Size of angular (bond angle, dihedral angle) displacements.

**Freq%atmcrd**

ZMAT or CART: specifies whether a Z-matrix structure is present. This does not define the type of displacement coordinates, see the next item.

**Freq%geocrd**

Type of coordinates in which the displacements are carried out: ZMAT or CART.
Freq%nfree
Number of free and independent displacement variables.

Freq%idfree
References from the atomic coordinates (in internal order) to the independent displacement variables.

Freq%all freedoms
(logical) flags whether or not the complete energy surface is scanned around the equilibrium or only part of the internal degrees of freedom are used.

Freq%xyz
equilibrium coordinates (internal order of atoms).

Freq%kmatrix
Z-matrix structure. Pointers are indexed by and refer to atoms in the internally used order.

Freq%zmatrix
Z-matrix coordinates of the equilibrium geometry (internal ordering of atoms).

Freq%rigids
6 rigid motion vectors (one may be zero, in case of a linear molecule). Each vector has as many components as there are atomic coordinates. The values correspond to the internal ordering of atoms.

Freq%xyz displaced
Cartesian coordinates of displaced geometry to carry out now. In a non-restart run this would be the equilibrium geometry.

Freq%zmatrix displaced
Similar for the Z-matrix coordinates.

Freq%Dipole previous
dipole vector (3 components) for the last geometry handled.

Freq%Dipole
dipole at the equilibrium geometry.

Freq%Dipole derivatives
Derivatives of the dipole wrt atomic coordinate displacements.

Freq%Gradients previous
Energy gradients (derivatives wrt atomic coordinate displacements) in the last handled geometry.

Freq%Force constants
Matrix of force constants. This is, together with the Dipole derivatives the final quantity to compute. At each cycle of the Frequencies data are added to it. Upon completion of the Frequencies cycles the frequencies and normal modes are computed from it. Together with the dipole derivatives it then also yields the InfraRed intensities.

**Printed Output**

The amount of printed output is regulated with the keys `PRINT`, `NOPRINT`, `EPRINT` and `DEBUG (NO) PRINT` and `DEBUG` are simple keys, `EPRINT` is a block type key.

Many print options pertain to debug situations and are included here only for completeness. This section is intended to give a survey of all possibilities. Some items may be mentioned again in other sections where the subject of a particular print switch is discussed.
PRINT / NOPRINT

<table>
<thead>
<tr>
<th>PRINT Argumentlist</th>
</tr>
</thead>
<tbody>
<tr>
<td>NOPRINT Argumentlist</td>
</tr>
</tbody>
</table>

Argumentlist
A sequence of names separated by blanks or commas.
The keys PRINT and NOPRINT may occur any number of times in the input file. The names in the argument list may refer to various items. For some of them printing is normally on, and you can turn them off with NOPRINT. For others the default is not printing; use PRINT to override that.

Follows a list of the recognized items that are applicable in the argument lists, with a short explanation and defaults. Item names must be used exactly as given in the table – abbreviated or elongated forms will not be recognized – but they are not case sensitive.

<table>
<thead>
<tr>
<th>Item</th>
<th>Default</th>
<th>explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>$ALL</td>
<td>No</td>
<td>Turns on all print options. This will not be affected by any additional NOPRINT instructions. Be careful: this generates a large amount of output. To be used only for debugging purposes.</td>
</tr>
<tr>
<td>Atdist</td>
<td>No</td>
<td>Inter-atomic distance matrix at each new geometry (in an optimization)</td>
</tr>
<tr>
<td>Bas</td>
<td>Yes</td>
<td>General control of output related to elementary basis functions (BAS).</td>
</tr>
<tr>
<td>BlockCheck</td>
<td>No</td>
<td>Intermediate data during the determination of the block length. (see Blocks)</td>
</tr>
<tr>
<td>Blocks</td>
<td>No</td>
<td>Numerical integrals, consisting of loops over large numbers of points, are split up in loops over blocks of points. The block length is determined by the available amount of workspace. Given this amount, the maximum block lengths, according to memory usage in a few relevant routines, are computed (and printed with this PRINT option) and used to impose upper bounds on the block length actually use.</td>
</tr>
<tr>
<td>Character-Table</td>
<td>No</td>
<td>Table of characters for the irreducible representations of the pointgroup symmetry.</td>
</tr>
<tr>
<td>Computation</td>
<td>Yes</td>
<td>Reports progress of the computation, with (concise) info about each SCF cycle and each Geometry update in an optimization.</td>
</tr>
<tr>
<td><strong>Core</strong></td>
<td>No</td>
<td>Description of the frozen core: frozen core expansion functions (CORBAS) and the expansion coefficients for the frozen orbitals. This printing can only be activated if Functions is also on, otherwise it is ignored.</td>
</tr>
<tr>
<td>--------------------</td>
<td>------</td>
<td>------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td><strong>CoreOrt</strong></td>
<td>No</td>
<td>The valence basis set contains auxiliary Core Functions. They are not degrees of freedom but are used solely to ensure orthogonalization of the valence set to the frozen Core Orbitals. The orthogonalization coefficients and some related overlap matrices are printed.</td>
</tr>
<tr>
<td><strong>CoreTable</strong></td>
<td>No</td>
<td>Internally the charge density and potential of the atomic frozen cores are processed as tables with values for a sequence of radial distances. A few initial and a few final values from these tables are printed, along with the (radial) integral of the core density, which should yield the number of core electrons.</td>
</tr>
<tr>
<td><strong>EKin</strong></td>
<td>No</td>
<td>At the end of SCF: Kinetic energy of each occupied MO.</td>
</tr>
<tr>
<td><strong>EndOf</strong></td>
<td>No</td>
<td>Flags the exit from a few major routines, with CPU times used in these modules. Primarily a debug tool.</td>
</tr>
<tr>
<td><strong>EPauli</strong></td>
<td>Yes</td>
<td>The repulsive Pauli term in the bonding energy (also called exchange repulsion) with its decomposition in density functional (LDA and NL) and Coulomb terms.</td>
</tr>
<tr>
<td><strong>Fit</strong></td>
<td>Yes</td>
<td>General control of output related to the density fitting.</td>
</tr>
<tr>
<td><strong>Fmat</strong></td>
<td>No</td>
<td>Fock matrix computed at each cycle of the SCF.</td>
</tr>
<tr>
<td><strong>ForceConstants</strong></td>
<td>Yes</td>
<td>Force constants matrix (Frequencies run only)</td>
</tr>
<tr>
<td><strong>FreqHess</strong></td>
<td>No</td>
<td>matrix of force constants (Frequencies run) after each applicable step in its processing: transformation from/to Cartesian and Z-matrix coordinates, symmetrizations, ...</td>
</tr>
<tr>
<td><strong>Frag</strong></td>
<td>No</td>
<td>General control of output related to build-molecule-from-fragments.</td>
</tr>
<tr>
<td><strong>Functions</strong></td>
<td>Yes</td>
<td>List of employed Slater-type exponential basis functions and fit functions.</td>
</tr>
<tr>
<td><strong>Gradients</strong></td>
<td>No</td>
<td>detailed info of computed energy gradients (in optimization runs)</td>
</tr>
<tr>
<td><strong>Group-Operators</strong></td>
<td>No</td>
<td>3*3 matrices of pointgroup symmetry operators, with the axis and angle of rotation</td>
</tr>
<tr>
<td><strong>HessEig</strong></td>
<td>No</td>
<td>Eigenvalues of the Hessian in each cycle of a Geometry Optimization. The print-out in the intermediate cycles is suppressed if output of updated coordinates etc. is turned off (see the EPRINT subkey REPEAT (option GeoStep).</td>
</tr>
<tr>
<td>Feature</td>
<td>Default</td>
<td>Description</td>
</tr>
<tr>
<td>------------------</td>
<td>---------</td>
<td>----------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>Idfree</td>
<td>No</td>
<td>List of free atomic coordinates with indication whether they are optimization coordinates (this info is also contained in the output of new atomic coordinates at each step of an optimization)</td>
</tr>
<tr>
<td>Inertia</td>
<td>No</td>
<td>Warning message in the log file in case of zero product of moments of inertia (this may correctly be the case for certain molecules)</td>
</tr>
<tr>
<td>Inputkeys</td>
<td>No</td>
<td>List of keys that were specified in input, together with some of the associated data. The list is printed directly after the echo of the Input File, before the header with ADF program information. A few special keys will not be echoed: (NO)PRINT, (NO)SKIP, ALLOW.</td>
</tr>
<tr>
<td>Irrep-Matrices</td>
<td>No</td>
<td>Irreducible representation matrices</td>
</tr>
<tr>
<td>Logfile</td>
<td>Yes</td>
<td>At the end of the calculation a copy of the log file is appended to standard output</td>
</tr>
<tr>
<td>LOW</td>
<td>No</td>
<td>Construction of the LOW basis from the elementary BAS functions and from the SFOs: combination coefficients</td>
</tr>
<tr>
<td>LOWMO</td>
<td>No</td>
<td>MOs are printed in the LOW (Löwdin) representation, in the RESULTS section</td>
</tr>
<tr>
<td>OvlBAS</td>
<td>No</td>
<td>Overlap matrices processed during the construction of the LOW basis</td>
</tr>
<tr>
<td>Parser</td>
<td>No</td>
<td>Most input records are echoed twice (at the very beginning of output). First the original version, then the parsed version in which expressions have been replaced, redundant blanks removed, etc. The parsed version is what the program really uses as input. Comment blocks, and function definitions (in DEFINE blocks) are not parsed, and are not affected by this switch. If the print switch is off only the original, non-parsed input record is echoed in output. This print switch affects only the part of input after its occurrence.</td>
</tr>
<tr>
<td>Pmat</td>
<td>No</td>
<td>The density matrix (in Löwdin representation) in each cycle of the SCF.</td>
</tr>
<tr>
<td>QMpot</td>
<td>Yes</td>
<td>At the end of the SCF for each atom the electrostatic potential at its nucleus (excluding its own contribution of course).</td>
</tr>
<tr>
<td>RedCrdBonds</td>
<td>No</td>
<td>Redundant Coordinates used in the construction of the initial – force field derived – Hessian</td>
</tr>
<tr>
<td>RedCrdH</td>
<td>No</td>
<td>atom-atom bonds determined for the construction of the initial Hessian</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Hessian in the redundant coordinates representation</td>
</tr>
<tr>
<td>Key</td>
<td>Default</td>
<td>Description</td>
</tr>
<tr>
<td>---------</td>
<td>---------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>SCF</td>
<td>Yes</td>
<td>Controls the information about progress of the SCF procedure. Applies only if</td>
</tr>
<tr>
<td></td>
<td></td>
<td>the print switch COMPUTATION is on.</td>
</tr>
<tr>
<td>SDIIS</td>
<td>No</td>
<td>Expansion coefficients applied by the DIIS procedure during the SCF.</td>
</tr>
<tr>
<td>SDIISMAT</td>
<td>No</td>
<td>Turns on SDIIS (see above) <em>and</em> prints the <em>error vector</em> constructed by</td>
</tr>
<tr>
<td></td>
<td></td>
<td>the DIIS routine (this is the commutator of the Fock matrix and the Density</td>
</tr>
<tr>
<td></td>
<td></td>
<td>matrix). This is used to determine the DIIS expansion coefficients and to</td>
</tr>
<tr>
<td></td>
<td></td>
<td>assess convergence.</td>
</tr>
<tr>
<td>SFO</td>
<td>Yes</td>
<td>General control of SFO-related output. If turned off, (almost) all such</td>
</tr>
<tr>
<td></td>
<td></td>
<td>output is suppressed. If on, as is the case by default, such printing is</td>
</tr>
<tr>
<td></td>
<td></td>
<td>controlled by the EPRINT subkey SFO.</td>
</tr>
<tr>
<td>Smat</td>
<td>No</td>
<td>Overlap matrix of BAS functions.</td>
</tr>
<tr>
<td>Smearq</td>
<td>No</td>
<td>Smear parameter – if and when applied – used in the determination of</td>
</tr>
<tr>
<td></td>
<td></td>
<td>electronic occupation numbers for the MOs, with details of how it works</td>
</tr>
<tr>
<td></td>
<td></td>
<td>out at every cycle of the SCF. For debugging purposes.</td>
</tr>
<tr>
<td>SpinOrbit</td>
<td>No</td>
<td>Detailed information about how double-group symmetry representations are</td>
</tr>
<tr>
<td></td>
<td></td>
<td>related to the single group representations.</td>
</tr>
<tr>
<td>Tails</td>
<td>No</td>
<td>In each block of integration points (see Blocks) the evaluation of (Slater-</td>
</tr>
<tr>
<td></td>
<td></td>
<td>type) exponential functions (basis, fit) is skipped when the function has</td>
</tr>
<tr>
<td></td>
<td></td>
<td>become negligible for all points in that block due to the distance of those</td>
</tr>
<tr>
<td></td>
<td></td>
<td>points from the atom where the function is centered. The relative savings</td>
</tr>
<tr>
<td></td>
<td></td>
<td>due to this distance screening is printed at the first geometry cycle (use</td>
</tr>
<tr>
<td></td>
<td></td>
<td>DEBUG for printing at all cycles).</td>
</tr>
<tr>
<td>TechPar</td>
<td>Yes</td>
<td>Technical parameters such as maximum vector length in vectorized numerical</td>
</tr>
<tr>
<td></td>
<td></td>
<td>integration loops, SCF and Geometry Optimization strategy parameters.</td>
</tr>
<tr>
<td>Timing</td>
<td>No</td>
<td>Print out of more timing info (in particular with respect to performance of</td>
</tr>
<tr>
<td></td>
<td></td>
<td>the parallel version of ADF) than is provided by the standard Timing</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Statistics tables at the end of each output.</td>
</tr>
<tr>
<td>TimingDetail</td>
<td>No</td>
<td>Similar, but more details.</td>
</tr>
<tr>
<td>TimingTooMuchDetail</td>
<td>No</td>
<td>Similar, but even worse.</td>
</tr>
<tr>
<td>Workspace</td>
<td>No</td>
<td>Statistics of calls to the Workspace Manager (memory management).</td>
</tr>
</tbody>
</table>

*Arguments for the keys PRINT and NOPRINT.*

For print switches that start with Frag., Fit., Freq., Geostep., Numint., Repeat., SCF., TF., see the key EPRINT below.
**DEBUG**

The key `DEBUG` is used to generate extensive output that is usually only relevant for debugging purposes. It operates exactly like the `PRINT` key but there is no converse: `NODEBUG` is not recognized; it would be irrelevant anyway because by default all `DEBUG` print switches are off.

A list of the possible items for the `DEBUG` key is given below. All items of the `PRINT` list can also be used with the `DEBUG` key. If they are not mentioned in table III, the meaning is the same as for the `PRINT` key, but the corresponding output may be generated more often, for instance at every SCF cycle rather than at the last one only.

<table>
<thead>
<tr>
<th>Item</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Basis</td>
<td>Construction of the orthonormal LOW basis from elementary (BAS) and fragment (FO) basis.</td>
</tr>
<tr>
<td>Core</td>
<td>Core Orthogonalization procedure</td>
</tr>
<tr>
<td>Ekin</td>
<td>Kinetic energy matrices. (compare the print switches EKIN)</td>
</tr>
<tr>
<td>Fit</td>
<td>Construction of the symmetry adapted fit functions</td>
</tr>
<tr>
<td>Fitint</td>
<td>Construction of integrals used in the Fit procedure.</td>
</tr>
<tr>
<td>Freq</td>
<td>Force matrices processed in the computation of frequencies: Cartesian and internal representation, before and after symmetrization, etc. (as far as applicable).</td>
</tr>
<tr>
<td>GeoStep</td>
<td>Geometry optimization procedure. All relevant items.</td>
</tr>
<tr>
<td>Hess</td>
<td>Complete eigensystem of the Hessian during geometry optimizations.</td>
</tr>
<tr>
<td>NumInt</td>
<td>Numerical integration. Very extensive output (including the coordinates and weights of all generated points).</td>
</tr>
<tr>
<td>Pmat</td>
<td>P-matrix (density matrix) during SCF and in the ETS analysis program in the BAS representation.</td>
</tr>
<tr>
<td>Rhofih</td>
<td>Computation of fit coefficients during the SCF.</td>
</tr>
<tr>
<td>SCF</td>
<td>Extensive output during the SCF procedure about many different items. See also <code>EPRINT</code>, subkey <code>SCG</code>.</td>
</tr>
<tr>
<td>SDIIS</td>
<td>All data concerning the DIIS as used during the SCF. See <code>EPRINT</code>, subkey <code>SDIIS</code>.</td>
</tr>
<tr>
<td>TransitionField</td>
<td>The Transition State procedure to compute and analyze certain terms in the bonding energy. The distinct components, the involved transition field Fock matrices, etc.</td>
</tr>
</tbody>
</table>

Table III. Arguments for the print key DEBUG. All debug switches are by default off.

**EPRINT**

The key `EPRINT` is an extended version of the `(NO)PRINT` key, employed for print switches that require more specification than just `off` or `on`.
Contrary to what is the case for the keys PRINT and NOPRINT, the key EPRINT must occur only once in the input file; any subsequent occurrences are incorrect and ignored or lead to abort.

```
EPRINT
SUBKEY
SUBKEY
......
END
```

**subkey**

A subkey-type structure: it consists of a keyword followed by data, so that it functions as a simple (sub)key, or it is a keyword followed by a data block which must then end with the word `SUBEND`.

The subkeys used in the EPRINT data block are called Eprint keys. A complete list of them is given below. All available EPRINT keys are discussed in the schemes below. The enclosing records EPRINT and END are omitted in these schemes.

<table>
<thead>
<tr>
<th>EPRINT subkeys</th>
<th>Subject</th>
</tr>
</thead>
<tbody>
<tr>
<td>AtomPop</td>
<td>Mulliken population analysis on a per-atom basis</td>
</tr>
<tr>
<td>BASPop</td>
<td>Mulliken population analysis on a per-BAS-function basis</td>
</tr>
<tr>
<td>Eigval</td>
<td>One-electron orbital energies</td>
</tr>
<tr>
<td>Fit</td>
<td>Fit functions and fit coefficients</td>
</tr>
<tr>
<td>Frag</td>
<td>Building of the molecule from fragments</td>
</tr>
<tr>
<td>FragPOP</td>
<td>Mulliken population analysis on a per fragment basis</td>
</tr>
<tr>
<td>Freq</td>
<td>Intermediate results in the computation of frequencies (see DEBUG: FREQ)</td>
</tr>
<tr>
<td>GeoStep</td>
<td>Geometry updates (Optimization, Transition State, ...)</td>
</tr>
<tr>
<td>NumInt</td>
<td>Numerical Integration</td>
</tr>
<tr>
<td>OrbPop</td>
<td>(Mulliken type) population analysis for individual MOs, both on a per-SFO basis and on a per-BAS function basis. In a SpinOrbit calculation no SFO-type analysis is available (not yet implemented).</td>
</tr>
<tr>
<td>Repeat</td>
<td>repetition of output in Geometry iterations (SCF, optimization, ...)</td>
</tr>
<tr>
<td>SCF</td>
<td>Self Consistent Field procedure</td>
</tr>
<tr>
<td>SFO</td>
<td>Information related to the Symmetrized Fragment Orbitals and the analysis (populations and MO coefficients) in this representation.</td>
</tr>
<tr>
<td>TF</td>
<td>Transition Field method for the evaluation and analysis of certain bonding energy terms.</td>
</tr>
</tbody>
</table>

*Table IV. List of EPRINT subkeys.*
**EPRINT subkeys vs. PRINT switches**

Several EPRINT subkeys are merely shortcuts for normal (NO) PRINT switches. All such simple subkeys are used in the following way:

```
| ESUBKEY argumentlist
```

**ESUBKEY**

One of the following EPRINT subkeys: Fit, Frag, GeoStep, NumInt, Repeat, SCF, SDIIS, SFO, TF, Time.

**argumentlist**

A sequence of names, separated by delimiters. Each of these names will be concatenated with the ESUBKEY and the combination will be stored as a normal PRINT switch.

Example:
```
Frag rot, SFO
```

will be concatenated to FRAGROT and FRAGSFO and both will be stored as PRINT switches. All such combinations can also be specified directly with the key PRINT. The example is therefore exactly equivalent with the input specification:
```
PRINT FragRot, FragSFO
```

If any of the names starts with the two characters NO, the remainder of the name will be concatenated with the EPRINT, but now the result will be stored and treated as a NOPRINT switch. Items that are on by default can in this way be turned off. Example:

```
| EPRINT
| FRAG noRot Eig
| END
```

This turns Rot off and Eig on for the EPRINT subkey FRAG. Equivalent would be:

```
| NOPRINT FragRot
| PRINT FragEig
```

Follows a description of all simple EPRINT subkeys:

**Fit**

The subkey FIT controls output of how the elementary fit functions are combined into the symmetric (A1) fit functions. It controls also printing of the initial (start-up) and the final (SCF) fit coefficients.

```
| FIT list
```
list
  A list of items, separated by blanks or commas. The following items are recognized: Charge, Coef, Comb.
Charge
  The amount of electronic charge contained in the fit (start-up), total and per fragment.
Coef
  The fit coefficients that give the expansion of the charge density in the elementary fit functions.
Comb
  The construction of the totally symmetric (A1) fit function combinations from the elementary fit functions.
  By default all options are off.

Freq

list
  A list of items, separated by blanks or commas. The following items are recognized: Eig, Fit, Rot, sfo.
Eig
  The expansion coefficients in elementary functions (BAS) of the fragment Molecular Orbitals as they are on the fragment file.
Rot
  The rotation (and translation) required to map the master fragment (i.e. the geometrical data on the fragment file) onto the actual fragment which is part of the current molecule.
  N.B.: if eig and rot are both on, the rotated fragment orbitals are printed also.
Fit
  The fit coefficients that describe the fitted charge density of the fragments after the rotation from the master fragment on file to the actual fragment. These are the molecular fit coefficients that are used (by default) to construct the total molecular start-up (fitted) charge density and hence the initial Coulomb and XC potential derived from it.
sfo
  The Symmetry-adapted combinations of Fragment Orbitals that are used in the current calculation. This feature ensures that the definition of the SFOs is printed. This will happen anyway whenever the EPRINT subkey SFO itself is activated.
  By default all options are off.
  Remark: SFO analysis in a Spin-Orbit relativistic calculation is not yet implemented.

Freg

Controls printing of Force matrices and a few more data that are intermediate results in the computation of frequencies after all coordinate displacements have been carried out.
| FREQ | list |

list contains any of the items SymCoord, DMuRot, Hess.

SymCoord
print the Symmetry Coordinates, both as they are generated, and some related info in their processing later on. The Symmetry Coordinates are symmetry-adapted combinations of cartesian displacements, with the pure translations and rotations projected out.

DMuRot
info about generating Dipole-Derivative information in transformation between cartesian and internal coordinate representation as regards the rotational aspects.

Hess
processing of the completed matrix of force constants, symmetrization, transformation to other coordinates. By default all options are off.

GeoStep
Controls output concerning the geometry update method, parameters, energy gradients, etc. It plays no role in a SinglePoint calculation.

| GEOSTEP | list |

list A list of items, separated by blanks or commas. The following items are recognized: Energy, GradientTerms, Gradients, Upd.

Energy
summary of the (bonding) energy and its components as computed in the geometry update procedure.

Gradients
Energy gradients on the free variables. These may be all or some of the cartesian or the Z-matrix coordinates, depending on the case.

GradientTerms
The decomposition of the gradients in computed terms, as described in the thesis of L. Versluis [7].

Upd
parameters used and adapted in the geometry update procedure. By default Gradients, Upd are on, the other items off.

NumInt
Output related to the numerical integration procedure: parameters, generated points, tests on the accuracy of the generated scheme, etc.
NUMINT

list

A list of items, separated by blanks or commas. The following items are recognized: All, Geo, Ovl, Par, Pnt, Res, Sym, Test.

All
inicates all other options and prints in addition the coordinates and weights of all generated points. This can be a lot of output!

Geo
geometric data such as boundary planes around the molecule, as they are computed and used in the program section where the point grid is generated.

Ovl
numerically integrated are the auto-overlaps of symmetry-adapted combinations of elementary basis functions SBAS. The deviations from the analytically computed values is printed. The test option, see below, yields a summary of these data: the maximum error and the root-mean-square error.

Par
employed precision parameters, atomic spheres radii etc.

Pnt
the generated numbers of points in each of the subregions processed in the point-generating procedure.

Res
results as regards the total number of points, the sum-of-weights and the partitioning of the points in blocks (for segmented vectorization).

Sym
the symmetry operators that are computed directly from the coordinates (irrespective of the input Schönfliess symbol) and that are used to construct the numerical integration grid in a symmetric fashion.

Test
a few external tests are performed after the grid has been generated, such as the numerical integration of the sum-of-fragment densities. See also the norms option.

By default Res and Test are on, the other options off.

Repeat

Control the repetition of output in Geometry iterations: optimization, computation of frequencies, transition state search.

REPEAT

list

contains one or more of the following items: NumInt, SCF.

NumInt
Output from the numerical integration procedure, like parameters, numbers of points generated, test data is controlled by the NUMINT subkey (see below). The REPEAT subkey controls whether the output is repeated for
all geometries (if the flag is on) or only for the first (if the flag is off). Some concise info is produced
(repeatedly) anyway if the PRINT switch COMPUTATION is on.

scf
Controls similarly the SCF output, like population analysis and orbital eigenvalues. If the flag is on, these
items are printed at the last SCF cycle in every geometry, otherwise only at the last (in case of an optimization,
not in case of a Frequencies calculation).
By default both options are off.

SCF
Output during the SCF procedure.

| SCF list

list
is a list of items, separated by blanks or commas. The following items are recognized: Eigval, Eigvec,
Err, Fmat, Keeporb, MOPop, Occ, Pmat, Pop, Start.

Eigval
Eigenvalues of the one-electron orbitals at the last SCF cycle. In a run with multiple SCF runs (Geometry
Optimization,...) this printing occurs only for the last SCF procedure. See also the EIGVAL subkey of EPRINT.
(Use DEBUG or the REPEAT subkey of EPRINT to get output on all cycles).

Eigvec
MO eigenvector coefficients in the BAS representation. Only printed on the last SCF cycle.

Err
SCF error data which are checked for convergence. By default this takes effect after cycle 25 of the SCF. If the
key is set it takes effect at the first cycle. Optionally one may type ErrN, where N is an integer (written
directly after Err without a blank in between), in which case the key takes effect at cycle N.

Fmat
Fock matrix in the LOW representation.

Keeporb
If the KeepOrbitals option is activated (see the key SCF), output is generated whenever this option actually
results in a change of occupation numbers as regards the energy ordering.

Occ
concise output of SCF occupation numbers on last SCF cycle if no eigenvalues are printed (see: Eigval).

MOPop
Mulliken populations in terms of the elementary basis functions (BAS), per MO, for input-specified MOs (see
the EPRINT subkey ORBPOP)

Pmat
Density matrix

Pop
General control of BAS Mulliken populations. This supervises all printing (whether populations are printed or
not) according to the EPRINT subkeys ATOMP0P, FRAGPOP, ORBPOP (the latter only as regards the BAS
population analysis at the end of the SCF procedure).
Start

Data pertaining to the first SCF cycle (of the first SCF procedure, in case of an optimization; use REPEAT to get this for all SCFs).

By default Eigval, Keeporb, Occ, and Pop are on, the others off.

**SFO**

Information pertaining to the use of Symmetrized Fragment Orbitals (for analysis purposes).

| SFO list |

list

A list of items, separated by blanks or commas. The following items are recognized: **EIG, EIGCF, ORBPOP, GROSSPOP, FRAGPOP, OVL**.

Eig

The MO coefficients in terms of the SFOs.

EigCF

idem, but now also containing the coefficients pertaining to the CoreFunctions.

OrbPop

population analysis of individual orbitals. The orbitals analyzed are set with the **EPRINT subkey ORBPOP**.

GrossPop

Gross populations of the SFOs, split out in symmetry representations. **GrossPop** is automatically turned on when OrbPop is activated.

FragPop

Population analysis on a per-FragmentType basis. This analysis does in fact not depend on the SFOs (ie, the result does not depend on how the SFOs are defined), but the computation of these populations takes place in the SFO-analysis module, which is why it is controlled by the SFO print option. **FragPop** output is given per orbital when OrbPop is activated, per symmetry representation when GrossPop is activated, and as a sum-over-all-orbitals-in-all-irreps otherwise (if FragPop is active).

Ovl

Overlap matrix of the SFO basis, separately for each symmetry representation.

By default **EIG, ORBPOP, OVL** are on, the other options off.

In a Spin-Orbit calculation the SFO analysis is not yet implemented completely.

*Remark*: the options **EIG, EIGCF** replace the previous (now disabled) simple **PRINT options EIGSFO and EIGSFO**.

Note that the simple **PRINT key SFO** controls whether or not the **EPRINT subkey SFO** is effective at all.
**TransitionField**

Part of the bonding energy is computed and analyzed by the so-called Transition State procedure [3, 75]. This has nothing to do with physical transition states, but is related to the Fock operator defined by an average charge density, where the average is taken of the initial (sum-of-orthogonalized-fragments) and the final (SCF) charge density. There is also an analogous term where the average is taken of the sum-of-fragments and the sum-of-orthogonalized-fragments. Various terms, Fock operators and Density Matrices used in this approach may be printed. To avoid confusion with real Transition States (saddle points in the molecular Energy surface) the phrase TransitionField is used here.

| TF List |

List

A list of items, separated by blanks or commas. The following items are recognized: Energy, Fmat, DiagFmat, FragPmat, DiagFragPmat, F*dPmat, DiagF*dPmat, OrbE.

**Energy**

Energy terms computed from the TransitionField.

**Fmat**

TransitionField Fock matrices.

**DiagFmat**

Idem, but only the diagonal elements.

**FragPmat**

The molecular P-matrix constructed from the sum-of-fragments.

**DiagFragPmat**

Idem, but only the diagonal elements.

**F*dPmat**

The TransitionField energy term can be expressed as a Fock operator times the difference between two P-matrices (initial and final density).

**DiagF*dPmat**

only diagonal elements

**OrbE**

Orbital energies in the TransitionField.

By default all options are off.

**Other EPRINT subkeys**

We discuss now the remaining EPRINT sub keys that are not simple shortcuts for PRINT switches.

**Orbital Energies**

| EIGVAL   noocup nvirtual |
This specifies the number of one-electron orbitals for which in the SCF procedure energies and occupation numbers are printed whenever such data is output: the highest nocup occupied orbitals and the lowest nvirtual empty orbitals. Default values are nocup=10, nvirtual=10. If only one integer is specified it is taken as the nocup value and nvirtual is assumed to retain its standard value (10). Printing can be turned off completely with the EPRINT sub key SCF, see above.

**Mulliken Population Analysis**

All population subkeys of EPRINT refer to Mulliken type populations.

- **ATOMPOP** level

Populations accumulated per atom. level must be none, gross or matrix. none completely suppresses printing of the populations; gross yields the gross populations; matrix produces the complete matrix of net and overlap populations. Default value: matrix.

- **BASPOP** level

Populations are printed per elementary (BAS) basis function. The level options are none, short, gross, matrix. none, gross and matrix are as for ATOMPOP. short yields a summary of BAS gross populations accumulated per angular momentum (l) value and per atom. Default value: gross.

- **FRAGPOP** level

Completely similar to the ATOMPOP case, but now the populations per fragment. Of course in the case of single-atom fragments this is the same as ATOMPOP and only one of them is printed. Default: matrix.

For all three population keys ATOMPOP, FRAGPOP and BASPOP, specification of a higher level implies that the lower-level data, which are in general summaries of the more detailed higher level options, are also printed.

Printing of any populations at the end of the SCF procedure is controlled with the EPRINT sub key SCF (POP).

**Population Analysis per MO**

A very detailed population analysis tool is available: the populations per orbital (MO). The printed values are independent of the occupation numbers of the MOs, so they are not populations in a strict sense. The actual populations are obtained by multiplying the results with the orbital occupations.

The analysis is given in terms of the SFOs and provides a very useful characterization of the MOs at the end of the calculation, after any geometry optimization has finished. This feature is not available (yet) in a Spin-Orbit relativistic calculation.
The same analysis is optionally (see EPRINT subkey $SCF$, option $MOPOP$ also provided in terms of the elementary basis functions (BAS).

```
ORBPOP {noccup {nvirtual}} {tol=tol}
  subspecies orbitals
  subspecies orbitals
  ....
SUBEND
```

**noccup**
- Determines how many of the highest occupied orbitals are analyzed in each irrep. Default noccup=10.

**nvirtual**
- Determines in similar fashion how many of the lowest virtual orbitals are analyzed in each irrep. Default nvirtual=4.

**tol**
- Tolerance parameter. Output of SFO contributions smaller than this tolerance may be suppressed. Default: 1e-2.

**subspecies**
- One of the subspecies of the molecular symmetry group.

**orbitals**
- A list of integers denoting the valence orbitals (in energy ordering) in this subspecies that you want to analyze. This overrules the noccup,nvirtual specification for that symmetry representation. In an unrestricted calculation two sequences of integers must be supplied, separated by a double slash ('//').

Any subset of the subspecies can be specified; it is not necessary to use all of them. No subspecies must occur more than once in the data block.

A total SFO gross populations analysis (from a summation over the occupied MOs) and an SFO population analysis per fragment type are preformed unless all MO SFO-populations are suppressed.

**Reduction of output**

One of the strong points of ADF is the analysis in terms of fragments and fragment orbitals (SFOs) that the program provides. This aspect causes a lot of output to be produced, in particular as regards information that pertains to the SFOs. Furthermore, during the SCF and, if applicable, geometry optimizations, quite a bit of output is produced that has relevance merely to check progress of the computation and to understand the causes for failure when such might happen.

If you dislike the standard amount of output you may benefit from the following suggestions:
- If you are not interested in info about progress of the computation:
  ```
  noprint Computation
  ```
- If you'd like to suppress only the SCF-related part of the computational report and make the GeometryUpdates related part more concise:
noprint scf, geo

(Keep COMPUTATION on, so you get at least some info about the GeometryUpdates)

If you don't want to see any SFO stuff:

noprint sfo

To keep the SFO definitions (in an early part of output) but suppress the SFO-MO coefficients and the SFO overlap matrix:

eprint
sfo noeig, noovl
end

Note: the SFO-overlap matrix is relevant only when you have the SFO-MO coefficients: the overlap info is needed then to interpret the bonding/anti-bonding nature of the various SFO components in an MO.

If you are not interested in the SFO populations:

eprint
sfo noorbpop
end

### ASCII Output Files with Atomic Coordinates

You may want to have a special result file that contains the atomic coordinates corresponding to all the geometries processed in the calculation, for instance to feed it to a ‘movie’ generator to display the development of an optimization run. This is regulated with the key FILE:

```plaintext
FILE filetype filename  { filetype2 filename2 }
```

**filetype**
- Specifies the format of the output. Currently supported are three varieties: MOPAC, MOL and XYZ

**filename**
- The file to which the output is written; the file should not yet exist. The name may include a full or relative path with respect to the directory where the calculation runs.

The same input record may contain any number of pairs-of-arguments, for instance to specify that both a mol-type and a xyz-type result file are to be generated. The key may also occur more than once in the input stream, in which case the argument lists are effectively all concatenated (by the program).

### 2.3 MORE OPTIONS

We continue with keys and other aspects of input that are less important in most applications. A few keys have been mentioned already before but allow additional or alternative usage to be discussed now.

Some keys may easily be misused yielding ridiculous results, possibly without any warning or relevant message to this effect from the program. This applies in particular, but not exclusively, to the sections Precision and Control of Program Flow.
General

Link-in Input files

Part of the input file can be put into a separate ASCII file, which can be addressed from the (standard) input stream:

```
INLINE  inlinefile
```

`inlinefile` must be the name of the auxiliary ASCII file (including its path, absolute or relative to the run-directory). When `INLINE` is encountered in the input file, ADF opens the specified file and continues reading from that file as if it were in-line expanded into the input file. When the end-of-file is encountered reading resumes from the original file.

The contents of the `inlinefile` must not end with `END INPUT`, unless you wish to terminate all input reading at that point.

`INLINE` may occur any number of times in the input file. Use of `INLINE` may also be nested (up to 10 levels): the key `INLINE` may be used in the `inlinefile` in the same fashion as in the standard input file.

The `INLINE` feature makes it easy to pack your preferred settings that are not matched by the program’s defaults in one file and use them in every run with a minimum of input-typing effort. Obvious applications are output control (`PRINT`) settings and precision parameters.

Note: you can not use `INLINE` to store `PARALLEL` settings, not even by using `INLINE` on the first line of your input and placing the `PARALLEL` keyword on the first line of the `inlinefile`: before opening the `inlinefile` and expanding it into the inputfile, the program has already detected that the first line of input does not specify the `PARALLEL` settings.

Title and Comment

```
TITLE  Title
```

`Title` may be any string. The program combines it (that is, the first approximately 50 characters) with date and time of the job to construct the `job identification`. The job identification is used to stamp an identification on result files, which will be read and printed if such a file is used again, for instance as a fragment file. The job identification will also be echoed in the output header to identify the current run. By default the date and time are combined with a dummy string. In Create mode the title is first read from the data file that supplies the basis functions etc and can then be overwritten via input.

Note that, contrary to some other programs, ADF does not take the first input record as a title. Typing your title as the first record, without starting the record with the keyword `TITLE`, may produce very strange results: ADF will try to interpret the first word on that line as a keyword, possibly abbreviated!
You can put more remarks in the input file to be echoed in the standard output file; these will not become part of the job identification:

```
COMMENT
text
...
END
```

The `text` records are copied to the output header, directly after the job identification. Expressions are not parsed and constants or functions are not replaced: it is a straightforward copy. The key `COMMENT` may occur any number of times; all text blocks are printed in the output header with a blank line between any two text blocks.

**Layout of input**

Empty records and leading blanks in records are allowed and ignored, and can be used to enhance clarity and readability of the input file for human readers.

An exclamation mark (!) is interpreted by the input reading routine as denoting the end-of-line. Instead of the exclamation mark you may also use a double colon (::). The part of the line after the exclamation mark (double colon) – including the ! or :: itself – is ignored. In this way one can include comments and clarifying remarks, which will not be echoed in the output header (compare the key `COMMENT`).

**Geometry**

**Orientation of Local Atomic Coordinates**

As discussed before the atomic positions are input with the key `ATOMS`. One option has thus far not been mentioned: the possibility to redefine the local coordinate frame of an atom.

```
ATOMS  {type of coordinates}
{n} atomname  coordinates  {F=fragment}  {Z=xx  yy  zz}
...
END
```

Except for the `z=` option all aspects have been examined already before.

```
z=xx  yy  zz
```

defines a reorientation of the local atomic z-axis; it is interpreted as a direction vector with components (xx,yy,zz) pointing away from the atom. In the local, reoriented frame the local atomic x-axis will be rotated to the plane defined by the directions of the molecular z-axis and the local atomic z-axis.
This feature can be used only for single-atom fragments (otherwise it is ignored). Its purpose is to give more flexibility in the analysis of the final molecular orbitals in terms of the atomic orbitals. In such a case it may be very helpful to redefine the orientation of say the \( p_z \)-orbitals of an atom. For instance, you may orient all \( p_z \)-orbitals towards the origin by specifying for each atom \( z = -x - y - z \) (with \( x,y,z \) the coordinates of that atom).

By default the local and molecular z-axes are identical.

**Symmetry**

Together with the point group symmetry, a tolerance parameter can be supplied.

\[
\begin{align*}
\text{SYMMETRY} & \quad \{\text{symbol}\} \quad \{\text{TOL}=\text{tolerance}\}
\end{align*}
\]

symbol

The Schönsfliess symmetry symbol. A complete list of allowed values for this argument is given in Appendix 3.

tolerance

The tolerance (absolute deviation in the Cartesian coordinates) for atomic positions being symmetry equivalent. The same tolerance applies to check the mapping of fragments on attached fragment files with the actual fragments.

If the tolerance is specified it is interpreted in the chosen unit of length (\textsc{units}). The default tolerance is 0.001Å.

Input atomic coordinates that are slightly (within the tolerance) off from their correct positions are adjusted by the program.

**Ghost Atoms & Non-standard Chemical Elements**

The atom type names used under \textsc{atoms} (and in the \textsc{create} record) must begin with the standard chemical element symbol (H, He, Li...). The program uses this to deduce the nuclear charge and other elemental properties.

For the standard elements one can redefine the atomic mass (for instance to define a suitable isotope).

\[
\begin{align*}
\text{CREATE} & \quad \text{H} \quad \{m=value\} \quad \text{datafile}
\end{align*}
\]

value

The atomic mass, which will then override the default value for the indicated chemical element.

A more extensive feature is available to define an artificial chemical element with user-specified properties. Such new elements are denoted \textit{Alternative Elements}; and may for instance have a non-integer nuclear charge. The chemical symbol of an Alternative Elements is Gh (for ghost) or J: either one is OK.

You can create J-type or Gh-type basic atoms and use them subsequently as fragments in a molecule.
The nuclear charge of an Alternative Element is not pre-defined, and must therefore be specified in the Create run. The atomic mass is optionally supplied.

```plaintext
CREATE J.NewElement q=Q (m=mass) datafile
```

**J.NewElement**

The atom type name, beginning with the alternative chemical element symbol J. It has an (optional) suffix of the form `.text`, completely similar to the construction of atom type names from standard chemical element symbols.

**Q**

The nuclear charge. The q= option must be used for a J-element.

It must not be used for standard chemical elements.

**mass**

The atomic mass, in atomic mass units. If not supplied it will be set to the atomic mass of the standard chemical element with nuclear charge A, where A equals Q rounded to the nearest integer, but not smaller than 1 and not larger than 103.

**datafile**

The Create data file.

If you want to use the Alternative Element feature you'll have to construct your own Create data file, suited to the Alternative Element you have in mind. Appendix 1 describes the format of such a file.

**Use as fragment**

J-type basic atoms can be used like any other basic atoms to build up larger fragments and molecules. In fact, J (or Gh) can be considered just one more chemical symbol along with the 103 traditional ones. The element J has no pre-defined properties. Therefore you have to specify them where appropriate (c.f. the nuclear charge and atomic mass).

You may have different J-elements in a molecule, with different nuclear charges for instance. Yet, they must be denoted with the same chemical symbol J; the difference can only be made clear by the `.text` suffix in the atom type name.

It does no harm of course to make this suffix a concise but clear description of the main characteristics.

**Basis Set Superposition Error (BSSE)**

The Ghost Atom feature enables the calculation of Basis Set Superposition Errors (BSSE). The idea is as follows. In a normal calculation of the bonding energy of a molecule C, composed of fragments A and B, one compares the total energies of C vs. those of isolated A and isolated B added together. In ADF this can be done in one stroke by running C from fragments A and B.

The BSSE is determined as the bonding energies of a pseudo-molecule D composed of (1) A plus a ghost B and (2) B plus a ghost A. The ghost atoms in the calculations are at their normal positions in the true molecule C, and
they have their normal basis (and fit) functions. However, they do not have a nuclear charge and no electrons to contribute to the molecule. To set such a calculation up one needs first to make the appropriate ghost database files: for each involved atom, copy the database file that was used for its creation and modify it so as to remove the frozen core. Next, Create the ghosts with zero mass and zero nuclear charge. Apply these ghost fragments in the BSSE runs.

An example is worked out in the EXAMPLES document.

**Hamiltonian**

*Spin-polarized start-up potential*

The Coulomb and $XC$ (exchange + correlation) potentials are computed from the fit approximation of the charge density (see Chapter 1.2).

The fit coefficients of this approximation for the first SCF cycle, needed to compute the first Fock matrix, are read from the fragment files: the start-up density is chosen as a sum-of-fragment-densities (fit approximations) and this combined density defines the initial potential.

In the SCF restart run the fit coefficients may be read from the attached TAPE21 file, see the key `RESTART`.

In some applications you may want to modify the initial fit coefficients (from the restart file or the fragment files). This is achieved with the key `MODIFYSTARTPOTENTIAL`. It allows you to scale them in some way so as to represent user-chosen amounts of spin-$\alpha$ and spin-$\beta$ fit density on some or all of the fragments. This will adjust the spin-$\alpha$ and spin-$\beta$ initial potentials.

This option applies only to *unrestricted* calculations of course. It may be used to help the program find a particular state. This might, for instance, be hard to find otherwise due to the $\alpha$–$\beta$ symmetry in the start-up situation. It may also be useful to speed up the SCF convergence in case you know what the final distribution of spin-$\alpha$ and spin-$\beta$ density over the molecule will approximately be.

```
MODIFYSTARTPOTENTIAL {specification}
   { frag alfa // beta
     frag alfa // beta
   ....
   END }
```

A general key: it has an argument or a data block.

**specification**

Must be two numbers, $\text{ASPIN}$ and $\text{BSPIN}$, if provided at all. They specify the (relative) amounts of spin-$\alpha$ and spin-$\beta$ fit density to define the spin-dependent potential at the first SCF cycle. The coefficients retrieved from the fragment files (or from the restart file in case of a SCF restart) are scaled accordingly. This will not affect the total amount of fit density: the absolute values of $\text{ASPIN}$ and $\text{BSPIN}$ play no role, only their ratio.
In case of a restart run the restart file must have been generated in a restricted calculation, while the continuation run must be an unrestricted one.

If no argument is given a data block must be supplied with records \texttt{frag alpha // beta}.

This is very much similar to the main option with \texttt{ASPIN and BSPIN}; you specify \texttt{ASPIN} and \texttt{BSPIN} now separately for each fragment. This involves somewhat more input but increases the possibilities to tune the initial potential. Again this can be applied only in an unrestricted calculation. It cannot be used in a restart: the affected fit coefficients are those from the fragment files, while in an SCF restart run these are ignored and replaced by the coefficients on the TAPE21 restart file.

Each line specifies a \texttt{frag} with its corresponding \texttt{ASPIN} and \texttt{BSPIN} fit partitioning. If \texttt{frag} is the name of a fragment type, the specified \texttt{ASPIN-BSPIN} is applied to all individual fragments of that type. Alternatively an individual fragment can be specified, using the format \texttt{fragtype/n}, where \texttt{n} is an index between one and the total number of fragments of that type. In such a case the \texttt{ASPIN-BSPIN} data applies only to that particular fragment while different values may be supplied for the other fragments of the same type.

It is allowed to specify for certain fragment types individual fragments and for other fragment types only the type. Duplicate specifications are not allowed; an individual fragment must not be specified if its fragment type is also specified as a whole.

If the data block form is used, only the fit coefficients of the referenced fragments are affected. For the not-referenced fragments the fit densities are used as they are defined on the corresponding fragment files.

The SCF convergence of a spin-unrestricted calculation usually improves when you start with potentials that correspond to the correct ratio of spin-$\alpha$ and spin-$\beta$ electrons. By default \texttt{ASPIN=BSPIN=0.5}, as implied by the spin-restricted start density of the fragments or restricted molecule.

The total amount of fit density used on the first iteration is defined by the sum-of-fragment densities (or the density on the restart file). This may be different from the total nr. of electrons in the actual calculation. On the second SCF cycle the fit density will internally be normalized so as to represent the correct number of electrons.

The block-form of the key makes the start up of broken symmetry calculations easy. For example one may want to start a calculation in broken symmetry with spin-$\alpha$ density on one fragment and spin-$\beta$ density on another, e.g. in a spin-unrestricted calculation of H$_2$ at large separation. It is particularly useful for larger systems, e.g. for magnetic coupling between spin-polarized magnetic centers, as in Fe-S complexes [76]: start with oppositely polarized Fe centers, but with, for instance, the remaining bridge and terminal ligands unpolarized. See also the N$_2^+$ sample run in the EXAMPLES.

**Unrestricted fragments**

The fragments from which the molecule is built must be spin-restricted, that is: the fragment files must be result files of spin-restricted calculations. For purposes of analysis, however, it may be desirable in some applications to build your molecule from unrestricted fragments. This can be simulated as follows.

You tell ADF that you want to \textit{treat} the fragments as if they were unrestricted; this causes the program to duplicate the one-electron orbitals of the fragment: one set for spin-$\alpha$ and one set for spin-$\beta$. You can then specify occupation numbers for these spin-unrestricted fragments, and occupy spin-$\alpha$ orbitals differently from spin-$\beta$ orbitals.
Of course, the restricted fragments that you use in this way, are not self-consistent: different numbers of spin-α and spin-β electrons usually result in different spatial orbitals and different energy eigenvalues for spin-α and spin-β when you go to self-consistency, while here you have spatially identical fragment orbitals. Nevertheless it is often a fair approximation which gives you a considerable extension of analysis possibilities.

<table>
<thead>
<tr>
<th>FRAGOCCUPATIONS</th>
</tr>
</thead>
<tbody>
<tr>
<td>FRAGTYPE</td>
</tr>
<tr>
<td>irrep spin-a // spin-b</td>
</tr>
<tr>
<td>irrep spin-a // spin-b</td>
</tr>
<tr>
<td>...</td>
</tr>
<tr>
<td>END</td>
</tr>
</tbody>
</table>

**fragtype**

One of the fragment types and functions as a (block type) subkey. The data block for the subkey ends with the standard end code for block type subkeys (SUBEND).

**irrep**

One of the irreducible representations (irreps) for the point group symmetry that was used in the computation of that fragment.

**spin-α // spin-β**

Two sequences of occupation numbers, which will be applied to the spin-α and spin-β versions of the Fragment Orbitals. The sequences must be separated by a double slash (//). See for comparison the specification of occupation numbers for the overall system (key CHARGE).

The sum of spin-α and spin-β occupations must, for each fragment orbital in each irrep separately, be equal to the total (restricted) occupation of that orbital as it is stored on the fragment file. In other words: you can only change the distribution over spin-α and spin-β electrons within one orbital. (Without this restriction the spatial distribution of the total (sum over spins) fragment charge density would be changed, leading to an incorrect bonding energy analysis after the calculation).

The data block of FRAGOCCUPATIONS is not parsed for expressions and constants or functions defined under DEFINE. Any such items will not be recognized and not be replaced by their values.

Be aware that in more-dimensional irreps (E, T) the number of electrons in a fully occupied orbital is input as the dimension of the irrep times the one-electron orbital occupation. Compare the key OCCUPATIONS.
For irreps that are not mentioned in this input block, and hence for all irreps of fragment(type)s that are not mentioned at all, the spin-\(\alpha\) and spin-\(\beta\) occupations will be set equal, which is of course what they in fact are on the (restricted) fragment file.

For an example of applying this option see [77].

**Remove Fragment Orbitals**

By default all fragment orbitals (the MOs of the fragment computation), which are stored on the fragment file, are used as basis functions for the overall molecule, see Chapter 1.2. You can remove one or more of these fragment orbitals from the basis set of the molecule. This may be useful for special analyses, for instance to study the effect of deleting all virtual MOs of a particular fragment (CSOV analysis). It may also enhance the efficiency since you effectively reduce the size of the basis set, but you should be aware of the potential effects on the results.

```
REMOVEDFRAGORBITALS
  FRAGTYPE
    subspecies nremove
    subspecies nremove
...
  SUBEND
  FRAGTYPE
    subspecies nremove
...
  SUBEND
  ....
  (etc.)
  ....
  END
```

**fragtype**

One of the fragment types in the system. Any subset of the available fragment types can be used here as subkey. The subkeys are block type keys; their data blocks end **SUBEND**.

**subspecies**

One of the subspecies of the irreducible representations of the point group symmetry that was used in the calculation of the fragment itself. This requires of course that one knows the symmetry that has been used for the fragment calculation.

**nremove**

The number of fragment orbitals of the pertaining representation that will not be used as basis functions for the overall system. The highest (in energy eigenvalue) \(n_{\text{remove}}\) orbitals are discarded. You must not remove occupied fragment orbitals.

By default (omission of the key) all fragment orbitals are used in the basis set for the system.
**Important Note**

It is imperative that any removal of fragment orbitals will not break the symmetry of the molecule. This consideration is relevant when for instance two different subspecies of a fragment irrep contribute to different partner subspecies in one of the irreps of the molecule. In such a case, when one removes an orbital in such a fragment subspecies, its partner orbital should also be removed. If this is violated an error may occur or the results will simply be wrong. Quite likely, the program will detect the error, but this may occur only in the final (analysis) stage of the calculation so that a lot of CPU time may have been wasted.

Example: consider a single-atom fragment, computed in ATOM symmetry, used as fragment in a C(lin) molecule and assume that the p:x and p:y fragment orbitals contribute to respectively the pi:x and pi:y subspecies of the molecule. Then, when you remove one or more p:x fragment orbitals, you should also remove the same number of p:y fragment orbitals. Practical cases may be more complicated and whenever you use this key, make sure that you’ve fully analyzed and understood how the fragment irreps combine into the molecular symmetry representations. Hint: run the molecule, without removing any fragment orbitals, and stop at an early stage after the program has computed and printed the build-up of the molecular SFOs from the fragment orbitals. To control early aborts via input, use the key STOPAFTER.

**Core Potentials**

In the standard approach the Coulomb potential and the charge density due to the atomic frozen core are computed from the frozen one-electron orbitals. ADF stores the computed core density and core potential for each atom type in the molecule on a file TAPE12. Alternatively, you may attach a file with (core) potentials and densities. The file must have the same structure as the standard TAPE12. It should contain one or more sections, each with the core information for one type of atom. With the key COREPOTENTIALS you specify the core file and (optionally) which sections pertain to the distinct atom types in the molecule. It is a general key that can be used as a simple key or as a block key.

```plaintext
COREPOTENTIALS corefile { &
atomtype  index
atomtype  index
...
END}
```

**corefile**
The file with core potentials and charge densities. The name may contain a path.

**atomtype**
One of the atom type names as defined by ATOMS.

**index**
Points to the core section on the attached file that applies to the atom type. Different atom types may use the same section. A non-positive index tells the program that the atoms of that type don't have a frozen core. If the information on the corresponding fragment file (or data file in Create mode) indicates the contrary the program will abort with an error message.

If the key is used as a simple key (specifying only the core file) the sections on the file are associated with the atom types in order: the first section is used for the first atom type, et cetera. This is overruled by applying the
block form. However, since the key must have the core file as argument, the block form requires that you apply
the continuation symbol: an ampersand (&), separated from the core file name by at least one blank.
If you omit an atom type from the data block it gets a zero index (no core).

The attached file may contain more sections than used in the calculation, and the indices specified in the data
block don't have to be in ascending order, consecutive, or cover a specific interval.

When a file with NON-STANDARD (e.g. relativistic) cores is attached and used in the calculation of an atom or
molecule, and the result is used as fragment in a subsequent calculation, you should attach and use the same core
potentials again. Otherwise, the program will internally compute the standard core potentials and hence
implicitly employ another fragment than you may think, i.e. a fragment with other properties. ADF will not check
anything in this respect and COREPOTENTIALS should therefore be handled with great care.

The primary application of the COREPOTENTIALS option is to include (scalar) relativistic corrections in the
(frozen core part of the) Fock operator. The relativistic core potentials can be computed with the auxiliary
program dirac (see the UTILITIES document)

Properties and Analysis

NMR Chemical Shifts

NMR Chemical shifts have been implemented [78-82] in a separate utility program NMR. It requires the
TAPE21 result file from an ADF calculation. See the Utilities document for the input description of the NMR
module.

Note: NMR calculations on systems computed with the ZORA relativistic approach are not yet supported.
Note2: NMR calculations on systems computed with Spin-Orbit relativistic effects can only be performed by the
NMR module if the ADF calculation has suppressed usage of symmetry, i.e. when the symmetry used in the
ADF calculation has been NOSYM.

Localized Molecular Orbitals

ADF provides the Boys-Foster method for localization of Molecular Orbitals [83-85]. This implies a unitary
transformation of the occupied molecular orbitals as computed in the SCF procedure, with the objective to obtain
a (transformed) set of orbitals that represent exactly the same charge density but with molecular orbitals that are
more localized in space than the original MOs.
The goal of orbital-localization lies in analysis: the localized orbitals provide an easier-to-interpret picture.

Orbital localization procedures require a measure of the localization of the orbitals which can then be optimized
in the space of the allowed unitary transformations. Methods advocated in the literature differ in the definition of
this measure. The Boys-Foster method minimizes the mean extension of the occupied orbitals around their center of gravity; see the literature for details.

Occasionally it is useful to apply the localization only to a subset of the MOs, with the objective to expose certain features better. This is accomplished by performing the localization in a number of distinct steps, where at each step the localization is restricted by keeping a subset of the MOs frozen. A case is worked out in the Examples document.

The computation of localized orbitals is controlled with the block-type key LOCORB. By default (if the key is not supplied in input) no orbital localization is carried out.

```
LOCORB  {NOPOP}
Spintype FrozenMOs
Spintype FrozenMOs
....
END
```

**NOPOP**

Specifies that no SFO population analysis is to be carried out on the localized MOs. By default this population analysis will be printed in the output file.

**Spintype**

Must be either ALFA or BETA (not case sensitive) and refers to spin-A and spin-B orbitals respectively. In a spin-restricted run BETA records are meaningless and must not be used.

**FrozenMOs**

A list (possibly empty) of integers, referring to a list of MOs from the SCF, and/or labels of irreducible representations. The integers and/or labels may be given in any order.

Each record Spintype FrozenMOs in the data block defines a localization cycle in which the localization procedure is carried out on all orbitals (of the indicated spin), except those indicated by the FrozenMOs. For either spin at least one localization cycle is carried out. If no data record for that spin is found in the data block, a full localization is performed, without any MOs excluded.

The data block may be completely empty (but the record END must be supplied since the key is block-type) and would be equivalent with specifying two records, one for either spin, without any FrozenMOs:

```
LOCORB
END
```

is equivalent with

```
LOCORB
  alfa
  beta
END
```
The integers in *Frozen* MOs refer to an overall list of SCF MOs consisting of all valence MOs in each symmetry representation up to and including the highest non-empty one. So, when for instance in the first irrep MO #4 is the highest non-empty one and in the second irrep MO #2 is the highest non-empty one, then in the overall list the first 4 are the orbitals of the first irrep, the no.s 5 and 6 are from the second irrep, etcetera. Each symmetry label in *Frozen* MOs collectively denotes in one stroke all molecular orbitals of that representation up to and including the highest occupied one (in that symmetry). The label may be the name of an irreducible representation or of a subspecies. In the former case all partner representations are denoted collectively. In an ATOM symmetry for instance, specifying $P$ would be equivalent to $P:x\ P:y\ P:z$.

Note that if the final SCF has in any symmetry representation empty orbitals below the highest non-empty orbital in that symmetry — violating the Aufbau principle — then these empty orbitals are included in the above-defined overall list and hence a *Frozen* MOs specification is necessary, namely to avoid mixing MOs with different occupation numbers in the localization.

Note:
It is imperative that in a particular localization cycle only MOs from the SCF are combined that have identical occupation numbers. If this is violated the program will carry out the localization without error message, but the results are incorrect in the sense that the density defined by the localized orbitals is not the same anymore as the SCF density.

So, if any of the MOs in the overall list defined above is not fully occupied (open shell, excited state, ...) you need to define precisely the localization cycles — localizing in each cycle only MOs with identical occupations and freezing all others — in order to obtain sensible results.

In the output file the localized MOs are printed as expansions in SFOs and (optionally) a population analysis is given, again in terms of the SFOs. Furthermore, each localized MO has associated with it an energy value and an occupation number. The energy is the expectation value of the Fock operator for the orbital. The occupation number is obtained as a weighted sum from the SCF MOs that were combined into the localized orbital. As mentioned before one should combine only SCF MOs with identical occupations into a localized orbital, in which case its occupation number will be the same. The printout of the occupation number of the localized orbital allows therefore a verification that a correct localization procedure has been carried out.

**Precision**

**Numerical integration**

The key INTEGRATION has been introduced in its simple form in Chapter 2.2.
accint is a real number. The key is used as a simple key here. Alternatively you can use it as a block key. This is activated if you give no argument. In the data block you specify which of several integration methods you want to use, and you give values for the involved parameters. Consult the literature for detailed information about the various schemes.

```
INTEGRATION
 data
 data
 ...
END
```

The block form is used to override default relations between various parameters that are applied in the generation of the integration grid in the polyhedron method [70]. All these parameters are accessible with subkeys in the data block of INTEGRATION. Most of the subkeys are simple keys with one single value as argument; a few subkeys are block-type (sub) keys themselves and hence require the usual format of a data block closed by SUBEND.

**accint**

The main precision parameter

Its value defines the number of significant digits by which an internal set of standard integrals must be evaluated. The number and distribution of integration points is tuned accordingly. For normal applications this should yield a nearly optimal (given the underlying method) generation of points and weights. The default depends on the run type.

**accsph**

The polyhedron method of generating integration points partitions space in atomic polyhedrons, partitioned in pyramids with their tops at the atom in the center of the polyhedron. A core like atomic sphere is constructed around the atom; this truncates the tops of the pyramids. **accsph** specifies the test precision for the generation of points within the spheres. By default accsph=accint.

**accpyr**

Similarly this subkey sets the test level for the parts of the pyramids outside the atomic sphere. Default: accpyr=accint.

**accpyu, accpyv, accpyw**

The truncated pyramids are mathematically transformed into unit cubes. A product Gauss integration formula is applied to the cubes, with three (test precision) parameters for the three dimensions. **accpyw** controls the direction that is essentially the radial integration from the surface of the atomic sphere to the base of the pyramid. The other two control the orthogonal directions (angular). By default all three equal accpyr.

**ACCOUT**

The region of space further away from the atoms, outside the polyhedrons, has its own precision parameter. By default accout=accint.

**NOUTER**

This outer region is treated by a product formula: outwards times parallel. The latter involves two dimensions: the surface of the molecule say. The outward integration is performed with Gauss-Legendre quadrature, in a few separate steps. The lengths of the steps are not equal, they increase by constant factors. The total length is fixed. The number of steps is controlled with this subkey; default: 2.
OUTRAD
The parameter that defines the number of Gauss-Legendre integration points for each outward step. The precise relation between the actual number of points and this subkey, and the default relation between outrad and accout can be found in the implementation.

OUTPAR
Similarly the integration in the directions parallel to the surface of the atomic system is controlled by a parameter. See the implementation for details.

DISHUL
Sets the distance between the outermost nuclei of the molecule and the boundary planes that define the boundary between the polyhedrons and the outer region. By default dishul=2.3*R_{max}, where R_{max} is the radius of the largest atomic sphere in the molecule.

FRANG
The outward range of the outer region: integration is not performed to infinity but to a distance frange from the outermost atoms, where all functions can be assumed to be essentially zero. By default frange is derived both from accint, the general precision parameter, and from the present chemical elements: heavier atoms have longer-range functions than hydrogen say. The precise relations can be found in the implementation.

LINROT
This parameter is significant only for symmetries with an axis of infinite rotational symmetry: C_{∞v} and D_{∞h}
It is the highest rotational quantum number around this axis that occurs among the integrands. This depends on the employed basis functions and fit functions. By default the program finds this out for itself.

qpnear
If you specify point charges in the input file, there are two considerations implied for the numerical integration grid.
First, since the point charges create a Coulomb singularity. The integrands (of for instance the basis function products against the Coulomb potential) can only be evaluated with high precision if the grid around the point charges has spherical symmetry and uses local spherical coordinates, exactly as is done for the atomic nuclei.
Second, the point charges do not carry fit or basis functions, hence they play only a role in the more diffuse tails of the actual functions involved in integrals. Therefore, a relative low precision of the integral part close to the point charge may have little effect on the total integration accuracy.
Since additional “spherical centers” with their own surrounding grids increase the total number of points significantly, typically a few thousands per Coulomb center, this may result in high computational effort.
Therefore, the program generates spherical grids only about those point charges that are close to the other atoms. The criterion, input with the qpnear subkey, is the closest distance between the point charge at hand and any real atom. Default 4.0 Angstrom. Any input value is interpreted in the unit-of-length specified with the UNITS key.

Next come the subkeys that require a list of data. The subkey must be placed on one line, the data on the next.
This somewhat peculiar structure suggests that the subkeys are block keys; however their data blocks have no end code (SUBEND) as for normal block type subkeys.
The list of data for such a subkey contains one value for each atom type. The data must be in the order in which the atom types were defined under ATOMS, implicitly or explicitly: remember that atoms belonging to different fragment types automatically have different atom types, even if their atom type names have been specified as identical under ATOMS.
**RSPHER**

gives the radii of the atomic spheres, one value for each atom type. By default, the radii are derived from the chemical element (heavier atoms get larger spheres) and from the environment: the sphere must not be too large for the atomic cell (polyhedron).

**LINTEG**

The maximum angular momentum quantum number of integrands centered on an atom of that type (one value for each atom type). This depends on the basis functions and on the fit functions. By default the program checks the function sets and sets the LINTEG values accordingly. This subkey is applied for the generation of grid points in the atomic spheres.

Items that relate to geometric lengths (dishul, frange, rspher) must be given in bohr (=atomic units), irrespective of the unit of length defined with UNITS.

**Symmetric density fit**

The density fitting procedure in ADF is carried out separately for each pair of atoms. The implemented approach has several advantages in efficiency but it has a drawback in that it necessitates the use of all available fit functions rather than only the symmetric combinations although the final result of course needs only a symmetric fit because the total density is a symmetric (A1) function. For atoms far apart the density fitting is performed with only symmetric functions. Given the implemented algorithm this entails an approximation which can be tuned:

\[
\text{A1FIT atomicseparation}
\]

atomicseparation

is the threshold distance between atoms, in Angstrom. The symmetric fit approximation is applied only for atoms farther apart. Default is 4.0 Å.

**Fit integrals**

For the computation of the Coulomb potential the program uses a large number of so-called fit integrals: the overlap integrals of a fit function with a product of two basis functions, where at least two of the involved three functions are centered on the same atom. In fact these are ordinary overlap integrals of STOs because the fit and basis functions are all STOs and a product of STOs on a center is itself also an STO.

Obviously, when the two involved atoms are far enough apart, such overlap integrals become negligibly small. All fit integrals are ignored (and not computed) that are smaller – according to a rough but reasonable estimate – than a preset threshold.

The value of this threshold can be set via input:

\[
\text{EPSFIT accfitint}
\]

The threshold for ignoring fit integrals is \(10^{-\text{accfitint}}\). The default for accfitint is 4.0.
Atomic radial grid

For each atom the charge densities and the coulomb potentials of frozen core and valence electrons are computed in a radial grid and stored on TAPE21. The values in the points of the molecular numerical integration grid are then evaluated by interpolation from the table of radial values.

The radial grid consists of a sequence of \( r \)-values, defined by a smallest value, a constant multiplication factor to obtain each successive \( r \)-value, and the total number of points. Equivalently it can be characterized by the smallest \( r \)-value, the largest \( r \)-value, and the number of points; from these data the program computes then the constant multiplication factor. The characteristics are set with

\[
\text{RADIALCOREGRID} \{\text{nrad}=\text{points}\} \{\text{rmin}=\text{rmin}\} \{\text{rmax}=\text{rmax}\}
\]

points
The number of radial grid points; default: 5000.

rmin
The shortest distance used in the radial grid; default: 1e–6 Å.

rmax
The largest distance in the radial grid; default: 100 Å.

\( \text{rmin} \) and \( \text{rmax} \), when specified, are interpreted as specified in units of length defined by UNITS.

The keyword name RADIALCOREGRID has historical reasons: in earlier releases the radial grid was used only for the frozen core density and potential.

Dependency (basis set, fit set)

Conceivably the sizes of basis and/or fit sets may be so large that the function sets become almost linearly dependent. Numerical problems arise when this happens and results get seriously affected. Although for the fit set a few (incomplete) tests are carried out, the program will generally not check such aspects and carry on without noticing that results may be unreliable.

A new feature has been implemented to take care of this. For reasons of compatibility with previous versions and also because our experience with it is limited so far, we have chosen to make application of it not the default.

You have to activate it explicitly. Our experience so far suggests that real problems only arise in case of large basis sets with very diffuse functions (i.e.: not with the normal basis sets provided in the standard package).

Use of the key DEPENDENCY turns internal checks on and invokes countermeasures by the program when the situation is suspect. A few technical (threshold-type) parameters can be set as well, but this is not necessary, assuming that the defaults are adequate.

\[
\text{DEPENDENCY} \{\text{bas}=\text{tolbas}\} \{\text{eig}=\text{BigEig}\} \{\text{fit}=\text{tolfit}\}
\]

tolbas
A criterion applied to the overlap matrix of unoccupied normalized SFOs. Eigenvectors corresponding to smaller eigenvalues are eliminated from the valence space. Default value: 1e-4. Note: if you choose a very
coarse value, you'll remove too many degrees of freedom in the basis set, while if you choose it too strict, the numerical problems may not be countered adequately.

**BigEig**

Merely a technical parameter. When the `DEPENDENCY` key is activated, any rejected basis functions (i.e.: linear combinations that correspond with small eigenvalues in the virtual SFOs overlap matrix) are normally processed until diagonalization of the Fock matrix takes place. At that point, all matrix elements corresponding to rejected functions are set to zero (off-diagonal) and `BigEig` (diagonal). Default: 1e8.

**tolfit**

Similar to `tolbas`. The criterion is now applied to the overlap matrix of fit functions. The fit coefficients, which give the approximate expansion of the charge density in terms of the fit functions (for the evaluation of the coulomb potential) are set to zero for fit functions (i.e.: combinations of) corresponding to small-eigenvalue eigenvectors of the fit overlap matrix. Default 1e-10.

**Notes:**

5) Application / adjustment of `tolfit` is not recommended: it will seriously increase the CPU usage while the dependency problems with the fit set are usually not so serious anyway.

6) Application of the `DEPENDENCY/tolbas` feature should not be done in an automatic way: one should test and compare results obtained with different values: some systems look much more sensitive than others. We have, so far, not been able to understand an unambiguous pattern in these experiences. Of course, when things become clearer in this respect, we will implement the corresponding intelligence into the program.

7) When the `DEPENDENCY` key is used, the numbers of functions that are effectively deleted is printed in the output file, in the SCF part (CYCLE 1) of the `COMPUTATION` section.

8) The `TAPE21` result file of a calculation that used the `DEPENDENCY` key contains information about the omitted functions and these will also be omitted from the fragment basis when the `TAPE21` is used as a fragment file.

**Control of Program Flow**

**Limited execution**

`STOPAFTER` *programpart*

`programpart`

Must be a predefined name associated with a (major) part of the program With this key you tell ADF to terminate the job after the named program part has been executed.

A survey of the recognized names with a brief explanation follows below. The program parts are listed in order of execution: by taking a name further down the list you execute a larger part of the program.
**INIT**
initialization procedure, input reading and printing of the output header with the job identification.

**INPUT**
input-reading module.

**GEOMET**
geometry section: organization of atoms in types of atoms and fragments, checks of the actual fragments against information on the attached fragment files.

**CONFIG**
electronic configuration (if not determined only by the SCF procedure), printout of symmetry subspecies.

**MAINSY**
generation of symmetry information, representation matrices, etc.

**SYMFI**
construction of symmetry adapted fit functions.

**CBLOCK**
generation of integration points and the distribution of them in the blocks that control the internally used segmented vectorization loops.

**ENGRAD**
Relevant only in an optimization calculation. Engrad calculates energy gradients. The geometry is not yet updated and no printing of convergence tests and new coordinates is carried out.

**GEOPT**
This routine evaluates energy gradients and updates the geometry accordingly; it also prints the convergence tests and the computed new coordinates. Compare ‘stopafter engrad’.

**FORCEMATRIX**
in a Frequencies run, terminate the calculation when all displacements have been done and before any further processing of the computed hessian, such as the determination of normal modes, takes place.

**Direct SCF: I/O vs. recalculation of data**

The program's performance can be defined in terms of the amounts of time (CPU and I/O seconds) and disk space used in a calculation. Also important for the human user is the turn-around time. On multi-user machines CPU-cheap jobs may take a lot of real time to execute due to I/O scheduling. Therefore it can be a good idea to recompute some items rather than store them on disk. This will increase the amount of CPU time but reduce disk access and it may also improve the turn-around. Another consideration is of course that storage of data on disk may exhaust the available disk space in case of big calculations so that recalculation rather than storage is unavoidable.

```
| DISK{no}fit | {no}basis |
```

instructs ADF how to handle the values of the fit functions and basis functions in all integration points: calculate once and store on disk or recompute whenever needed. The (optional) arguments are fit or nofit, and basis or nobasis. fit and basis tell ADF to store the corresponding data on file; the prefix no induces recalculation whenever the data is needed.
Defaults are nofit and nobasis: direct-SCF mode for both features (this can be modified at the installation of ADF, see the Installation Manual).

The key DISK has replaced in ADF 2.0 the key DIRECTSCF in ADF 1.x, and extended the applicability of the I/O versus recalculation choice from fit functions-only to basis functions as well.

**Skipping**

With the following key you can restrict which parts of the program are actually executed:

```
SKIP argumentlist
```

**argumentlist**

A sequence of names, separated by blanks or commas. SKIP may occur any number of times in input. The names in the argument list refer to various items that are associated with parts of the program. With this key you tell ADF to skip the named program part(s) and to continue execution thereafter. The program does not check any consequences and may even crash when variables have not been initialized or have attained incorrect values due to the skipping.

Use of this key should be contemplated only in debugging and testing sessions, in which you may skip the computation of certain data when before that data will be needed you'll halt the program to inspect something. Recognized and operational arguments are for instance (possibly not complete due to frequent extensions in this respect): ATPAIR, ETS, FITINT, ORTHON, QMPOT

**Ignore checks**

ADF performs several checks during a calculation, and stops with an error message when intermediate results are suspicious, when input-specified instructions are incompatible, etc. These controlled aborts can in some cases be overruled. Of course, the checks have been inserted for good reasons and one should realize that ignoring them probably produces incorrect results and/or may lead to a program-crash.

```
ALLOW argumentlist
```

**argumentlist**

A sequence of names, separated by blanks or commas. ALLOW may occur any number of times in input, see the list below for the names that can be used.

**BADCOREINT**

Numerical integration of the frozen core density should closely approximate the analytical value. If the deviation is large compared to the user-specified numerical integration precision the program aborts with an error message like “BAD CORE INTEGRAL”. This control is overruled by using this ALLOW option.
**BAD INTEGRALS**

Only applicable when the direct-SCF option is turned off for the basis functions. (This happens automatically for ZORA full-potential calculations). In that case, a sequence of elementary overlap integrals are evaluated with the numerical integration grid and the outcomes tested against the analytical value. If the deviation is too large a warning is issued. Above a certain threshold the program will abort, unless you override the exit with this ALLOW option.

**BADSCF**

If the SCF procedure hasn't converged, any geometry manipulations (optimization, linear transit...) will be aborted because the energy gradients are not reliably computed in a non-self-consistent field.

**CLOSEATOMS**

Atom-atom distances should not be less than 0.2 Bohr. This is checked in the program section where the numerical integration grid is generated.

**REL GEO**

Geometry manipulation (optimization, linear transit...) is not supported for all of the relativistic options. See RELATIVISTIC.

**SMALL BLOCKS**

The list of numerical integration points is partitioned in blocks, so as to fit data arrays (for instance values of all basis functions in the points of a block) in available memory. The program computes the maximum block length from available memory and size parameters such as numbers of basis functions. A small block size implies a severe reduction in CPU efficiency. Therefore, the program aborts (by default, to override by this ALLOW option) if the block length turns out to be very small (less than 10).

**XC**

Certain combinations of the Density Functional options or application of them with some other features are not allowed. See XC.

---

**Parallel Communication Timings**

With the key

```
| COMMTIMING
```

in the input you instruct ADF to skip normal execution and perform only a test on the gather, broadcast and combine routines, used in a PVM version of ADF. Obviously, this is only meaningful if such an ADF version has been installed.

---

**Technical Settings**

**Memory usage**

The amount of memory used by the program during a calculation is determined by three quantities:
- The size of the program itself (executable statements, fixed arrays in subroutines...). This quantity depends on the program release number and is currently somewhere between 10 and 16 Mb.
- Buffer space used by ADF for more efficient I/O handling. This quantity is set at installation. See the Installation Manual.
- Dynamically allocated arrays. The program allocates memory dynamically during the run conform the requirements of the actual calculation. A preset maximum applies (defined by the Installation, see the Installation Manual) to the total amount of such dynamical memory. The program will stop when an attempt is made to exceed this maximum.

The actual dynamical allocation of memory happens in memory blocks, separately for the different data types (real, integer, logical, character). The size of these blocks is set at installation.

Via input the parameters used in the dynamical allocation mechanism can be adjusted to override the installation-set defaults. This applies to the total maximum of dynamical memory (the most significant application), but also to the sizes of the memory blocks by which the actually used memory is built up. The latter aspect is rather technical and should not play a role for the normal user.

All input keys pertaining to dynamical allocation are simple keys and specify the corresponding quantities in Mbytes. The total amount of memory (for scratch space):

MAXMEMORYUSAGE n

The technical parameters, determining the chunks of memory allocated at a time:

REALMEMBLOCK n
INTEGERMEMBLOCK n
LOGICALMEMBLOCK n
STRINGMEMBLOCK n

The defaults are defined by the installation. The MAXMEMORYUSAGE value will have been chosen specifically by the user/installer to accommodate his machine(s). The other (*MEMBLOCK) parameters are in principle merely technical and will usually not have been adjusted by the installer. Standard values are:

- reals: 1
- integers: 1/2
- logicals: 1/32
- strings: 3/2

As already suggested by the default values: the input values (n) for the memory-usage keys need not be integers: reals or integer fractions are allowed.

Vector length

Numerical integration is applied in ADF to evaluate Fock matrix elements and many other quantities that are defined as integrals over basis functions, the charge density, the potential, etc. As a consequence a large part of the CPU time is spent in simple do-loops over the integration points. The total number of points depends on the
required precision and on the number of atoms, the geometry and symmetry. All such numerical integration
loops are segmented into loops over blocks of points, each block consisting of a certain number of points. This
latter defines the most inner do-loop and hence determines vectorization aspects.
Depending on the computer, c.f. the compiler, vector operations may be executed more efficiently using longer
vectors. Long vectors increase the demand on Central Memory however because the program may sometimes
have to access large numbers of such vectors in combination (for instance all basis functions) so that they must
be available in memory simultaneously. The optimum vector length depends therefore on the balance between
vectorization efficiency and memory usage. The maximum vector length that you allow the program to use can
be set via input.

VECTORLENGTH vectorlength

The default is set at the installation of ADF on your platform, see the Installation manual. For organizational
reasons the true vector length actually used in the computation may be smaller than the value defined with this
key, but will not exceed it (except in a Create run, but in that case performance and memory usage are no hot
topics).

Tails
Each block of points (see above) covers (more or less) a certain region in space and can hence be assigned a
distance value with respect to a particular atom. These distances are used to control whether or not to evaluate
functions centered on that atom in that particular block of points.

TAILS {bas=tailbas} {fit=tailfit}
tailbas, tailfit
Accuracy levels, similar to the INTEGRATION parameter: a higher value implies higher precision: in this case,
basisfunctions and fitfunctions respectively are assumed zero in blocks of points that are at a sufficiently large
distance from the atom at which the function is centered. Sufficiently large is defined by comparing
the integral of the (radial part of the) function beyond that distance with the total integral. By default
tailbas and tailfit both depend on the numerical integration parameter

Special: if only the key is given, without any argument, all functions are evaluated in all blocks of points, so that
all screening of function tails is suppressed.

All Points
ADF makes use of symmetry in the numerical integrations. Points are generated for the irreducible wedge, a
symmetry unique sub region of space. Optionally the symmetry equivalent points are also used. This is achieved
by setting the key

ALLPOINTS
The key has no argument. The CPU time increases roughly by a factor equal to the number of symmetry operators, and the results should be the same. This key is available only as a debugging feature, to check the correctness of certain symmetry related algorithms.

**Full SCF**

During a geometry optimization the SCF convergence criterion is relaxed as long as the geometry has not yet converged. The value actually in effect depends on the current maximum (Cartesian) gradient as printed in the geometry update section. The effective convergence criterion is kept between the primary (final) and secondary criterion respectively which are both controlled by the key SCF (subkey CONVERGENCE). The key FULLSCF turns this feature off: the primary criterion will apply always.

```
| FULLSCF
```

The same effect is achieved by specifying the secondary criterion (key SCF) to be the same as the primary one.

**Full Fock**

At every cycle in the SCF procedure the Fock operator is computed in all integration points. By default the difference with the values of the previous cycle are used to compute changes in the Fock matrix elements. This leads in general to better computational efficiency in two ways: 1) when all such difference values in a block of integration points are very small such a block is skipped in the calculation. 2) if the values are not negligible but still rather small, the contribution from such a block to matrix elements between basis functions with small overlaps are neglected.

With the key

```
| FULLFOCK
```

this is turned off, so that the complete matrix elements are computed, no blocks are skipped and the neglect of matrix elements between functions with small overlaps (see also the key TAILS) is controlled solely by the function characteristics and precision requirements, not by the development of the SCF.

**Electrostatic interactions from Fit density**

By default the program tries to evaluate the electrostatic Coulomb interaction energy between the fragments in a molecule using the exact fragment charge densities. The implemented algorithm requires that all fragments are spherically symmetric. This is checked by the program by verifying that all fragments have been computed in ATOM symmetry. It that is not the case, an alternative method is applied, using the fitted charge densities of the atoms; this is an approximation with a small, but not insignificant error. The following key forces the program to apply the fit density approach even in the case of spherically symmetric fragments. This aspect applies only to the final bonding energy analysis, not to energy computations and their gradients within the automatic geometry optimizer. The purpose of this option is to simulate a previously existing situation where the electrostatic term in the bonding energy was computed from the fit density regardless of the fragments and their internal symmetries.
presence of this key in the input file triggers using the fit density.

### Save info

Several types of information, gathered during the run, are lost on exit. The `SAVE` key allows you to prevent the removal of such information.

#### info

A sequence of names separated by blanks or commas. `SAVE` may occur any number of times in the input file.

`SAVE` can also be used in the negative form:

#### NOSAVE info

The structure is similar:

- `info` is a list of arguments and `NOSAVE` may, like `SAVE`, occur any number of times in the input file.
- `SAVE` and `NOSAVE` turn save-info options on and off. A list of the available options, with their default status.

<table>
<thead>
<tr>
<th>item</th>
<th>default</th>
<th>explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>TAPE10</td>
<td>no</td>
<td>File with numerical integration data: points and weights, values of functions (depends on direct-SCF options) and core densities and potentials.</td>
</tr>
<tr>
<td>TAPE11</td>
<td>no</td>
<td>File with fit integrals.</td>
</tr>
<tr>
<td>TAPE13</td>
<td>no</td>
<td>Check point file. This file is lost (by default) only upon normal program exit, i.e. a program-controlled termination (including a program-detected error condition leading to controlled exit). In all such cases all info on TAPE13 is also present on TAPE21. <strong>TAPE13 exists when the program crashes into a core dump for instance, in which case it is uncertain what the contents of TAPE21 will be.</strong> The <code>SAVE</code> feature allows you to specify that <strong>TAPE13 is kept also upon normal exit.</strong></td>
</tr>
<tr>
<td>TAPE14</td>
<td>no</td>
<td>Scratch file with numerical integration data, mainly pertaining to individual fragments.</td>
</tr>
<tr>
<td>Timing</td>
<td>no</td>
<td></td>
</tr>
<tr>
<td>--------</td>
<td>----</td>
<td></td>
</tr>
<tr>
<td></td>
<td>During an ADF calculation the program gathers a large amount of timing information about the performance of different program parts. It can be printed, at various levels of detail, on standard output (key PRINT). It can also be stored on TAPE21, for later inspection, in a section Timing.</td>
<td></td>
</tr>
</tbody>
</table>

*Table VII. Arguments for the keys SAVE and NOSAVE.*
3 RECOMMENDATIONS, PROBLEMS, QUESTIONS

3.1 RECOMMENDATIONS

Precision

The quality of the calculation, given the selected model Hamiltonian – density-functional, relativistic features, spin-restricted/unrestricted... – is determined to a large extent by several technical precision parameters. The most significant ones are:

Basis set

Obviously, the quality of the basis set may have a large impact on the results. As a general rule, minimum and almost-minimum basis sets (types I and II) may be used for pilot calculations, but polarization functions should be included (III, IV) for more reliable results.

SCF convergence

The self-consistent-field (SCF) and geometry optimization procedures terminate when convergence criteria are satisfied. If these are set sloppy the results may carry large error bars. The default SCF convergence tolerance is tight enough to trust the results from that aspect. However, when the SCF procedure encounters severe problems an earlier abort may occur, namely if a secondary (less stringent) criterion has been satisfied (see the key SCF). Although this still implies a reasonable convergence, one should be aware that for instance the energy may be off by a few milli hartree (order of magnitude, may depend quite a bit on the molecule). It is recommended that in such cases you try to overcome the SCF problems in a secondary calculation, by whatever methods and tricks you can come up with, rather than simply accept the first outcomes.

Note: in a geometry optimization the SCF convergence criteria are relaxed as long as the geometry optimization has not yet converged. This should generally not affect the final results: the SCF density and hence the energy gradients may be somewhat inaccurate at the intermediate geometries, but since these are not a goal in themselves the only concern is whether this might inhibit convergence to the correct final geometry. Our experiences so far indicate that the implemented procedure is reliable in this aspect.

Geometry convergence

This is a far more troublesome issue. Three different types of convergence criteria are monitored: energy, gradients and coordinates. The energy does not play a critical role. Usually the energy has converged well in advance of the other items. The coordinates are usually what one is interested in. However, the program-estimated uncertainty in the coordinates depends on the Hessian, which is not computed exactly but estimated from the gradients that are computed in the various trial geometries. Although this estimated Hessian is usually good enough to guide the optimization to the minimum – or transition state, as the case may be – it is by far not accurate enough to give a reasonable estimate of force constants, frequencies, and as a consequence, neither of the uncertainties in the coordinates. An aspect adding to the discrepancy between the Hessian-derived coordinate-errors and the true deviations of the coordinates from the minimum-energy geometry is that the true energy surface is not purely quadratic and using the Hessian neglects all higher order terms. The gradients provide a better criterion for convergence of the minimizer and therefore it is recommended to
tighten the criterion on the gradients, rather than anything else, when stricter convergence than the default is required. The default convergence criteria, in particular for the gradients, are usually more than adequate to get a fair estimate of the minimum energy. Tighter convergence should only be demanded to get more reliable coordinate values (and in particular, when the equilibrium geometry needs to be determined as a preliminary for a Frequencies run).

Numerical integration accuracy

The key INTEGRATION determines the numerical precision of integrals that are evaluated in ADF by numerical integrals, primarily the Fock matrix elements and most of the terms in the gradients. In addition the INTEGRATION settings also determine several other computational parameters. The demands on numerical integration precision depend quite a bit on the type of application. The SCF convergence seems to suffer hardly from limited integration precision, but geometry convergence does, especially when tight convergence is required and also in transition state searches, which are generally more sensitive to the quality of the computed energy gradients. An extreme case is the computation of frequencies, since they depend on differences in gradients of almost-equal geometries. Frequency calculations on molecules with sloppy modes suggest that a NumInt precision value of 6.0 may be required. We recommend at least 5.0 in TS searches and in tricky optimizations, and 4.0 in normal optimizations. For optimizations with user-set convergence criteria we recommend to set the INTEGRATION precision at least 1.5 higher than the requested level of convergence for the gradient. For examples, a convergence threshold of 1e-3, should be combined with INTEGRATION=(3+1.5)=4.5

Note: a large INTEGRATION value implies that a lot more points will be used in the numerical integrals, thereby increasing the computational effort (roughly linear in the number of points). However, in optimizations and TS searches, the program will internally reduce the INTEGRATION settings as long as the geometry is far from convergence, so the costs in intermediate geometry steps may not so large. See the key INTEGRATION.

Electronic Configuration

Not specifying occupation numbers in input will not automatically result in the computational of the ground state. It may even lead to non-convergence in the SCF and/or in the determination of minimum-energy geometries or transition states. Therefore: whenever possible, specify occupation numbers explicitly in input (key OCCUPATIONS)!

Misunderstanding results of a calculation may easily result from a lack of awareness of how ADF treats the electronic configuration, which orbitals are occupied and which are empty. Unless you specify occupation numbers in input they will be determined from the aufbau principle but only during the first few SCF cycles. Thereafter the distribution of electrons over the different symmetry representations is frozen (see the key OCCUPATIONS, options AUFBAU and AUFBAU2). If at that point the potential has not yet sufficiently relaxed to self-consistency the final situation may be non-aufbau.
A related aspect is that the ground state does not necessarily have an aufbau occupation scheme. In principle, different competing electronic states have to be evaluated to determine which has the lowest total (strongest bonding) energy.

**Check output always carefully as to which orbitals are occupied.** In general, whenever possible, supply occupation numbers in input. Be aware that the automatic choice by the program may in a Geometry Optimization result in different configurations in successive geometries: the automatic assessment by the program will be carried out anew in each SCF procedure. If competing configurations with comparable energies have different equilibrium geometries, the geometry optimization has a high failure probability. The gradients computed from the SCF solution of a particular configuration drive the atoms in a certain direction, but in the next geometry, when the program re-determines the occupations and finds a different configuration, the resulting gradients may drive the atoms in another direction.

See the keys **CHARGE** and **OCCUPATIONS** for user-control of occupation numbers.

**Spin-unrestricted versus spin-restricted, Spin states**

If your molecule has unpaired electrons, you should run an unrestricted calculation, in principle. However, if this exhibits convergence problems (or if you simply want to save time: an unrestricted calculation takes a factor 2 more CPU time and data storage), you may consider to do it in two steps. First, run a spin-restricted calculation. Then perform a spin-unrestricted calculation using the restricted TAPE21 as a restart file. In the follow-up calculation you should specify the precise occupation numbers for the state you’re interested in, and use the SCF input key to specify only one SCF cycle (iterations=1). This prohibits convergence (so you keep the converged restricted orbitals) and gives you a fairly adequate approximation to a converged unrestricted result. See also the H₂ example run for a discussion in the Examples document.

An unrestricted calculation does not necessarily yield the multiplet configuration (triple, doublet…). This is a rather complicated matter, see the discussion in the Theory document.

**Geometry Optimization**

**Bond angles of zero or 180 degrees**

Avoid bond angles of 0 or 180 degrees. Use a dummy atom at a location orthogonal to the co-linear triple and define angles w.r.t. the dummy atom.

Be aware that bond angles can be explicit – these are easily recognized – but also implicit, in the definition of dihedral angles: it is absolutely imperative that such implicit bond angles are never 0 or 180 degrees: the dihedral angle will not be properly defined and an error will occur.
The program may in some cases be able to recover from 0/180 degree bond angles, but this is not a certainty. If it fails, the geometry update steps may go completely wild. Even worse: the steps may remain small but convergence is not reached, without a clear and explicit indication in the output about the cause.

**Sloppy modes**

Many molecules have sloppy modes, implying that geometric departures along these modes from the true minimum hardly change the energy and do not result in sizeable gradients. This usually shows up in slow convergence: energy and gradients appear to be converged but the computed step lengths, an assessment of the error in the geometry itself, do not disappear. It depends then on the purpose of the run whether a continued search for the minimum is useful: not if you only want to know the energy at the minimum, but certainly so if you want to determine all geometric parameters to high precision. Depending on the case you may therefore want to relax the convergence criterion on the coordinate steps. In the case of Z-matrix optimization this has to be done primarily for the *angular* coordinates because the bond lengths are usually much stiffer and will therefore not suffer from sloppy mode problems. If you insist on strict convergence of sloppy modes you should use a fair integration precision (at least 4.0, preferably 5.0).

**Step convergence**

The criterion on convergence of the coordinates (steps) is often *not* a reliable measure for the precision of the final coordinates, although it does give a reasonable estimate (order of magnitude). To get accurate results you should tighten the criterion for the gradients, rather than for the steps.

**Basis Sets for Organic Molecules: Single-zeta vs. Double-zeta**

A few tests have been done on small (less than 10 atoms) and medium sized (20-30 atoms) organic molecules (not containing transition metals) to compare double-zeta with single-zeta (minimal) basis sets. The two procedures were: *a*) a straightforward geometry optimization with the double-zeta basis set, and *b*) a geometry optimization with a single zeta basis, followed by a double zeta single-point calculation in the optimized geometry to evaluate the bonding energy and other properties. The results differed very little as regards the final geometry and therefore also as regards the energy etc.: from less than 0.01eV for small molecules to 0.25eV for a 26-atom case (debrisoquine C<sub>10</sub>H<sub>13</sub>N<sub>3</sub> ).

The additional single-point double-zeta calculation, required to obtain the bonding energy, makes that the computational costs are not automatically lower for the single-zeta optimization procedure. For the smaller molecules they take indeed some 25% *more* time. For the larger molecules this seems to get reversed however, the single-zeta approach being less than half as expensive as the double zeta.

Results for optimized bond-lengths in single-zeta bases are found to be very *inaccurate* when Sulphur atoms occur in *chains*. In such molecules these atoms need a 3d-polarization function, as provided in ADF's standard type-III (double-zeta plus polarization) basis sets. Results so far suggest that this particular problem with single-zeta bases does not occur when such atoms occur in a *ring* rather than in a *chain* within the molecule. Probably this is related to the empty d-shell in the atom being rather close to the valence p-shell. It can be expected that
the same behavior will be displayed by Phosphorus. Obviously one should not rely on such generalizations too strongly; it is sensible to always run a few tests and verify whether it can be applied to the case at hand.

Although the remarks above suggest a promising time-saving approach for the optimization of larger organic molecules, the conclusions must be considered with very great care since these investigations have been carried out only for a very small number of molecules. Also one should be extremely careful to extend the conclusions to transition metal complexes.

**Frequencies**

Outcomes of Frequencies calculations are usually quite sensitive to the geometry, so before computing the frequencies, one should make sure that the geometry is well converged at the level of the subsequent Frequencies calculation: the same model parameters and basis sets.

In all cases one should take care that the precision of Numerical Integration is adequate, preferably at least 5.0 (this is good advice anyway for a sound Frequencies calculation). Doing one-point, rather than two-point differentiation will roughly save you half of the time needed to complete the calculation. Increasing the integration precision will work the other way. To obtain high-precision results using one-point differentiation requires for one thing that you use very small displacements (smaller than the defaults) and high accuracy of numerical integration. Recent studies [14, 73] suggest to use a) two-sided displacements, b) an integration precision of 6.0 (!!).

This may not always be feasible due to the high CPU costs, but it should at least stress the importance of accuracy in the computation of frequencies.

A computation of frequencies runs over discrete displacements of atomic coordinates. When using Cartesian displacement coordinates, the program applies symmetry to skip symmetry-equivalent displacements and thereby save CPU time. In the output and logfile you’ll find in such a case that the ‘frequency displacement counter’ skips one or more values: the counter counts all possible displacements, while only the symmetry-unique ones are actually carried out.

**Relativistic methods**

The ZORA relativistic approach is often superior and in other cases at least similar to the older Pauli method. In particular for all-electron calculations generally, and for very heavy elements even within the frozen core approach, the Pauli method may exhibit significant shortcomings. This is mostly due to the variational instability of the Pauli formalism in the deep-core region near the nucleus. The bigger the basis set and the smaller the frozen core, the more likely this will show up, while generally speaking you might be tempted to use smaller cores and bigger basis sets to improve your results. The ZORA approach does not suffer from these problems and is, therefore, highly recommended over the Pauli formalism.
3.2 TROUBLE SHOOTING

This chapter contains hints to help you solve some problems and comments on frequently asked questions.

License file corrupt

You may find that, after having installed the license file, the program still doesn’t run and prints a message like “your license file is corrupt”. To explain how this may come about, and how you overcome this, a few words on license files.

Each license file consists of pairs of lines. The first of each pair is text that states, in more or less readable format typical aspects such as an expiration date, the version number of the software and so on. The second line contains the same information in encrypted format: a (long) string of characters that seem to make little sense.

The program reads the license file and checks, with its internal encrypting formulas, that the two lines match. If not, it stops and prints the “corrupt” message. So, there are two common reasons why it may happen to you:

1) You are using a license file for another version of the software than your executables correspond to. Newer (major) releases may contain a different encrypting formula, so that the match in old license files is not recognized anymore. So, please verify that your license file and executable belong to the same major release.

2) More likely: the license file as it has been created has been modified in some way. Sometimes, people inspect it and ‘clean it up’ a little bit, for instance by removing ‘redundant’ spaces, or by making some other ‘improvements’. Unfortunately, every such modification will destroy the encryption match and lead to the ‘corrupt’ error. Most of the times, however, the reason lies in the mailing system, by which the license file has been sent to you. If the encrypted line is rather long, the mailer may have cut it in two shorter lines. To verify (and correct) this: edit the license file and see if it consists of pairs of lines as described above. If not, re-unify the broken lines and try again.

Recover from Crash

A calculation may terminate in two ways: controlled or uncontrolled. Controlled termination includes cases where the program itself detects an error and decides that continuation of the calculation is impossible or pointless. In all such cases the standard exit routine is executed, resulting in an output section with some final information. This also ensures that the general result file TAPE21 is closed properly and all relevant information flushed to it.

Uncontrolled termination may occur, for instance when some bug causes the program to divide by zero, violate memory access restrictions, etc. Usually this leads to an immediate abort of the program by the Operating System and hence loss of control by the program. In such situations the information on TAPE21 may be incomplete because some of the data are kept in memory until the final termination of the program is carried out. It would be a terrible nuisance to see all time spent so far being lost. To remedy this ADF supports a check point file, named TAPE13, to help you recover at least some, if not most, of the results: not for analysis, but for
continuation from a point not too long before the fatal condition occurred. TAPE13 can be used, just like
TAPE21, as a normal restart file. See the RESTART key.

Memory Management

**Insufficient Space for Allocation**

*Problem:* The program aborts with error message **INSUFFICIENT SPACE FOR ALLOCATION** or **IRALOC:**
EXCEEDED MAXIMUM MEMORY! This message is issued both in the logfile and in the output file. In the latter a lot more is printed after it, giving more specific information about the problem.

*Cause:* The total amount of memory available for dynamic allocation of arrays is exceeded. This maximum amount is defined (by default) by the installation and can be adjusted via input for each particular calculation.

*See the key MAXMEMORYUSAGE.*

*Cure:* Use the key MAXMEMORYUSAGE in input (or reinstall the program with a larger default for this parameter).

*Note:* If workspace problems occur for relatively small calculations, there might be a bug. Notify your ADF contact. The same holds if the problem remains after you have increased the workspace: send us the output file so that we can have a look and check things out.

**Iraloc: out of memory**

*Problem:* The program aborts with error message **iraloc: out of memory.**

*Cause:* This means that the program attempts to use more memory than the system allows (for whatever reason).

*Cure:* decrease (!) the amount of memory that you allow the program to consume, by setting a lower value to the input key MAXMEMORYUSAGE.

*Note:* if this happens while you haven't even used the key MAXMEMORYUSAGE, the installation-parameters have not been set suitable for the machine at hand. This is not a serious problem, but it means that you will almost always have to use the MAXMEMORYUSAGE key in input.

**Marker Error**

*Problem:* The program aborts with one or more messages **MARKER ERROR**.

*Cure:* You've done nothing wrong. It is a serious bug, fortunately detected by the program itself. Send the complete output file to your ADF contact.
**SCF**

**No convergence**

**Problem:** The SCF procedure does not converge, the errors of successive cycles don't diminish. This may be due to any of a large number of causes. Some of the more frequently occurring:

**Possible Cause 1:** Electrons are hopping back and forth between two (or more) spatially very different orbitals at successive cycles. The reason that this may happen is that in DFT virtual (empty) one-electron orbitals tend to have a too-low orbital energy compared to the occupied ones, so that, if the true \( \text{HOMO} - \text{LUMO} \) gap is small enough, the computed spectrum may produce an empty orbital below an occupied one. Consequently, on the next SCF cycle the program may assign electrons according to the aufbau principle (say) and hence put electrons in orbitals that were empty on the previous cycle, and vice versa. On the next cycle, however, the now-occupied orbital gets a higher energy and the energy ordering is reversed again, resulting in re-adjusting the assignment of electrons. This may lead to strong oscillations and non-convergence.

**Cure:** Apply the feature `KEEPORBITALS` (key `OCCUPATIONS`), to let the program try to keep orbitals occupied that are spatially similar over subsequent cycles, rather than assigning electrons according to energy ordering.

**Possible Cause 2:** You have put a wrong number of (valence) electrons in the system, so that you are, for example, trying to compute a highly ionized system. Although this should in principle not always give rise to SCF problems, it turns out to happen often in practice.

**Cure:** Check that the net total charge of the molecule is correct. If you want to compute a strongly charged system that seems to cause SCF problems, try to use the `KEEPORBITALS` feature (key `OCCUPATIONS`). You may also apply strong damping (option `MIX`, key `SCF`), and suppress the DIIS procedure (key `SCF`, option `DIIS`, specifications `OK` and `CYC`): all these tricks may help to let the program operate more carefully (and more slowly) in SCF updating.

**Remark:** In case of SCF convergence problems, **always** check carefully whether it is a technical problem or a physical one. Technical problems may be addressed with various SCF strategy parameters. More often you’ll find it is in fact a physical problem. The system at hand may have two or more configurations that are competitive in energy, so that a one-determinant wavefunction approach is not suitable anyway, even within DFT. In such a case you should reconsider what you want in fact to be computed.

**Convergence difficulties with spin-unrestricted calculations**

If spin-unrestricted calculations fail to converge, you may try to run first a restricted calculation. Then perform the unrestricted while using the TAPE21 from the earlier run as a restart file. Now, in the restart run you have two alternatives that may stand a better chance to give you the desired result than your original failing calculation:

1. **Apply the MODIFYSTARTPOTENTIAL** key to steer the (at the least the initial) spin-density on a per-atom (fragment) basis towards what you expect should be the final self-consistent situation. It may be necessary to apply stronger damping, or do other SCF convergence tricks like level shifting, in order to preserve your initial construction long enough for the system to get near the minimum it should converge to. It is recommended to supply occupation numbers (in case the system has symmetry) in input.

2. **If all SCF tricks fail** (but you did get the restricted converged result), you may simply run a one-cycle unrestricted run, specifying the occupation numbers you consider appropriate and so obtain the unrestricted
occupation state for the restricted-self-consistent MOs. The difference with the ‘true’ converged unrestricted MOs is often quite small. See the Examples document for the discussion of a similar case (H₂).

Geometry Optimization

No convergence

Problem: In a Geometry Optimization there is no progress: the atomic positions hardly change or oscillate around, while the energy gradients don’t go to zero.

Possible Cause: Occupation numbers have not been supplied in input (OCCUPATIONS). For some molecules the procedure gets stuck between two (or more) different electronic configurations, with gradients pointing in different directions for the competing states. In each new SCF procedure, after a geometry update, the occupation numbers are re-determined, by default according to the aufbau principle. As a consequence, the successive SCF procedures may handle different electronic configurations and hence produce contradicting geometry-updates. See the key OCCUPATIONS.

Check: in the output file the occupation numbers that have been in effect during the successive SCF procedures. If they are different, then:

Cure: supply occupation numbers in input.

Alternative cause: SCF convergence not reached, or the criterion was too weak, or the precision of numerical integration in insufficient. Such causes may lead to inconsistencies between the true energy surface properties and the computed gradients. Usually this will only slow down the convergence but not prohibit it. However, if the SCF convergence is really problematic, it might get more serious. Reconsider the SCF strategy parameters, the Numerical Integration precision, or try KEEFORBITALS (a subkey to the key OCCUPATIONS).

Spurious jumps

Problem: during geometry optimization, the atomic configuration makes a large and unrealistic jump.

Possible cause 1: the triplet of atoms to which the current atom is related by the Z-matrix is (almost) co-linear. When, in a geometry step, the triplet passes through co-linearity, the dihedral angle for the current atom should make a discontinuous jump of 180 degrees. This is not checked in the program and the dihedral angle may not get corrected, resulting in a geometric jump of the atom (and hence of all atoms related to it by the Z-matrix).

Check: the triplets of atoms, used in your Z-matrix to define the dihedral angles. If one of them is almost colinear, then:

Cure: redefine the Z-matrix or use Cartesian optimization.

Alternative cause 2: the connectivity of the Z-matrix does not reflect the important bonds. Especially when the molecule contains (a large number of) rings, this badly affects the stability of the geometry update step. The reason is basically that computed Cartesian forces are transformed into changes of the curvilinear internal coordinates. The transformation between the two systems of coordinates is non-linear, but mathematically assumed to be linear. This is only a good approximation for small steps.
**Cure:** redefine the Z-matrix and/or (if the geometry steps are very large) set a smaller upper bound on the maximum step (key GEOMETRY, subkey STEP).

**Constraints are violated**

**Problem:** constraints are violated: coordinates that were specified as frozen turn out to change during the optimization or coordinates that should remain the same start to differ after a few geometry update cycles.

**Possible cause:** there is an internal conflict between different demands, usually: symmetry versus constraints. The problem arises easily when a constrained optimization is requested for a molecule with some symmetry while the coordinates were defined with a Z-matrix structure that does not properly reflect the symmetry. Usually the deviations from the requested constraints are small. If they are really large, there might be a bug and you should contact an ADF representative.

**Cure:** redefine the Z-matrix and/or use Cartesian optimization (if the constraints are expressible in Cartesians).

**Clearly wrong results (bond lengths)**

If the computed equilibrium geometry appears to exhibit unlikely values, typically significantly too short bond lengths, you may have run into a basis set problem, in particular (but not only) if the Pauli relativistic method is applied.

**Problem:** Optimized bond lengths are clearly too short. The energy may also look suspicious.

**Possible cause 1:** Basis set trouble: onset of Pauli variational collapse, if you have applied the Pauli relativistic option. Caused by small (or absent) frozen cores and/or relatively large basis sets, applied to heavy elements.

**Possible cause 2:** Basis set trouble also, but quite different from the previous potential cause: you have used relatively large frozen cores. When the atoms approach each other during the optimization and the frozen cores start to overlap, the energy computation and the computed energy gradients become more and more incorrect. This is a result of the inappropriateness of the frozen core approximation, which indeed assumes that frozen cores of neighboring atoms do not significantly overlap. Without going into a detailed explanation here, the net effect is that certain repulsive terms in the energy computation are missing and hence a spurious tendency to a ‘core collaps’ arises, yielding too short bond lengths.

**Cure:** Best is to abandon the Pauli method and use the ZORA approach instead for any relativistic calculation. If for whatever reason you insist on using the Pauli formalism, apply bigger frozen cores and, if that doesn’t help, reduce the basis set (not by deleting polarization functions, but by reducing the flexibility of the occupied-atomic-orbitals space, in particular s- and p-functions). Note, however, that large frozen cores can be a cause for trouble by themselves, irrespective of any relativistic feature. If you have reason to believe that your frozen cores might be too large, given the resulting bond lengths in your calculation, you have to pick smaller cores (and hence be very wary of using the Pauli formalism for any relativity).
Frequencies

Imaginary Frequencies

**Problem:** totally unexpected significant imaginary frequencies are obtained (in a Frequencies run) where you are pretty convinced that all frequencies should be real.

**Possible cause:** problems with the electronic configuration. If there are competing configurations, the electronic *states* in the different displaced geometries may be different, resulting in energies and gradients belonging to different potential energy surfaces to be compared and combined into force constants (frequencies).

**Check:** orbital occupations and SCF convergence behavior: if the SCFs in the displaced geometries start with large errors and/or converge very slowly you are likely to have stumbled into different configurations, so that the results from the displaced geometries are incompatible.

**Cure:** This is a difficult situation that may require some experimenting and judicious manipulation of the various SCF options. The bottom line is that you should try anything you can to ensure that all involved geometries have the same electronic configuration. As long as you fail to achieve this, the results are meaningless.

Geometry-displacement numbers in the logfile are not contiguous

**Problem:** successive displaced geometries in the logfile are numbered, but in your case these numbers make sudden jumps, like “0, 1, 2, 5, 6, 13...”

**Cause:** you’re using Cartesian displacements in a system that has some symmetry in its equilibrium geometry. The program skips the displacements of symmetry-equivalent atomic coordinates to save time. The displacement counts in the logfile do not run over the actually performed displacements but over all atomic coordinates that could be displaced if no use were made of symmetry properties.

**Cure:** there is no error, don’t worry.

Input ignored

**Problem:** the program doesn’t get past input and aborts with a message `EOF WHILE READING`... Or the program seems to ignore some parts of input and as a consequence goes wrong somewhere. Or it seems that part of the input has not been read correctly or not at all.

**Cause 1:** You have used tab characters in your input file. These are not normally visible when you edit your file, but they will affect the program’s scanning of the input. When you use tab characters in the input, it is very likely that the program will do something wrong somewhere. Tabs may be ignored by the program, so that items that you believed were separate (by a tab!) are in fact read as contiguous.

**Check:** the input file on tab characters.

**Cause 2:** misusage of one of the block-type keys or general keys. A case that relatively often shows up is typing a title as first line of the input file, *without preceding it by the keyword TITLE*. The program does not understand this as the title, but rather tries to interpret the first word as a
keyword. This leads to an error if the first word is recognized as one of the pre-defined block-type keys (possibly abbreviated).

*Check:* the input file on usage of block-type keys and on proper usage of a title.

*Cause 3:* incorrect processing of expressions or unintended replacement of names by numerical values. Various kinds of mis-typing or incorrect usage of variables may cause this.

*Check:* how the program sees input, after parsing. This can be done by rerunning the job, with as first line in input:

```
PRINT PARSER
```

This will cause the program to copy each input line *twice* to output, the second time after having parsed it. You may use `STOPAFTER_INPUT` or `STOPAFTER_INIT` to let the program quit early so you can inspect what is going on with the input reading.

### SFO Populations

In the section that prints the SFO populations of (selected) MOs you may occasionally find, for some SFOs in some MOs, negative SFO contributions. This may seem unphysical and hence suspicious, but it is ‘only’ a result of the Mulliken-type analysis method that underlies the computation of the SFO contributions. See the section below that discusses the output file. Likewise for larger-than-100% contributions: don’t worry too much, these numbers may be correct (mathematically, given the Mulliken population formulas).

### Error Aborts

The program performs a large number of checks during the calculation and may stop when it detects and error. It is close to impossible to show here a complete list of all possible error messages. In a large number of cases, additional information is printed in the output file to provide a clue as to the cause of the error. It is always useful to carefully inspect the printed info and to try to understand the meaning of any error- or warning messages. If you can’t find your way out, try to get help from your ADF provider. If that fails, contact us directly at support@scm.com.

### Warnings

The program attempts to detect bugs, instabilities, convergence problems, et cetera and may issue warnings when something looks suspicious. This is not necessarily fatal to your results, but you should be cautious and try to understand what the messages are about. Most warnings are printed in the *logfile*. Usually there is corresponding and more extensive information in the standard output file.
3.3 QUESTIONS

Overlap matrix in BAS representation

How do I get the overlap \((S)\) matrix in the BAS representation?

It is stored on a scratch file TAPE15, which is normally deleted at the end of the calculation because it can be pretty big. To retain that file, insert “SAVE TAPE15” in your input, see the \texttt{SAVE} key.

TAPE15 is a KF file, which you can manipulate with the KF utilities. On TAPE15 the overlap matrix is stored as the variable “\texttt{smat}” in the section “Matrices”, in reduced (triangular) format: \((1,1)\), \((1,2)\), \((2,2)\), \((1,3)\) et cetera
4 FILES

ADF produces two ASCII files: standard output and the log file. The latter is a very concise summary of the calculation's progress during the run. Furthermore, ADF produces and reads binary data files. Most of these files have the so-called KF format. KF stands for Keyed File: KF files are keyword oriented, which makes them easy to process by simple procedures. KF files are Direct Access binary files. Consult the UTILITIES document for how to use some standard utilities for processing KF files.

4.1 PARALLEL EXECUTION

If a parallel version of ADF has been installed and the installation parameter hasowndirectory in the file settings is false (see the Installation manual), then all parallel processes generate and use files in the same directory. In this situation the different kids must be able to discriminate between their own files and those of other kids. This is achieved by modifying the filenames: an integer is appended with an underscore. So, rather than TAPE10 (say), we would have TAPE10_0 for the parent process and TAPE10_1, TAPE10_2, etc. for the kids. This does not need to bother you if you use the program run scripts provided with the package: these control execution of adf in parallel (PVM). See the Installation manual and the Examples document for more details.
4.2 STANDARD OUTPUT

The (standard) output file contains information of the main characteristics of the run, the SCF and geometry optimization results, bonding energy and population analyses. Major parts of output can be regulated with print switches, see the keys (NO) PRINT and EPRINT.

By default the program produces quite a bit of output, for a large part related to (Mulliken-type) population analyses of the molecule in total, as well as of individual orbitals, both in terms of the elementary basis functions and in terms of the SFOs, the symmetry-adapted Fragment Orbitals.

The fragment-oriented approach of ADF is very suitable for a thorough chemical analysis of molecular orbital properties and a conceptual representation of results. New users are advised to spend time and get familiar with the SFO-type analysis. It is an extremely more powerful tool to understand the electronic structure of the molecule than the classical atomic orbital populations.

A summary of output is given below, assuming that default values apply for all print switches. Keep one of the Example outputs at hand when reading the description below.

Input Echo, Output Header

- Copy of the input file, except any InLine records: these are expanded and the contents of the inlinefile replaces the InLine command in the echo.
- Header with the program name, the release number and a copyright statement.
- Directly below the header are printed the job identification, title, and any comments that may have been supplied via input (key Comment). The job identification is comprised of the adf release number and the date and time of the calculation.

Main Job Characteristics

- The Model Parameters such as the Density Functional and relativistic options.
- A list of attached files: restart data files and fragment files.
- The run type: Geometry Optimization, Frequencies...
- (Initial) geometric data: atomic positions, atom types, defined fragments, and the inter-atomic distance matrix.
- The point group symmetry, with a list of the irreducible representations and subspecies.
- The electronic configuration: occupation numbers (if specified), their distribution over spin-α and spin-β, and the net charge of the molecule.
Build Info: Fragments and Function Sets

See the print options EPRINT:FRAG, EPRINT:SFO and FUNCTIONS.

- Correspondence between fragments in the molecule and the corresponding master fragments on the pertaining fragment file. (This output is by default off)
- SFOs: the Symmetry combinations of Fragment Orbitals. The SFOs are the basic conceptual entities for the analysis of MOs and other results.
  Note: The FO coefficients that expand the SFOs are normalized in the sense that they add up (squared) to unity. The resulting SFO function is not necessarily a normalized function. The FOs are normalized, so it depends on the overlap between the FOs what the self-overlap and hence the norm of the SFO is.
  Also printed are, for each subspecies in each irrep separately, the indices of the elementary basis functions from which the FOs and hence the SFOs are built up. (The overlap matrix of SFOs is printed much later, in the SFO Populations section after everything (SCF, Geometry) has cycled to convergence).
- The elementary basis functions, fit functions, and the frozen-core levels of the atoms. First the lists of function sets, defined by radial behavior and the angular quantum number, are printed for all atom types on which the functions are centered. Thereafter follows the complete BAS list where the function sets have been expanded over all atoms (the sets are printed only for the atom types) and also over all Cartesian harmonics (6, not 5 d-functions, et cetera). In this printout the numbering can be found to which the SFO survey above refers.

Technical Parameters

See the PRINT key TECHPAR.

- Parallelization and vectorization characteristics.
- Direct versus Store-On-Disk options.
- Update strategy parameters for Geometry updates (if applicable) and for the SCF procedure.
- General precision settings for numerical integration and neglect-of-small function values (in integral evaluations).

Computational Report

See the print switches COMPUTATION, EPRINT:NUMINT, EPRINT:SCF, EPRINT:GEO.

Numerical integration

General grid-generating parameter(s) and the number of generated (symmetry unique) integration points, with their distribution over the distinct kinds of integration regions: the atomic (core-like) spheres, the remaining interstitial regions between the atoms (atomic polyhedra), and the outer region, i.e. the part of space around the molecule.

Partitioning of the points in blocks. In general there are too many integration points to have all pertaining data (values of basis functions in the points etc.) in memory. A segmentation in blocks of points is therefore applied, processing a block of data at a time after loading it from disk or recomputing it (depending on the
Direct options). This also determines vector lengths and hence vectorization performance in numerical integral evaluations.

Integration Tests. The generation of the points involves an adaptive procedure to tune the point distribution such that a pre-set precision of several test integrals is achieved with a minimal number of points. The generated scheme is a posteriori tested by evaluating a few integrals in the actual molecule. This does not result in any subsequent adaptation of the grid but only produces info for the user to verify that all goes well. If the results are suspicious a warning is issued and if the results are too bad, the program will abort.

The most important and significant test is the evaluation of the self-overlaps of all symmetry-adapted elementary basis functions. The maximum and root-mean-square (relative) errors are printed. The number of significant figures suggested by the rms error should roughly equal the accuracy parameter. This may not hold so well for extremely low values of the parameter (less than 1.5 say) where results become unpredictable. Likewise for very high values (greater than 6.0 say) where the adaptive procedure has not extensively been tested and hence the results might deviate more (not necessarily in the wrong direction!). This extensive testing is not carried out in Direct-SCF (BAS) mode because in that case the necessary information is not available (basis functions are only computed when needed in the SCF).

A test that is always carried out is the numerical integration of the total frozen core density (summed over all atoms in the molecule). Also here a warning or even abort will occur when the result indicates that the integral has insufficient accuracy compared with the integration precision parameter.

SCF procedure

at each cycle: for each irreducible representation: the one-electron orbital energies and the occupation numbers for a contiguous sequence of orbitals. The indices of the lowest and highest MOs (in energy ordering) are printed directly after the irrep label. With this information you can check the electronic configuration. When convergence is problematic, more info appears at the higher iterations.

The involved orbitals are usually the highest few occupied and the lowest few unoccupied orbitals, see the EPRINT subkey EIGVAL. During the SCF, as soon as the distribution of electrons over irreps is frozen, only the occupied orbital energies are computed and hence printed.

Also printed at each SCF cycle is the difference of the density matrix (P-matrix) with the previous cycle: the average and maximum difference in the diagonal elements.

At the end of the SCF: concise information about the density-fit precision: the error integral for the SCF density. The error integral is the integral of the difference between the exact density and the fit density, squared. Such values have very little to do with numerical integration, rather they show whether or not the employed set of fit functions are adequate to describe the SCF density. Error integral values that significantly exceed 1e-4 times the number of atoms are suspicious and may indicate some deficiency in the fit set for the actual calculation.

On the last geometry (in an optimization) the fit-error integrals are also printed (in the Results section, see below) for the initial (sum-of-fragments) density and the orthogonalized fragments (see Chapter 1.2)

- Gross atomic charges, computed from a Mulliken population analysis.
- Geometry Updates. The contents of this section depends on the RunType: Geometry Optimization, Frequencies... It is absent in a Create run and in a SinglePoint calculation.
- Gradients on the atoms: derivatives of the energy w.r.t. changes in the nuclear coordinates.
Summary of convergence issues. One of the items considered for convergence is the maximum Cartesian gradient. This value corresponds in principle to one of the Gradients on the Atoms. Differences may occur due to user-set and automatic constraints. The printed Gradients are the raw gradients, the maximum Cartesian gradient is the maximum over relevant gradients: this ignores gradients in frozen coordinates. Furthermore, gradients in coordinates that are forced to remain equal are averaged before the maximum is selected; finally the raw gradients are processed to eliminate spurious components such as gradients in rigid motions (translations and possibly rotations). In a Z-matrix optimization any user-set constraints apply to the Z-matrix coordinate-derivatives and the maximum Cartesian gradient is selected from the Cartesian gradients that are recomputed from the constrained Z-matrix gradients.

New coordinates: Cartesian and Z-matrix if applicable. Optionally the new inter-atomic distance matrix is given (not by default).

The Computational info is repeated in all cycles (SCF and geometry) until the iterations have terminated.

Results

See also Chapter V.
Details of the RESULTS part in the output file depend on the run type. For output in a Frequencies run, see below. In other applications:

Nuclear and Electronic Configuration

- The final atomic coordinates (only in an optimization run).
- One-electron orbital data: occupation numbers and energies, HOMO and LUMO energies and, if applicable, a list of partially occupied MOs.
- Orbital energies of the Core Orbitals: using Koopman's theorem this gives an approximation for the core-orbital ionization energies (XPS). Absolute values may not be very good, because relaxation upon the ionization is not accounted for, but relative values are expected to be fair.

ESR Properties

In an ESR calculation only (keywords ESR, QTENS):
- When the G-tensor is computed (spin-orbit calculation, spin-restricted, one unpaired electron):
  - The MO of the unpaired electron, in BAS representation and in Lowdin representation.
  - The eigenvectors of the G-tensor and the isotropic value
  - The eigenvalues of the G-tensor.
Note: in a G-tensor calculation, the MO and G-tensor values are also printed after the first SCF cycle, corresponding to the sum-of-fragments density.

Then the A-tensor and Q-tensor parts (Q-tensor if the QTENS keyword has been specified in input). For each atom:
- The isotope characteristics (nuclear spin, g-value and quadrupole moment)
- The position of the nucleus in the molecule (Cartesian coordinates)
- The eigenvectors of the A-tensor (Nuclear Magnetic Dipole Hyperfine interaction)
- The eigenvalue (if the g-value of the nucleus is non-zero, the eigenvalues have been multiplied by the g-value) and the isotropic value.
- The eigenvectors and eigenvalues of the Q-tensor. The eigenvalues have been multiplied by the quadrupole moment if non-zero.

**Populations, Charge analysis**

**Mulliken populations**

These are based on the elementary atomic basis functions (BAS). The individual BAS populations are printed together with summaries of the populations in all basis functions with the same angular moment quantum number on the same atom.

A final summary is obtained by adding all functions on each atom, yielding the atom-atom populations. The atomic gross charges are derived from the net and the overlap populations in the usual way.

In addition, a population analysis may be given of individual MOs (by default this is suppressed). See the EPRINT keys SCF (option MOPOP) and ORBPOP.

**Hirshfeld charges**

Of the three methods applied in ADF to compute charges (Mulliken, Hirshfeld, Voronoi) we recommend the Hirshfeld analysis [86, 87] and the analysis based on Voronoi deformation charges [74, 88], see below. The fragments to which the Hirshfeld charges apply are enumerated in the early geometry part of the output file, where for each fragment the numbers of the atoms are given that belong to the fragment.

The sum of the Hirshfeld charges may not add up to the analytical net total charge of the molecule. Any deviation from this is caused by numerical integration precision (small effect) and the neglect of long-distance terms that ADF uses to speed up the integral evaluations. This approximation does not affect very much the energy and molecular orbital properties, but it does show up in the sum-of-charges somewhat more. It does not indicate an error (unless the deviation is really large, say in the order of 1‰ of the total number of electrons).

**Voronoi Charges**

The Voronoi charge analysis is based on a straightforward numerical integration of the charge density and assignment of the density in a particular point in space simply to the nearest atom. This is a bit awkward when very differently sized atoms are neighbors: the charge partitioning is taken exactly halfway, even for say a Uranium-Hydrogen atom pair. The absolute numbers of this charge analysis should not be relied on.

However, the corresponding changes upon going from the initial (sum-of-fragments) situation to the final SCF do give a good impression of how the charge distribution between atoms modifies upon the formation of bonds. The so-defined Voronoi Deformation Density is a very suitable method for the determination of atomic charges.

The Voronoi analysis is split up in the amounts of charge within the atomic spheres and those in the remainder of the atomic Voronoi cells (the Voronoi cell of an atom is the region in space closer to that atom than to any other).

In the same fashion as for the Hirshfeld analysis, a summation over all atoms is given which should yield zero (for a neutral molecule). The deviation from zero is caused by numerical integration and by neglect-of-long-distance-terms; the same remarks apply as for the Hirshfeld analysis above.
**Dipole moment, Electrostatic potential**

Dipole moment. Note that in a ion the value of the dipole moment depends on the choice of the origin, as follows from elementary electrostatic theory.

Electrostatic potential at the nuclei: the Coulomb potential of the molecule at the nuclear positions, where the contribution from the nucleus itself is omitted.

**Energy and MO analysis**

MOS expanded in SFOs

This gives a useful characterization of the character of the self-consistent molecular orbitals. Additional information is supplied by the SFO population analysis, see below.

The definition of the SFOs in terms of the Fragment MOs has been given in a earlier part of output (section BUILD). The SFO occupation numbers that applied in the fragments are printed. This allows a determination of the orbital interactions represented in a MO.

Be aware that the bonding/antibonding nature of a SFO combination in a MO is determined by the relative signs of the coefficients and by the overlap of the SFOs. This overlap may be negative! Note also that SFOs are generally not normalized functions. The SFO overlap matrix is printed later, in the SFO-populations part below.

**Bond Energy analysis**

The bond energy and its decomposition in conceptually useful terms: Pauli (exchange) repulsion, total steric repulsion, orbital interactions (partitioned into the contributions from the distinct irreducible representations), and corrections for some approximations (fitting and Transition State analysis procedure).

For a discussion of bonding energy decompositions and applications see e.g. [75, 77, 89-95]

**SFO population analysis**

For each irrep:

- Overlap matrix of the SFOs. Diagonal elements are not equal to 1.0 if the SFO is a linear combination of two or more Fragment Orbitals. The Fragment Orbitals themselves are normalized so the diagonal elements of the SFO overlap matrix give information about the overlap of the Fragment Orbitals that were combined to build the SFO.

- Populations on a per-fragment basis for a selected set of MOs (see EPRINT, subkey ORBPOP). This part is by default not printed, see EPRINT subkey SFO.

- SFO contributions per MO: populations for each of the selected MOS. In these data the MO occupation numbers are not included, so that also useful information about the virtual MOs is obtained. The printout is in matrix form, with the MOS as columns. In each printed matrix a row (corresponding to a particular SFO) is omitted if all populations of that SFO are very small in all of the MOs that are represented in that matrix. See EPRINT, subkey ORBPOP.

Note that this method to define SFO populations (for orbitals) is very similar to the classical Mulliken type analysis, in particular regarding the aspect that gross populations are obtained as the diagonal (net) populations plus half of the related off-diagonal (overlap) populations. Occasionally this may result in negative (!) values for the population of certain SFOs, or in percentages higher than 100%. If you have such results and wonder if they can be right, work out one of the offending cases by hand, using the printed SFO overlap matrix and the printed expansion of the MOs in SFOs to compute ‘by hand’ the population matrix of
the pertaining MO. To avoid doing large calculations it is usually sufficient to take only the few largest MO expansion coefficients; this should at least qualitatively give the correct outcomes.
- Total SFO gross populations in a symmetry representation: from a summation over all MOs (not only those analyzed in the previous section of output) in the symmetry representation under consideration. In the gross populations the MO occupation numbers have been included.
- (Per spin): A full list of all MOs (combining all symmetry representations), ordered by energy, with their most significant SFO populations. Since there might be several significant SFO populations for a particular MO, and an SFO may actually be a linear combination of several (symmetry-related) Fragment Orbitals, this table could get quite extensive. In order to confine each SFO population specification to one line of output, the SFOs are indicated by the characteristics of the first term (Fragment Orbital) of its expansion in Fragment Orbitals. So, if you see the SFO given as the “2 P:x on the first Carbon fragment”, it may actually refer to the symmetry combination of, for instance, 2P:x and 2P:y orbitals on the first, second and third Carbon fragments. A full definition of all SFOs in terms of the constituting Fragment Orbitals is given in an early part of the output.

**Summary of LT or IRC path(s)**

At the very end of the results section, a completed LT or IRC calculation will show tables of a few key properties in each point of the scanned path: atomic coordinates, energy, dipole moment, atomic charges and a few others, depending on the case. This gives you a quick survey of the computed profile.

**Frequencies Results**

In a Frequencies calculation the computed harmonic frequencies are printed. If a complete variation of coordinates has taken place, the program will compute the frequencies and normal modes also in terms of Symmetry Coordinates, along with the representation in the coordinates that were specified in input.

The zero-point energy is printed, computed as sum over frequencies:

\[ E_0 = \text{Error!} \]  

(4.2.1)

Any imaginary frequencies (printed in the output file as *negative* frequencies) are not included in the summation.

Thermodynamic properties (Heat Capacity, Entropy, Internal Energy) are printed, based on the ideal gas approximation. Electronic contributions are omitted. These are small when the energy gap with the next electronic configuration is large compared with the vibrational frequencies. For (near) degenerate configurations this assumption is incorrect.

Imaginary frequencies and very small frequencies (below 20 cm\(^{-1}\)) are ignored in the calculation.
Exit Procedure

NORMAL TERMINATION or an error message.

A list of all files that are (still) open when the exit routine is called. The program closes such files at this point. Information about buffered I/O processing during the calculation.

A check of workspace to see whether all dynamically allocated arrays have been cleaned-up. If so the program mentions All Arrays Delocated. Otherwise there is something wrong and the situation will be summarized. If the calculation seems to have completed normally, but nevertheless workspace has been found not-clean, we would appreciate to get the complete output file because it might signal a programming error. This does not apply when you have used the STOPAFTER feature: the program will then abort before the standard termination and usually not all workspace will have been cleaned up then.

Timing Statistics: a survey of CPU, System (I/O) and Elapsed times spend in various sections of the program.

Logfile

At the end of the calculation the log file is copied (optionally, see PRINT) to the tail of the standard output file. The log file contains a concise summary of the run.
4.3 LOG FILE

The log file (logfile) is generated during the calculation and flushed after (almost) each message that is sent to it by the program. Consequently, the user can inspect it and see what is going on without being delayed by potentially large system I/O buffers. Each message contains date and time of the message plus additional info.

A major part of the messages simply states the name of a procedure. Such messages are sent when the procedure is entered. During the SCF procedure, the SCF errors, which are a measure for non-self-consistency, are written at every cycle. In calculations where the geometry is changing (optimization, frequencies...) each set of new coordinates is sent to the log file (Cartesian, in angstrom and also Z-matrix, if a Z-matrix structure was provided in the input file). Other messages should be self-explanatory.

Be alert on ERROR messages. Take them seriously: inspect the standard output carefully and try to understand what has gone wrong. Be also alert to warnings. They are not necessarily fatal but you should understand what they are about before being satisfied with the results of the calculation. Do not ignore them just because the program has not aborted: in some cases the program may not be able to determine whether or not you really want to do what appears to be wrong or suspicious. If you believe that the program displays erratic behavior, then the standard output file may contain more detailed information. Therefore, in such case save the complete standard output file, together with the logfile, in case we need these files for further analysis.
4.4 TAPE21

TAPE21 is the general result file of an ADF calculation. It is a KF file: Direct-Access, binary, and keyword driven. It contains information about the calculation. You can use it as a fragment file in a subsequent calculation on a bigger molecule, where the current one may be a part, or in an analysis program.

The contents of TAPE21 is keyword-accessible and you may use the KF utilities (see the UTILITIES document) for conversion of TAPE21 from binary to ASCII format and vice versa. This facility is also useful when you intend to use a TAPE21 result file produced on one type of hardware, for a continuation run on a quite different computer: Transform the binary file to ASCII format with the KF utilities on the first machine. Then transport the ASCII file to the other machine, and make a binary out of it again.

Another utility (pkf) can be used to obtain a summary of the contents of TAPE21. The output should be more or less self-documenting: all variables are listed by name, type (integer, real, ..) and size (number of array elements) and grouped in named sections.

The data on TAPE21 is organized in Sections which group together related data. Each section contains a number of variables. Each variable may be an array or a scalar and may be integer, real, logical or character type.

A complete dump of the contents of TAPE21 is obtained with dmpkf. The resulting ASCII file contains for all variables on the file:

1. The name of the section it belongs to;
2. The name of the variable itself;
3. Three integers coding for the data of the variable:
   a) The number of data elements reserved on the file for the variable;
   b) The number of data elements actually used for the variable. In virtually all cases the number of used elements is equal to the number of reserved elements. The number of used elements is relevant for interpreting the data, the number of reserved elements has only relevance for the management of data on the file by KF-specific modules and utilities;
   c) An integer code for the data type: 1=integer, 2=real, 3=character, 4=logical;
4. The variable value(s).

A typical case of the contents of TAPE21 obtained by dmpkf operating on the binary TAPE21 file from an optimization run on H2O would be:

<table>
<thead>
<tr>
<th>contents of TAPE21</th>
<th>comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>General file-ident</td>
<td>name of (first) section</td>
</tr>
<tr>
<td>6 6 3</td>
<td>name of (first) variable in the current section (General)</td>
</tr>
<tr>
<td>6 6 3</td>
<td>characteristics of the data: 6 elements reserved on file for the variable, 6 data elements actually used, 3=integer code for the data type: character</td>
</tr>
</tbody>
</table>
Contents of TAPE21

Follows a survey of the sections and variables on TAPE21. Details may differ between different run types (SinglePoint, Frequencies...). Most items should be self-explanatory. Some are only significant for internal proceedings of the program and will not be explained in detail. The sections are described in an order that corresponds to the order in which they are generated and hence printed by the KF utility programs. However, the order of generation depends somewhat on the type of application, so some difference may be found when comparing to your own TAPE21 printout.

Note that variable and section names may contain spaces: these are significant.

A special section is the “SUPERINDEX” section, which is in fact a table-of-contents that lists all the sections in the file, with technical information regarding their position on the file, the amount of data corresponding to that section and similar items. The SUPERINDEXXX section is not discussed further here. See the KF documentation for more details.

Section General

General information about the calculation and the file

fileident
Name of the file. Here: TAPE21

jobid
ADF release number with date and time of the calculation

title
Title of the calculation. This may have been set in the input file, or be internally generated. In a create run it is picked up from the Create database file (if no input value for the TITLE key has been given).

runtype
The type of calculation, for instance SinglePoint or Frequencies
nspin
   1 for a spin-restricted calculation, 2 for spin-unrestricted

nspinf
   Similar for the fragment occupation numbers as they are used in the calculation, See the key
   FRAGOCCUPATIONS

ldapot
   An integer code for the applied LDA part of the XC potential functional used in the SCF. 1 for VWN, 2 for
   VWN+Stoll...

xcparv
   X-alpha parameter value. Only relevant for the X-alpha LDA potential, meaningless if another LDA potential
   functional has been selected.

ldaen
   As for ldapot: integer code for the LDA part of the Density Functional, now however pertaining to the
   (post-SCF) energy evaluation. Usually ldaen and ldapot are identical. See the key xc for details.

xcpare
   As xcparv, but now for the energy evaluation.

ggapot
   Specification (string) of the GGA part of the XC potential used in the SCF, for instance “Becke Perdew”. If no
   GGA potential is applied, the string ggapot is empty.

ggaen
   Similar for the GGA part of the XC energy evaluation

iopcor
   Code for usage of frozen core: 1=use frozen cores, 0=pseudopotentials. Pseudopotentials are not supported
   anymore in ADF, so this variable must always be 1.

electrons
   The number of valence electrons
   Note that this is not necessarily the same as what may consider, chemically, as the valence space. Rather, it
   equals the total number of electrons in the calculation minus the electrons in the frozen core orbitals.

unit of length
   Transformation factor between input-used geometrical units (for distances) and atomic units (bohr). If input
   of, say, the atomic coordinates is in Angstrom, the unit of length is approximately 1.89

unit of angle
   Similar for angles. Internal units in the program are radians. Input (bond and dihedral angles) may be in
   degrees, in which case the unit of angle equals approximately 0.017

Section Geometry

Geometrical data such as number of atoms, coordinates, etc: Most variable names should be self-explanatory

grouplabel
   Point group symmetry (string) used in the calculation, for instance O(H). This may be set in the input file.

Geometric Symmetry
   Auto-determined (“true”) symmetry (considering the nuclear frame and any external fields, but not taking into
   account any user-defined MO occupation numbers and hence the electronic charge distribution.
symmetry tolerance
Threshold for allowed deviation of input atomic coordinates from symmetry to be detected or verified.

orient
Affine transformation (3,4 matrix: rotation and translation) between the input coordinates and the frame in which the program processes the atoms. ADF has certain orientation requirements for all supported point group symmetries and may rotate and translate the input coordinates accordingly.

oinver
The inverse transformation of orient

lrotat
A logical flag to signal whether or not a rotation has been applied between the input frame and the internally used frame.

definitions

nr of fragmenttypes
The number of distinct types of fragments

nr of dummy fragmenttypes
Idem, but counting only dummy atom fragments. A dummy fragment, if it exists, must consist of one single (dummy) atom.

fragmenttype
Names (string) of the fragment types.

fragment mass
Sum of atomic masses in the fragment.

fragment charge
An array with 3 values per fragment type (nftypes,3): 1=sum of nuclear charges, 2=sum of effective nuclear charges (discounting for the frozen core shells), 3=nr of valence electrons

fframe
Signals whether or not special local coordinate frames are used for the atoms. Usually this is not so, in which case the variable has the value DEFAULT. fframe is an array that runs over the atoms. See the “z=” option to the data in the ATOMS input key block.

cum nr of fragments
An array (0:nftyps) that gives the total number of fragments for the fragment types up to and including the indexed one. The ordering of fragments and fragment types is printed in the standard output file.

nr of fragments
The total number of fragments in the calculation
This equals the last element of the previous variable “cum nr of fragments”

nr of dummy fragments
The total number of fragments that each consist of a single dummy atom.

fragment mapping
Affine transformation matrices (3,4: rotation and translation), one for each fragment in the molecule, that transform the fragment coordinates as they are on the fragment file(s), to the actual position of the fragments in the molecule.

cum nr of atomtypes
An array (0:fragmenttypes) that counts the number of atom types up to and including the indexed fragment type.

nr of atomtypes
Total number of atom types in the molecule. Must equal the last element of the “cum nr of atomtypes” array.
nr of dummy atomtypes

Similar, now counting only the atom types consisting of a dummy atom.

atomtype

Names (strings) of the atom types

mass

Atomic masses: array running over the atom types. Compare “fragment mass”.

charge

Similar as for “fragment charge”, but now the values per atom type.

cum nr of atoms

An array (0:atomtypes) that counts the number of atoms up to and including the indexed atom type.

nr of atoms

Total number of atoms. Must equal the last element of the array “cum nr of atoms”.

nr of dummy atoms

Total number of dummy atoms

atmcrd

Type of atomic coordinates in input: CART (Cartesian) or ZMAT (Internal).

kmatrix InputOrder

The connection matrix listing (and referencing) the atoms in the order as they were in the input file. This ordering aspect is significant because internally the program reorders the atoms and groups them together by atom type and fragment type. Hence it is relevant to know what ordering (input- or internal-) is assumed in data arrays.

zaxis

For each atom the direction of the local z-axis. Normally this is identical to the standard (0,0,1), but it may be different for analysis purposes. See the “z=” option to the data records in the ATOMS block.

fragment and atomtype index

An integer array (natoms,2) that specifies for each atom the fragment and the atom type it belongs to.

atom order index

An integer array (natoms,2) that defines the re-ordering of atoms between the list in the input file and the internally used list (which is driven by fragment types, fragments, atom types; dummies come last). The internally used list can be derived from the printout of the fragments, early in the standard output.

kmatrix

The connection matrix using the internally applied ordering of atoms

xyz

Cartesian coordinates of the atoms, in the internally used ordering of atoms

xyz InputOrder

Similar, but now for the ordering of atoms as in the input file.

zmatrix

Internal (Z-matrix) atomic coordinates

zmatrix InputOrder

Internal coordinates in the input-order of atoms

Atomic Distances

Inter atomic distance matrix

ntyp

Number of atom types, not counting dummy atoms,
nqptr
A cumulative counting array, very similar to “cum nr of atoms”
Differences: it runs only over “ntyp” atom types (not including dummy atoms) and its indexing as well as its values are shifted by one: nqptr(k) is the total number of atoms plus one, counting the atom types up to and including #(k-1)

nnuc
Total number of non-dummy atoms

qtch
Nuclear charges of the non-dummy atoms

qeff
Effective nuclear charges (subtracting charge for the frozen core shells) of the non-dummy atoms

nfragm
Total number of non-dummy fragments

nofrag_1
Integer array specifying for each non-dummy atom the fragment it belongs to.

nofrag_2
Integer array specifying for each non-dummy atom the fragment type it belongs to

nuclab
Names of the non-dummy atom types.

Section Fragments
(To be completed)

FragmentFile
Names of all used fragment files

FragRun Ident,Title
Job identification and title of each fragment run that is used in the current molecule

Section AtomTypes
(To be completed)

Section Properties
AtomCharge Mulliken
Atomic charges derived from Mulliken population analysis.

Dipole
Dipole moment in atomic units.

FragmentCharge Hirshfeld
Fragment charges derived from Hirshfeld analysis

AtomCharge_initial Voronoi
Atomic charges derived from Voronoi analysis for the initial (sum-of-fragments) charge density
AtomCharge\_SCF Voronoi
Similar as the previous item, but now for the SCF density

Electrostatic Pot. at Nuclei
Coulomb potentials at the positions of the atoms, not including the contribution from the nucleus itself

Section Basis

Information about the (valence) basis set

nbset
The total number of basis “sets”, where a “set” here means a Cartesian function set (3 for a p-type function, 6 for a d-type function, and so on), given by an entry in the “list-of-basis-functions” in the data base file.

nbaspt
Cumulative number of basis sets (see previous variable, for “set”), on a per atom type basis. Only non-dummy atoms (type) are considered. nbaspt(k) is 1+nr-of-basis sets up to, but not including atom type #k

nqbas
Main quantum number of each basis set. A 1s function has nqbas()=1

lqbas
Angular momentum quantum number of each basis set. The current implementation of ADF supports only l, p, d, and f basis functions, so the allowed lqbas values are 0, 1, 2, and 3

alfbas
The exponential decay parameters of the STO functions in the basis set

basnrm
Normalization coefficients for the basis sets

naos
The total number of basis functions, counting all Cartesian polynomials and all copies of the functions on the atoms of the pertaining atom type

nbos
The total number of Cartesian basis functions, not counting the copies of the functions on the different atoms of the atom type: the functions are defined per atom type and are (for nbos) counted only once.

The next few variables relate to lists of basis functions that run from 1—nbos: all the Cartesian polynomials, but counting the function only once per atom type. Essentially, this means counting all functions with distinct characteristics (apart from their geometrical center).

nbptr
Index array of the nbos functions, where the entries are the cumulative numbers of functions (+1) up to, but not including the atom type. The size of the array is (ntyp+1): one plus the number of (non-dummy) atom types.

kx
Powers of x of the nbos Cartesian STO basis functions

ky
Powers of y of the nbos Cartesian STO basis functions

kz
Powers of z of the nbos Cartesian STO basis functions

kr
Powers of r of the nbos Cartesian STO basis functions

168
alf
  Exponential decay factors of the nbos Cartesian STO basis functions
bnorm
  Normalization factors for the nbos Cartesian STO basis functions
nptra
Consider a list of all (naos) Cartesian STO basis functions, including copies of the functions on all atoms of
the same atom type. Build that list by first taking all true valence functions on all atoms (loop over atom
types, inner loops over atoms, inner loop over basis sets of the atom type, inner loop over Cartesian
polynomials for the function set), then all auxiliary core-orthogonalization functions (similar loop structure).
nptta(i) gives the index in that list of function #i, where i corresponds to a similar list of all naos functions in
which the core and valence subsets are not separated.
norde
  An array that runs over the non-dummy atom types. Each element gives the maximum of the main quantum
  number for all STO basis and fit functions corresponding to that atom type.
lorde
  As norde, but lorde applies to the angular momentum quantum numbers.

Section Core
Information about frozen core orbitals and the Slater-type exponential functions used to describe them.
nrcset
  The number of STO function sets to describe the frozen core orbitals in the calculation. The array is sized
  (0:llqcor,1:ntyp). llqcor is the maximum l-value in core orbitals (3), ntyp is the number of non-dummy atom
types.
nrcorb
  An array (0:llqcor, 1:ntyp) specifying the number of frozen core orbitals per l-value and per non-dummy atom
type.
ncset
  The total number of core expansion STO function sets, not counting copies on all atoms, and not counting the
  Cartesian polynomials (1 value per p-set, et cetera)
ncorpt
  Index array: 1 + cumulative number of core expansion sets up to, but not including, the indexed atom type.
The array runs from 1 to ntyp+1
nqcor
  Main quantum numbers for the core expansion sets
lqcor
  Angular momentum quantum numbers for the core expansion sets.
alfcor
  Exponential decay factors for the core expansion sets.
cornrm
  Normalization factors for the core expansion sets.
ncos
  Total number of core expansion functions, counting all copies on different atoms of each atom type, and
counting all Cartesian polynomials.
nccept
Index array: 1 + cumulative number of core orbitals, counting all copies on different atoms and all Cartesian (sub) functions.

ncptr
Similar, but applying to the STO core expansion functions.

ccor
All core expansion coefficients, which express the core orbitals in the core expansion functions. The array stores the expansion coefficient sequence for each core orbital shell (not for each Cartesian sub function) and only one sequence per orbital per atom type (no duplication for the different atoms of the atom type).

npos
An index array. For each atom type: the index where its data are stored on the TAPE12 core data file. npos(k) may be zero if no data for atom type #k are available on TAPE12.

kcos
The total number of core expansion functions, like ncos, but now counting only the truly independent functions. For instance: 5 functions per \( d \)-set, while in ncos there are 6 functions per \( d \)-set. The \( s \)-type combination in the 6-membered \( d \)-set is in the calculation projected out and does not represent a degree of freedom.

s
The \((kcos,kcos)\) overlap matrix of the core expansion functions. Note that, since the dimension is \((kcos,kcos)\), the \( s \)-type combination has been eliminated, and likewise for the 3 \( p \)-type functions in each \( f \)-set.

idfcor
Integer that indicates whether or not the core set contains \( d \)- and/or \( f \)-type functions. 1=yes, 0=no

nd
Total number of \( d \)-type core orbital sets (not counting the Cartesian sub functions)

nf
Total number of \( f \)-type core orbital sets (not counting the Cartesian sub functions)

ndorb
An array running over the \( d \)-type core orbital sets (loop over atom types, loop over atoms, loop over core orbitals with \( l=2 \)). It gives for each the index of the orbital (the first of the Cartesian subset) in the overall list of all core orbitals in the molecule (including the spurious \( s \)-type functions in the \( d \)-sets, and so on)

nforb
Similar as ndorb, but now for the \( f \)-type core orbitals.

cmat
Overlap matrix between core-orbitals (ncos, counting all Cartesian functions including the \( s \)-type function in each \( d \)-set, et cetera), and the basis functions. In the list of basis functions, all core functions (the auxiliary orthogonalization functions) come before all true valence basis functions, see array NPRTA.

Section Fit
This section stores information about the fit functions, which are used for the Coulomb potential evaluation.

Unrestr.SumFrag
A logical that flags whether or not the fit coefficients have been set and stored for the sum-of-fragments, but adjusted for the unrestricted fragments option (see the keys UNRESTRICTEDFRAGMENTS, MODIFYSTARTPOTENTIAL).
coef_SumFrag
   Fit coefficients pertaining to the sum-of-fragments charge density.
coef_SCF
   SCF fit coefficients.
nfset
   Total number of fit function sets (not counting the Cartesian sub functions, not counting the copies of the functions on the atoms of an atom type)
nfitpt
   Index array: 1+the total number of fit function sets up to, but not including, the indicated atom type.
nqfit
   Main quantum numbers of the fit sets
lqfit
   Angular momentum quantum numbers of the fit sets
alffit
   Exponential decay factors of the STO fit sets.
fitnmr
   Normalization factors for the STO fit sets.
nfos
   Total number of Cartesian fit functions, not counting copies on all atoms of an atom type, but including all (for instance, 6 for a d-set) Cartesian sub functions.
nfptr
   Index array: 1+ total number of Cartesian (see variable nfos) fit functions, up to but not including the indicated atom type.
nprimf
   Total number of Cartesian ("primitive") fit functions, counting also the copies on all atoms of each atom type.
nsfos
   The total number of fully symmetric (A1 symmetry) fit function combinations that represent the true dimension (variational freedom) of the space of fit functions in the calculation.
nalptr
   Index array, like nfptr, but applying to the nsfos symmetric function combinations.
niskf
   This refers to an atom-limited symmetry combination of primitive fit functions, in the code and some documentation indicated as a “g”. A “g” is the specific part of a molecule-wide A1 fit function combination (see nsfos) that consists of all the terms that are centered on one particular atom. The number niskf gives the total number of such “g” function combinations.

To clarify this, consider an A1 fit function combination in the molecule. Assume, that it consists of a specific linear combination the following functions: a p-x function on atom A, its partner p-y function, and the corresponding p-x and p-y functions on atom B. (Atoms A and B must be symmetry equivalent). In this example we have one A1 function (in the list of nsfos such functions) and two “g”s. Each “g” consists of a p–x and a p-y function combination on a specific atom.
iskf
   Compound index array. It runs over the niskf “g” fit function combinations and has 4 entries for each function (1:4:1:niskf). The meaning of the entries is as follows. #1 = number of the fit set (not counting the copies of fit functions on different atoms of an atom type, and not counting the Cartesian sub functions) this “g” belongs
to. #2=index where the combination coefficients for this “g” start in the arrays cofcom and numcom (see next). #3=number of terms in the expansion of this “g”. #4=number of the molecular fit A1 function combination this “g” belongs to.

**nalcof**

Length of the arrays numcom and cofcom, see next

**numcom**

Numcom (and cofcom) consists of a sequence of smaller sub arrays. Each sub array gives the expansion of a “g” function in terms of the Cartesian functions in the pertaining fit function set. The elements of numcom specify the particular Cartesian sub functions that participate in the expansion. Its values are therefore limited to lie between 1 and (L+1)(L+2)/2, where L is the maximum l-value occurring in the fit function sets.

**cofcom**

Compare numcom: cofcom gives the actual expansion coefficients for the expression of a “g” function in primitive Cartesian fit functions.

**Section Num Int Params**

Numerical integration parameters: the general precision parameter, but also more technical parameters used by the grid-generating modules.

**method**

Label of the method used to generate the grid. Usually: “polyhedra”

**accint min**

Minimum integration precision parameter. It is the lower bound of the range in which the value of the actual numerical integration precision parameter may vary.

**accint max**

Maximum value of the precision general parameter

**accint**

Actual value of the precision parameter. This variable governs by default almost all other integration parameters.

**ldim**

In fact, this a geometric parameter: the number of dimensions in which the system is periodic. For molecules this is zero.

**PointChargeTypes**

The number of point charges types used in the calculation. Point charges belong to a different point charge type if, and only if, their strengths are not equal.

**accsph**

The precision parameter that determines the (radial) integration grid in the atomic spheres

**accpyr**

The precision parameter that determines the general precision level of the grid in the atomic polyhedra

**accout**

The precision parameter that determines the general precision level of the grid in the outer region

**accpyu**

The precision parameter that determines the 1D grid along the first direction in the quadrangles and triangles of the bases of the atomic pyramids
accpyv
The precision parameter that determines the 1D grid along the second direction in the quadrangles and triangles of the bases of the atomic pyramids

accpyw
The precision parameter that determines the 1D radial integration in the atomic pyramids, between the atomic sphere surfaces and the pyramid basis

frange
Estimated maximum range of functions, to determine how far the integration grid has to extend outwards, away from the molecule

rspher
An array with the radii of the atomic sphere (a value per atom type)

rsph0
The smallest sphere radius

rsphx
The largest sphere radius

dishul
The distance between the innermost boundary planes, which separate the atomic pyramids from the outer region, and the surfaces of the outermost atoms

nouter
The number of intervals in which the outward (radial) integration in the outer region is broken up

outrad
The precision parameter that determines the outward radial integration in the outer region

outpar
The precision parameter that determines the 2D integrals in the outer region parallel to the boundary planes

linteg
An array with maximum angular momentum quantum numbers (one value per atom type), to determine the angular integration grid in the atomic spheres

lintgx
Maximum of linteg()

linrot
Angular momentum quantum number to determine the rotational integration parameter around the molecular axis (in linear molecules only)

ntyps
The number of atom types as seen by the numerical integration grid generator. This means in practice: the number of non-dummy atom types plus the number of point charge types.

nnucs
The number of atoms as seen by the numerical integration grid generator. This means in practice: the number of non-dummy atoms plus the number of point charges.

qatm
Nuclear charges for all ntyps atom types

nratst1
The numerical integration grid generator automatically determines the symmetry of the nuclear (nnucs atoms!) frame and then puts the atoms in sets of symmetry equivalent ones. nratst1() is an array (0:ntyps) that contains
the cumulative number of atoms in the symmetry sets. $\text{nratst1}(k)$ is the total number of atoms in the sets up to and including set $#k$

$\text{xyzatm}$
Cartesian coordinates of the atoms.

$\text{linteg all}$
Similar to array $\text{linteg}()$, extended to include also the point charge types

$\text{npowx}$
Maximum power of the radial variable $r$, in the set of test functions that the grid generator uses to tune the grid

$\text{alfas}$
An array that stores the exponential decay factors of all test functions, ordered by atom type and by the power of the radial variable $r$.

**Section Symmetry**

Symmetry related data.

$\text{nogr}$
The number of symmetry operators in the point group used in the calculation. NB, for the special cases of infinite symmetries, only the operators corresponding to finite elements are counted. Therefore, ATOM has $\text{nogr} = 1$ (only the unit operator); $C(\text{LIN})$ has $\text{nogr} = 1$, $D(\text{LIN})$ has $\text{nogr} = 2$.

$\text{faith}$
An array that stores all the (3,3) symmetry operator matrices in the real space representation

$\text{nsetat}$
The number of sets of symmetry equivalent atoms under the used symmetry

$\text{napp}$
An array that stores for each atom the number of the symmetry set it belongs to

$\text{notyps}$
An array that stores for each set of symmetry equivalent atoms, the atom type to which the set belongs

$\text{noat}$
Map between the normal list of atoms and the symmetry sets. When you loop over the symmetry sets and, inside, loop over the atoms in each set, you thereby run over the index of $\text{noat}()$. The value points to the position of that atom in the original (not set-ordered) list.

$\text{ntr}$
An array ($\text{nogr}$,$\text{nnuc}$) that stores for the each atom $A$ and each symmetry operator $R$, the atom onto with $A$ is mapped by $R$. The row index runs over all symmetry operators, the column index over the atoms.

$\text{npeq}$
The number of symmetry unique pairs of atoms

$\text{jjsym}$
An array that runs over the $\text{npeq}$ sets of symmetry equivalent atom pairs. Its value gives for the indicated set the index of a (c.f. the first) atom pair in that set.

$\text{jasym}$
An array that runs over the $\text{npeq}$ sets of equivalent atom pairs. Its value gives for the indicated the set the number of pairs in that set.
jalok
An array (1:npeq), with values 0 or 1. 1=the pair density can be fitted using A1 fit functions only. 0=all fit functions (on the involved atoms) are to be used. The value 1 may arise because of symmetry properties, or because the distance between the atoms is so large that the inaccuracy from using only A1 fit functions can be neglected.

ntr_setat
A condensed variety of array ntr: the columns are not the atoms, but the nsetat sets of symmetry equivalent atoms. The value is the index of the atom, onto which a representative (c.f. the first) atom of the indicated symmetry set is mapped by the given symmetry operator.

igr
A code that fixes, together with nogr and ngr, the point group symmetry. See the header of routine adf/maisya for a list

ngr
One of the code components that fix the symmetry group. See routine adf/maisya

grouplabel
Schoenfliess symbol as used in ADF

nsym
The number of symmetry representation (including subspecies) used in the calculation.

norb
For each of the nsym representations the number of basis function combinations (SFOs) that belong to it.

nfcn
For each of the nsym representations the number of primitive atom centered basis functions that participate in the representation.

ncbs
For each of the nsym representations the number of core orthogonalization functions that participate in the representation.

jsym
For each of the nsym representations: if it belongs to a one-dimensional irrep, the value is 1, otherwise: for the first subspecies in the irrep the value is the dimension of the irrep, for the other subspecies in the same irrep the value is 0

symlab
For each of the nsym representations the label (string) of the representation

norbc
An array (-2:2,nsym). The column runs over the symmetry representations. The positive row indices (1,2) specify for spin-A and spin-B (the latter only if the calculation is spin-unrestricted), the highest non-empty orbital. The negative indices (-1,-2) specify for spin-A and spin-B (if the unrestricted fragment option is used) the total number of non-empty SFOs. The zero row index specifies the number of non-empty SFOs, before applying any fragment occupation changes.

Section Spin_orbit
(To be completed)
Section Energy

XC energies
16 elements of an array enxc\((2, 2, 4)\): exchange-correlation energies of various charge densities:
first index: 1=exchange term, 2=correlation term
second index: 1=LDA term, 2=GGA term
third index: 1=energy of fragments (summed over fragments), 2=energy of sum-of-fragments density,
3=energy of orthogonalized fragments, 4=SCF.

Pauli TS Correction (LDA)
Correction to the “Transition State” method to compute terms in the bonding energy, in this case the Pauli exchange energy term. The Pauli TS Correction is not separately printed in the standard output file, but included in the Pauli interaction term.

Pauli FitCorrection
The first-order correction to the Pauli exchange interaction term, for the error in the Coulomb energy due to the fit incompleteness. This correction term is not printed in the output file but included in the Pauli interaction term.

Elstat Core terms
An obsolete variable, not used in the energy computation.

Elstat FitCorrection
The first-order correction to the electrostatic interaction term (putting the fragments together, without any relaxation of Pauli orthogonalization), for the error in the Coulomb energy due to the fit incompleteness.

Orb.Int. FitCorrection
The first-order correction to the electrostatic interaction term in the SCF relaxation energy (Orbital Interactions), for the error in the Coulomb energy due to the fit incompleteness. This term is not printed (anymore) separately, but incorporated in the symmetry-specific interaction terms.

Orb.Int. TSCorrection (LDA)
The difference between the representation-specific orbital interaction terms added, and a straightforward computation of the SCF relaxation energy is the result of the neglect of higher order terms in the Taylor expansion that underlies the ‘Transition State’ method. This difference, therefore, corrects exactly this neglect. It is not printed separately anymore in the output, but incorporated in (distributed over) the representation-specific orbital interaction terms.

Ebond due to Efield
Bond energy term due to any homogeneous electric field.

Corr. due to Orthogonalization
For analysis purposes, the concept of “orthogonalized fragments” has been introduced and the bonding energy is split in a part that describes the difference between the sum-of-fragments situation and the orthogonalized-fragments density at one hand, and the SCF relaxation (from the orthogonalized fragments density) at the other. Both terms contain a first order fit correction term. The result of adding the two parts is not identical to computing the total bonding energy directly and applying the first order correction to that approach. The difference is given by this term, which therefore corrects for the additional second order fit errors caused by using the orthogonalized fragments split-up.
**SumFragmentsSCF FitCorrection**

The ‘true’ first order fit correction for the complete bonding energy, resulting from a direct calculation that takes the sum-of-fragments as starting point and the SCF as final situation, without the intermediate step of orthogonalized fragments.

**Pauli Efield**

The contribution to the Pauli interaction energy due to any electric field

**Orb. Int. Efield**

The contribution to the SCF relaxation energy (orbital interactions) due to any electric field

**Electrostatic Interaction**

The electrostatic (sum-of-fragments compared to isolated fragments at infinite distances) Coulomb interaction energy, without any fit correction (if applicable)

**Pauli Total**

The Pauli exchange (orbital orthogonalization) interaction energy

**Steric Electrostatic**

The electrostatic interaction energy including any first order fit correction (if computed from the fit density)

**Steric Total**

The total steric interaction energy, consisting of the electrostatic and the Pauli interactions

**Orb. Int. ?**

? stands for one of the irreps of the point group symmetry. The value gives the orbital interaction (SCF relaxation) term for that symmetry representation

**Orb. Int. Total**

The total orbital interaction energy

**SCF Bond Energy**

Total bonding energy

**elstat**

Electrostatic interaction energy. Same as the “Electrostatic Interaction” variable in this section

**Bond Energy**

Total bonding energy, same as the “SCF Bond Energy” variable

**Pauli Kinetic**

Kinetic energy term in the Pauli exchange interaction energy

**Pauli Coulomb**

Coulomb energy term in the Pauli exchange interaction energy

**Pauli Kinetic+Coulomb**

Sum of the kinetic and Coulomb terms in the Pauli exchange interaction energy

---

**Section Point_Charges**

**NumberofPointCharges**

The total number of point charges used

**PointCharges**

The array with point charge values: (4,np), where np is the number of point charges and the 4 components are, respectively, the x y z components and the strength.
**Section GeoOpt**

Optimization data.
Where references are made to the list of atoms, the atoms are assumed to be in internal order. This may be
different from the input-list of atoms.

nfree
number of independent optimization variables

idfree
indices (3.nr-of-atoms) for all atomic coordinates referring to the optimization variables (values 1..nfree)
and/or LinearTransit parameters (values nfree+k, k being the k-th LT parameter). A zero value means that the
coordinate is frozen.

all freedoms
A logical the flags whether or not all fundamental degrees of freedom in the system are allowed to vary. This
is not the case when constraints are applied.

Gradients
The most recent values for the derivatives of the energy with respect to the atomic coordinates (cartesian or
Z-matrix, depending on the type of optimization variables).

Displacements
The most recent step executed for the atomic coordinates (optimization variables)

kmatrix
The connection matrix.

Hessian_CART
The Hessian matrix (second derivatives) as a n*n matrix, in the Cartesian coordinates representation. Note
that the reduced storage mode (typically, Fortran upper-triangular) is not applied.

Hessian_ZMAT
Same, but now in the internal coordinates representation

Hessian inverted_CART
The inverted Hessian, in Cartesian coordinates

Hessian inverted_ZMAT
Likewise, in internal coordinates

Note: in most cases only one, or maybe two of the Hessian cases are present on TAPE13. They can be
transformed into each other quite easily.

xyz old
cartesian coordinates at previous geometry cycle

zmatrix old
idem for internal coordinates

**Section TS**

Information about the Transition State search
modtrc
Defines the initial search direction. Positive value \( n \): the \( n \)-th Hessian eigenvector (default: 1). Negative value \( n \): the Hessian eigenvector with the largest (absolute value) component in the \( n \)-th optimization variable

itrace
Index of the Hessian eigenvector that is being followed

neghes
The assumed number of negative eigenvalues of the Hessian at the Transition State. Should be 1: searches for higher-order transition states are not supported.

mode to follow
Direction vector in atomic coordinates (Cartesian or Z-matrix, depending on the variable geocrd) that corresponds to the current estimate of the unique Hessian eigenvector with negative eigenvalue

**Section LT**

Information about the Linear Transit calculation

nr of points
The total number of LT points to be computed.

current point
Index of point that is currently computed

Energies
Energy values in the LT points

Dipole
Dipole moments in the LT points

Parameters
LT parameters, initial and final values (along the path, the values are obtained by even-spaced linear interpolation)

atmcrd
ZMAT if a Z-matrix structure (connection matrix) is available. CART otherwise. Used for printing

geocrd
Type of coordinates to optimize and scan along the path (CART or ZMAT)

xyz
Cartesian coordinates in the LT points (3,natoms,nlt)

zmatrix
Internal coordinates in the LT points

AtomCharge Mulliken
Mulliken atomic charges in the LT points

FragmentCharge Hirshfeld
Hirshfeld fragment charges in the LT points

AtomCharge_initial Voronoi
Voronoi atomic charges corresponding to the sum-of-fragment densities in the LT points

AtomCharge_SCF Voronoi
Voronoi atomic charges corresponding to the SCF densities in the LT points
**Section IRC**

This section contains general information about the IRC (Intrinsic Reaction Coordinate) calculation. Details of the computed reaction path are in sections IRC_Forward and IRC_Backward.

**atmcrd**

ZMAT is a Z-matrix structure (connection matrix) is available. CART otherwise

**geocrd**

CART or ZMAT: the type of coordinates to change, optimize and trace

**PointStatus**

A string status variable of the current IRC point. Value can be “DONE” (if the point has been computed), “EXEC” if its computation has not yet finished.

**nfree**

Number of optimization coordinates that can be varied. See section GeoOpt

**idfree**

(3,natoms) pointers to the optimization variables for each of the atomic coordinates. A zero means: frozen by constraint

**xyz**

Cartesian coordinates

**kmatrix**

Connection matrix, if a Z-matrix structure is available

**zmatrix**

Internal coordinates

**Energies**

Energy at the Transition State

**Dipole**

Dipole moment at the Transition State

**Gradients**

Computed energy gradients at the (assumed) Transition State (should be very small)

**AtomCharge Mulliken**

Mulliken atomic charges, for the TS geometry

**AtomSpinDen Mulliken**

Atomic spin densities (Mulliken) at the TS

**AtomCharge_initial Voronoi**

Voronoi atomic charges at the TS, from the sum-of-fragments density

**AtomCharge_SCF Voronoi**

Similar, for the SCF density

**modtrc**

Defines the start direction for the IRC path. A positive value \( n \) selects the \( n \)-th eigenvector of the Hessian. A value \(-1\) selects the gradient vector (which must then, of course, not be exactly zero). A value \(-2\) specifies that the start direction is specified in the input file.

**step**

Step length (in mass-weighted metric) between successive points of the IRC path.

**stepMin**

The minimum value for the step
stepMax
   The maximum value for the step
Hessian inverted_ZMAT
   Inverse Hessian in internal coordinates.
lfree
   The number of independent optimization step directions (for the restricted optimization orthogonal to the IRC path.
vfree
   Direction vectors (3,natoms,lfree) for the independent optimization directions
GradientVector
   The current gradient vector (during optimization)

Section IRC_Forward

Information about the “forward” IRC path. The choice, which direction down from the Transition State is forward or backward is arbitrary. By definition, in ADF the forward direction is in the positive direction along the first Hessian eigenvector, for which the sign convention is that the largest coefficient is positive.

PathStatus
   Status (string) variable for the “forward” half of the IRC path. May be “EXEC”, or “DONE”, “UNKNOWN”, “WAIT”, “OFF”
PointStatus
   Status variable for the current point at the “forward” path. May be “DONE”, “EXEC”
nset
   Size of arrays to store data in the IRC points along the path. Will be increased when too small
pivot
   Coordinates of the current pivot point
xyz
   Cartesian coordinates of the IRC points (3,natoms,nset)
zmatrix
   Internal coordinates of the IRC points (3,natoms,nset)
Path
   Lengths measures in mass-weighted metric along the path to the IRC points
Curvature
   Local curvature values of the path at the IRC points
Energies
   Energy values at the IRC points
Gradients
   Energy gradients at the IRC points (one value: the gradient along the path. The orthogonal components are presumably zero)
Dipole
   Dipole moments at the IRC points
AtomCharge Mulliken
   Mulliken atom charges at the IRC points
FragmentCharge_Hirshfeld

  Hirshfeld fragment charges at the IRC points

AtomCharge_initial_Voronoi

  Voronoi atomic charges at the IRC points, corresponding to sum-of-fragments densities

AtomCharge_SCF_Voronoi

  Voronoi atomic charges at the IRC points, corresponding to the SCF densities

CurrentPoint

  Integer index of the current IRC point (in the set of nset points)

step

  Current step length

Section IRC_Backward

All entries in this section match those in the section IRC_Forward. Of course, here they refer to the other half of the IRC path.

Section Freq

This section contains information about (progress) of the Frequencies calculation and results.

kountf

  An integer counter that keeps track of how many geometric displacements have been carried out to scan the potential energy surface around the equilibrium

nraman

  Integer to flag whether or not Raman intensities are computed

numdif

  Integer to determine the type of numerical differentiation (of gradients, to get the second derivative): 1=one-sided, 2=two-sided displacements.

disrad

  Size of displacements of Cartesian coordinates or bond lengths (in case of displacements in internal coordinates)

disang

  Size of displacements of angular coordinates

gecrd

  Type (string) of coordinates to displace: CART or ZMAT

atmcrd

  ZMAT if a z-matrix structure is available. This determines printed output but does not affect the computation.
  Compare the variable geocrd

nfree

  The number of degrees of freedom

idfree

  An array (3,natoms) that stores for each atomic coordinates (Cartesian or internal, depending on geocrd) the number of the (1..nfree) variational freedom it corresponds to. If zero, the coordinate is frozen by constraint.
xyz
  Cartesian coordinates of the equilibrium geometry
kmatrix
  Connection matrix that defines a Z-matrix
zmatrix
  The Z-matrix variable values of the equilibrium geometry
all freedoms
  Logical: true if all atomic coordinates are allowed to be displaced, not restricted by constraints.
nr of atoms
  The total number of atoms, including dummy’s
rigids
  Vectors of rigid motion directions, expressed in the atomic coordinates (3,natoms,6)
Dipole previous
  The dipole moment of the previous geometry. This is used to compute dipole derivatives by numerical differentiation. The “previous” geometry is the equilibrium geometry in case of one-sided displacements.
Dipole
  The dipole moment corresponding to the current geometry
Dipole derivatives
  The matrix of dipole derivatives with respect to atomic displacements
Polbty previous
  The polarizability tensor (6 elements, triangular representation) of the “previous” geometry. See the remarks about the dipole moment
Polbty
  The polarizability tensor corresponding to the current geometry
Polbty derivatives
  The matrix of derivatives (w.r.t. the atomic coordinates) of the polarizability tensor
Gradients
  The energy gradients corresponding to the current geometry
Gradients previous
  The energy gradients of the “previous” geometry. See the remarks about ‘previous’ dipole moment
Force constants
  The matrix of force constants (second derivatives), built up during the frequencies calculation.
xyz displaced
  The Cartesian coordinates of the current (displaced) geometry
zmatrix displaced
  Internal coordinates of the current (displaced) geometry
Dipole derivatives_CART
  Dipole derivatives with respect to Cartesian coordinate changes
Hessian_CART
  The Hessian matrix in Cartesian coordinates, computed at the end, when the construction of the “Force constants” has been completed.
Frequencies
  An array with harmonic frequencies.
Sections Ftyp \(n\)

\(n\) is an integer. All such sections give general information about fragment type \(n\), and more specifically about the ADF calculation that produced the corresponding fragment file.

**jobid**

Job identification of the fragment run

**title**

Title of that calculation

**nsym**

Number of symmetry representations (subspecies) used

**norb**

For each representation the size of the Fock matrix (variational degrees of freedom)

**bb**

Labels of the subspecies

**igr**

(Partial) code for the point group symmetry

**ngr**

(Partial) code for the point group symmetry

**grouplabel**

Schönfliess symbol of the point group symmetry (of the fragment calculation)

**nfcn**

An array over the representation: for each subspecies the number of primitive STO basis functions that participate in that subspecies

**jsyml**

An array (1:nsym). Value 1 means that the corresponding subspecies belongs to a 1D irrep. A value larger than 1 means a correspondingly higher dimensionality of the irrep and indicates that that subspecies is the first in that irrep. A value 0, finally, means that it is not the first subspecies in its irrep.

**nfrag**

Number of fragments used in that fragment calculation

**natom**

Number of atoms in the fragment

**naos**

Number of primitive atomic basis functions

**nrat \(l\)**

Maps the atoms of this fragment (the “1” signals the first fragment of this type) onto the list of all atoms

**rotfrg**

Rotation matrix to map the fragment coordinates as they are on the fragment file onto their actual orientation in the molecule

**nsot**

Total number of MO degrees of freedom, summation over all subspecies

**nmis**

The number of symmetry representations that could not be spanned by the basis set
mis
  Indices of the missing symmetry representations

Sections Ftyp n?

n stands for the n-th fragment type. The ? stands for one of the symmetry representations (of the point group symmetry used in the fragment calculation)

froc
  MO occupation numbers for the MOs in this subspecies

eps
  Orbital energies
  When they result from a ZORA calculation, the non-scaled values are stored on file (the scaled values are printed in the standard output file).

eigvf
  Fragment MO eigenvectors, expressed in all the primitive atomic orbitals of the fragment.

nsos  1
  Total number of MOs in this subspecies: size of variational problem

nbas  1
  Number of primitive atomic basis functions that participate in this subspecies

npart  1
  Indices that give for each of the nbas functions, the number of the basis function in the list of all basis functions

FO  1
  The fragment MOs (nbas*nsos coefficients)

nocc  1
  Number of non-empty orbitals

Section Freq Symmetry

Information about the true (possibly input-specified) symmetry of the equilibrium geometry (in a frequencies calculation). The displaced geometries may lose symmetry. Therefore, the program uses NOSYM symmetry, internally, for a frequencies calculation. The ‘true’ symmetry of the system is used for analysis purposes.

nr of operators
  Number of symmetry operators used

operators
  (3,3) matrices of the operators

nr of symmetries
  Number of subspecies

symmetry labels
  Names of the subspecies

atom indices
  List of indices to map the symmetry-ordered atoms (loop over symmetry sets, loop over atoms in each set) to the ‘normal’ list of all atoms
nr of atomsets
  Number of sets of symmetry equivalent atoms
atom mappings
  Integer array that provides mapping (back and forth) between the atom list in the input file and the internally used list, which is atom type driven
atomset indices
  The number of atoms in each of the sets of symmetry equivalent atoms
nr of displacements_X
  (X must be one of the symmetry representations.) The number of symmetry-combined atomic displacements that transform as X
degeneracy_X
  Degeneracy of X
displace_X
  The actual displacement direction vectors (3,natoms,N). N is the number of symmetry displacements for X.
nr of rigids_X
  The number of rigid motion direction vectors that transform as symmetry representation X
displ_InputOrder_X
  The displacement vectors, but now expressed in the atomic coordinates using the ordering of atoms in the input file
NormalModes_X
  Harmonic frequency normal modes in representation X
Frequencies_X
  The harmonic frequency values
IR intensities_X
  The infrared intensities

Sections X
X stands here for the label of a subspecies of the point group symmetry, for instance A1. Depending on the point group symmetry, there may be many such sections, each corresponding to one of the subspecies. All such sections have an identical structure.
nmo_A
  The number of MOs with spin-A, for which the coefficient vectors are calculated. During the SCF this may be severely reduced, at the end it is usually the complete basis in the pertaining symmetry representation.
nmo_B
  Similar for spin \( \beta \). This variable is not present in a restricted calculation.
SFO
  The definition of the SFOs in the representation, consisting of expansion coefficients in terms of the primitive atomic STO basis functions
frocf
  The occupation numbers of the SFOs in this representation
npart
  A list of indices of the BAS functions that are used in this symmetry representation
The occupation numbers of the MOs in the representation, for spin-A

Similar for spin-B, if a spin-unrestricted calculation is performed

Overlap matrix between core functions and SFOs

SFO occupation numbers

The orthogonalized fragment orbitals in the BAS representation.

The Lowdin orbitals in the BAS representation: the matrix to transform the MOs from Lowdin representation (orthonormalized SFOs) to the BAS representation

MO expansion coefficients in the BAS representation for all \( n_{mo,A} \) orbitals. The coefficients run over all BAS functions indicated by \( npart \)

Similar for spin-B, if present

The orbital energies for the \( n_{mo,A} \) orbitals of spin-A

When they result from a ZORA relativistic calculations, the non-scaled values are stored on file. (The scaled energies are printed in standard output.

Similar for spin-B, if present

MOs expressed in SFOs, for spin-A MOs

Similar for spin-B

Each such section contains the (core- and possibly also valence-) radial density and potential of one particular atom type. \( X \) is the atom type label and \( n \) is an index running over all atom types in the calculation. The list of all atom types is printed on standard output in the early geometry section.

The radial densities and potentials may be represented as simple tables – a sequence of values for \( r \), the distance to the nucleus, and the corresponding density or potential – or as a piecewise expansion in Chebyshev polynomials over a sequence of intervals \((r_1,r_2)\).

The core density and potential have been constructed from the Frozen Core orbitals, which are defined in the section Core. If a TAPE12 (COREPOTENTIALS) file has been attached to the calculation the core data is read off from that TAPE12 and stored also.

Maximum \( r \)-value for which the valence density is non-negligible
nrint val
Number of intervals for piecewise expansion of the valence density in Chebyshev polynomials
rup val
Arrays (1..nrint) of upper bounds of the intervals. The lower bound of the first interval is zero
ncheb val
Array (1..nrint) with the number of expansion coefficients for each interval
ccheb val
Coefficients of the expansion. All coefficients, for all intervals, are stored contiguously in one linear array.
The parts pertaining to a particular interval are determined by using the arrays ncheb()
nrad
Number of points used in the direct tabular representation of the atomic densities and potentials
rmin
The first r-value of the table: the radial grid is defined by a first value (rmin), a constant multiplication factor
defining r_{k+1} w.r.t. r_k (rfac, see next), and the total nr of points (nrad).
rfac
The multiplication factor of the radial grid
valence den
The valence density, in a table of nrad values.
valence pot
Similar for the Coulomb potential of the density, including a nuclear term Q/r, such that the long-range
monopole term in the potential is zero
qval
The number of electrons contained in the valence density
rx core
Maximum r-value for which the core density is non-negligible
nrint core
Number of intervals for piecewise expansion of the core density in Chebyshev polynomials
rup core
Arrays (1..nrint) of upper bounds of the intervals. The lower bound of the first interval is zero
ncheb core
Array (1..nrint) with the number of expansion coefficients for each interval
ccheb core
Coefficients of the expansion. All coefficients, for all intervals, are stored contiguously in one linear array.
The parts pertaining to a particular interval are determined by using the arrays ncheb()
qcore
The number of electrons contained in the core density
core den
The core density, in a table of nrad values.
core pot
Similar for the Coulomb potential of the density, including a nuclear term Q/r, such that the long-range
monopole term in the potential is zero
**Section LqbasxLqfitx_xyznuc**

This section will be removed again in the future. Temporarily it serves to transfer data from the calling program to the grid generator.

- **lqbasx**
  - An array with for each atom type the maximum angular moment quantum number in the basis functions for that type

- **lqfitx**
  - An array with for each atom type the maximum angular moment quantum number in the fit functions for that type

- **xyznuc**
  - Cartesian coordinates of the non-dummy atoms

**Section GenptData**

This section will be removed in the future. It serves, temporarily, to transfer data from the calling program to the numerical integration grid generator. Most of the entries here occur also in other sections but are packed together as replacement for previous common block structure.

- **numint**
  - Integer code for the type of integration grid. Usual value: 2 (polyhedra method)

- **iexcit**
  - Integer flag for excitations (response) calculation

- **lpolar**
  - Integer flag for polarizability (response) calculation

- **ldim**
  - Number of dimensions of periodicity

- **mdim**
  - Dimensionality of the molecule, for instance a linear molecule has mdim=1

- **r0mult**
  - A technical parameter that sets the radius outside which the multipole part of the fit coulomb potential functions is separated (from the exponentially decaying part), for separate treatment in the evaluation of the molecular coulomb potential.

- **avec**
  - (3,3) matrix with lattice vectors. Only the (ldim,ldim) sub matrix is significant.

- **bvec**
  - Inverse of avec (apart from a factor of 2 pi): lattice vectors in reciprocal space.

- **ngimax**
  - Maximum number of geometry optimization iterations

- **llbloc**
  - Block length determination parameter (maximum)

- **ipnbl**
  - Number of integration blocks processed by the current process
nbleqv
The number of symmetry equivalent blocks to each symmetry unique block of points. This value is 1 if any equivalent blocks are not constructed and used.

ngmax
The number of integration points, accumulated over all parallel processes

nblock
The number of integration blocks

lblock
The block length

lblx
An upper bound of the block length applied during the computation of the block length

nmax
The number of integration points generated by this process

twopi
Value of the constant 2*pi

fourpi
Value of the constant 4*pi

Section Multipole matrix elements

Information in a response calculation

dipole elements
The matrix elements of the 3 dipole operator components between occupied and virtual orbitals: outer loop over the operators (in order: y, z, x), loop over virtual MOs, inner loop over occupied MOs

quadrupole elements
Similar as for dipole. Order of operators:
\[ \sqrt{3} \times xy \]
\[ \sqrt{3} \times yz \]
\[ z^2 - (x^2 + y^2)/2 \]
\[ \sqrt{3} \times xz \]
\[ \sqrt{3} \times (x^2 - y^2)/2 \]

octupole elements
Similar as for dipole and quadrupole. Order of operators:
\[ \sqrt{10} \times y \times (3 \times x^2 \times y^2)/4 \]
\[ \sqrt{15} \times xyz \]
\[ \sqrt{6} \times y \times (4 \times z^2 - x^2 - y^2)/4 \]
\[ z \times (z^2 - 3x^2 - y^2)/2 \]
\[ \sqrt{6} \times z \times (4z^2 - x^2 - y^2)/4 \]
\[ \sqrt{15} \times z \times (x^2 - y^2)/2 \]
\[ \sqrt{10} \times z \times (x^2 - 3y^2)/4 \]

hexadecapole elements
Similar as for dipole and quadrupole. Order of operators:
\[ \sqrt{35} \times xy \times (x^2 - y^2)/2 \]
\[ \sqrt{70} \times z \times (3x^2 y - y^3)/4 \]
\sqrt{5}xy*(6z^2-x^2-y^2)/2
\sqrt{10}*(4y^2-3yz*(x^2+y^2))/4
(8x^4-24y^2(x^2+y^2)+3(x^4+2x^2y^2+y^4))/8
\sqrt{10}*(4xz^3-3xz(x^2+y^2))/4
\sqrt{5}*(x^2-y^2)*(6z^2-x^2-y^2)/4
\sqrt{70}z*(x^3-3xy^2)/4
\sqrt{35}*(x^4-6x^2y^2+y^4)/8

**Section Irreducible matrix elements**

Information in a response calculation

**Irreducible dipole elements**

The dipole matrix elements between occupied and virtual MOs, as in the section Multipole matrix elements. Here, however, the matrix elements are ordered by symmetry representations and "symmetry zeros" are omitted. The stored arrays, however, have the same size as in the previous section. See the implementation for details about the storage of this data. (Directory $ADFHOME/ADF/response/)

**Irreducible quadrupole elements**

Similar as for the dipole elements

**Irreducible octupole elements**

Similar as for the dipole elements

**Irreducible hexadecapole elements**

Similar as for the dipole elements

**Section ETS**

Technical data used in the ETS procedure.

**nff**

Size of array ncspt (next)

**ncspt**

Pointer array to find, for each atom type, the first element corresponding to that atom type’s section in the arrays ncsett, alfcst, and cfcset, see below

**ncs**

Size of the matrices ncsett, alfcst, and cfcset, see below

**ncsett**

Build a list of products of core orbital expansion functions, taking only the one-center products and looping over the atom types (not the atoms). ncsett stores the powers of the radial variable \( r \) for the products (from the main quantum numbers, one subtracted). A product of a 1s and a 2p yields ncsett()=1

**alfcst**

Similar as ncsett: the sum of the exponential decay factors of the factor functions

**cfcset**

The density matrix corresponding build from the frozen core orbitals (all atom types, but no copies for the distinct atoms of a type), in the representation of the core orbital expansion functions. Stored are, per atom
and per $l$-value (0..3) the upper-triangles of the corresponding density matrices, one after the other, all in cfcset
nnuc
The number of (non-dummy) nuclei
qcore
For each atom the number of electrons summed over its core orbitals, resulting from analytical integration of the core orbital expansions in STO core expansion functions.

**Using Data from TAPE21**

An ASCII dump of TAPE21 (complete or partial) can be obtained with the KF utility dmpkf, see the UTILITIES document. Alternatively you may build your own small program to extract any required information, using the KF library routines in the ADF package. Consult the KFS documentation for a description of this software.

**Representation of functions and frozen cores**

ADF uses the cartesian representation for the spherical harmonics part in functions:

$$f(x, y, z) = x^a y^b z^c d^e e^{-\alpha r}$$

The angular momentum quantum number $l$ is then given by $l = a + b + c$, and the main quantum number $n = l + d + 1$. There are $(l+1)(l+2)/2$ different combinations of $(a, b, c)$ for a given $l$-value, rather than $(2l+1)$. The excess is caused by the presence of spurious non-$l$ Functions in the set; a Cartesian d-set for instance consists of six functions, five of which are true d-functions while one linear combination is in fact an s-type function $(x^2+y^2+z^2)$. Only the five true d-combinations are actually used as degrees of freedom in the basis set, but lists of primitive basis functions (BAS) for instance run over all Cartesian functions including the improper ones.

A function set in ADF is characterized by the quantum numbers $l$ and $n$, and by the exponential decay factor $\alpha$. A set thus represents $(l+1)(l+2)/2$ Cartesian functions and $(2l+1)$ degrees of freedom.

The atomic frozen core orbitals are described as expansions in Slater-type functions; these are not the functions of the normal basis set but another set of functions, defined on the data files you use in Create mode. Orthogonality of the valence space to the frozen core states is enforced as follows: for each frozen core shell (characterized by the quantum numbers $l$ and $n$; all orbitals with $m=-l ... +l$ are identical apart from rotation in space) the set of valence basis functions is augmented with a so-called core orthogonalization function set. You may conceptually interpret the core orthogonalization functions as single zeta expansions of the true frozen core states. Each of the normal valence basis functions is now transformed into a linear combination of that valence function with all core orthogonalization functions, where the coefficients are uniquely defined by the requirement that the resulting function is orthogonal to all true core functions.

So the list of all Cartesian basis functions is much larger than the degree of freedom of the basis: it contains the spurious non-$l$ combinations and it contains also the core orthogonalization functions.
Evaluation of the charge density and molecular orbitals

TAPE21 contains all the information you need to evaluate the charge density or a Molecular Orbital (MO) in any point in space. Most of the information is located in section Basis:

A list of function characteristics \((kx, ky, kz, kr, alf)\), including the core orthogonalization functions. This list does not run over all BAS functions used in the molecule: if a particular function is used on the atoms of a given atomtype, the function occurs only once in the list, but in the molecule it occurs as many times as there are atoms of that type.

With array \(nbptr\) you locate the subsections in the function list that correspond to the different types of atoms: for atom type \(i\) the functions \(nbptr(i) \ldots nbptr(i+1)-1\). The distinct atom types are listed in an early section of the standard output file.

Array \(nqptr\) gives the number of atoms for type \(i\): \(nqptr(i+1) - nqptr(i)\). With this information you construct the complete list of all functions. Repeat the subsection of type \(i\) as many times as there are atoms of that type: the complete list can be considered to be constructed as a double loop, the outer being over the atom types, the inner over the atoms that belong to that type.

The total ‘overall’ list of functions you obtain in this way contains \(naos\) functions. Note that in this way we have implicitly also defined a list of all atoms, where all atoms that belong to a particular atom type are contiguous. This list is the so-called ‘internal’ atom ordering, which may not be identical to the order in which atoms were specified in input, under ATOMS.

For a given symmetry representation (Sections \(S\)) the array \(npart\) gives the indices of the basis functions in the overall list that are used to describe orbitals in this representation. In case of an unrestricted run the array \(npart\) applies for either spin: the same basis functions are used; the expansion coefficients for the molecular orbitals are different of course.

In the symmetry-representation sections \(Eigen\_BAS\) gives the expansion coefficients that describe the MOs. The expansion refer to the functions indicated by \(npart\), and the function characteristics are given by the arrays \(kx, ky, kz, kr, alf,\) and \(bnorm\), i.e. the expansion is in normalized functions.

The value of an MO is now obtained as a summation of values of primitive basis functions. For the evaluation of any such basis function you have to take into account that its characteristics are defined in the local coordinate system of its atom.

To obtain the charge density you sum all MOs, squared and multiplied by the respective occupation numbers (array \(froc\) in the appropriate irrep section).

Note that the auxiliary program \(densf\), which is provided with the ADF package, generates orbital and density values on a user-specified grid. See the UTILITIES document.
4.5 TAPE13

TAPE13 is the checkpoint file for restarts after a crash. It is a concise version of TAPE12, containing only the data the program uses for restarting the calculation. See the \texttt{RESTART} keyword.

Like TAPE21, TAPE13 is a binary, keyword driven KF file. You can manipulate it with the KF utilities, to get a print-out of its ‘table of contents’, or a complete ASCII dump of its full contents.

The calculation that produces TAPE13 determines which section are written on it. The following sections may occur (and if they occur, the listed variables are stored in them). The actual values of the variables should be identical to the corresponding variables on TAPE21. Also they should have the same names and be located in the same sections. In some cases, TAPE13 contains the complete corresponding section of TAPE21.

\textbf{Contents of TAPE13}

\textit{Section Fit}

\texttt{coef\_SCF}

SCF fit coefficients. Total number of them is nprimf, the number of primitive fit functions (counting all \textit{Cartesian} spherical polynomials: 3 for a \textit{p}-set, 6 for a \textit{d}-set, and so on). If the calculation is spin-unrestricted, each spin has its own set of fit coefficients: first all coefficients of spin-A, then those of spin-B

\texttt{coef\_FreqCenter}

Only in a Frequencies calculation: the fit coefficients that correspond to the equilibrium geometry. The variable \texttt{coef\_SCF} corresponds always to the current geometry, or the previous one if the geometry has just been changed and the new SCF has yet to start.

\textit{Section Freq}

This section is identical to the same section on TAPE21.

\textit{Section Geometry}

This section is identical to the same section on TAPE21

\textit{Section GeoOpt}

This section is identical to the same section on TAPE21
Section IRC
This section is identical to the same section on TAPE21

Section IRC_Foward
This section is identical to the same section on TAPE21

Section IRC_Backward
This section is identical to the same section on TAPE21

Section LT
This section is identical to the same section on TAPE21

Section TS
This section is identical to the same section on TAPE21
5 RESULTS

5.1 PROPERTIES

Electronic Configuration, Orbital Energies

The direct results from the SCF are the orbital energies and occupation numbers. This defines the electronic configuration: the occupation numbers and HOMO and LUMO energies for instance show whether or not the aufbau principle is satisfied in the final situation.

The energies of the Core Orbitals can be used to interpret for instance XPS data: from Koopman's theorem these core orbital energies are an approximation to the core ionization energies. This neglects the effect of relaxation upon the ionization so that absolute energy values may not be very good; relative values, however, should be fair and can therefore be used to study (relative) chemical shifts.

Populations and Atomic Charges

Mulliken populations

Mulliken-type populations are computed and printed at various levels of refinement (ranging from per-basis function to per-fragment type, data for the whole molecule as well as for individual MOs), and in two different representations, one based on the elementary basis functions (BAS), the other on SFOs (Symmetrized Fragment Orbitals). This is potentially a very large amount of data. Precisely what is printed by default, and how this can be modified so as to suppress output or, alternatively, to get more information, is regulated by the print keys (PRINT, EPRINT).

Atom and fragment charges

Mulliken populations can be summarized to yield atomic charges. Alternative methods exist to deduce atom charges from the self-consistent results of a molecular calculation. Two of those alternatives are provided by ADF: Hirshfeld analysis and Voronoi analysis.

The Hirshfeld analysis produces a charge value per fragment, computed as the integral of the SCF charge density over space, in each point weighted by the relative fraction of the (initial) density of that fragment in the total initial (sum-of-fragments) density:

\[ Q_{\text{frag}(i)} = \text{Error!} \]  (5.1.1)
The Voronoi charge analysis consists of assigning the charge density in a point in space to the nearest atom. The Voronoi cell of an atom is the region in space closer to that atom than to any other. This partitioning of space, using mid-way separation planes, is inappropriate to produce useful absolute numbers when neighboring atoms have very different sizes, for instance, Hydrogen and a heavy metal. However, changes in the density analyzed in this way do give a reasonable general insight in the effect of bonding on the location of charge densities, in particular because the Voronoi data per atom are split up in contributions within the atomic sphere and the rest of its Voronoi cell.

Hirshfeld and Voronoi charge analyses are printed at the end of the SCF (of the last geometry, in case of an Optimization).

The Hirshfeld analysis in ADF produces charges per fragment, so that atomic charges are obtained only if single-atom fragments are used. This limitation does not apply to Voronoi charges (data per atom). Mulliken charges are given both per atom and per fragment.

In the printout of charges per fragment (as for the Hirshfeld analysis), you have to be aware of the ordering of fragments. A complete list of fragments is printed in the early GEOMETRY section of standard output, where you also find which atom(s) correspond(s) to which fragment. Note that even when you use single-atom fragments only, the order of fragments is usually quite different from the order of atoms in your input file. Typically (but not necessarily exactly in each case), when you use single-atom fragments: consider the first non-dummy atom in your ATOMS block. This defines the first atom type. Then browse the ATOMS list until you find an atom of a different type. This defines the second atom type, and so on. The single-atom fragment list will often be such that you first get all atoms of the first atom type, then all atoms of the second type, and so on. Check the printed list-of-fragments always, to avoid mistakes in assigning Hirshfeld charges to atoms (fragments).

Energy

The program prints the bonding energy (not in a Create or Frequencies run) and its decomposition in terms that are useful for chemical interpretation. The total energy is not computed. The bonding energy is defined relative to the fragments. When basic atoms are employed as fragments one should realize that these do not represent the atomic ground state since they are computed as spin-restricted and spherically symmetric objects, with possibly fractional occupation numbers. The correct multiplet state is not computed. To obtain the bonding energy with respect to isolated atoms you should therefore add atomic correction terms to account for spin polarization and the multiplet state. See also the SLATERDETERMINANTS key and the discussion in the Theory document on multiplet states.

The spin polarization energy can be computed by running the single atom unrestricted, using as fragment the corresponding (restricted) basic atom. The true multiplet state is not necessarily obtained in this way.
For the comparison of computed bonding energies with experimental data one should furthermore be aware of any aspects that are not represented in the computational formalism, such as zero-point motions and environment (solvent) effects.

In a Geometry Optimization or Transition State search, the program may print a bonding energy evaluation at each geometry (depending on print switches). A test-energy value is written in the log file. This is not the bonding energy, although the difference is usually small. The test-energy printed in the log file is the energy expression from which the energy gradients are computed. The true bonding energy contains in addition a few (small) correction terms that are mostly related to the fit incompleteness. These correction terms are usually very small.

If Electric Fields are used in the computation (homogeneous and/or point charges), the printed Bonding Energy is the energy of the molecule in the field minus the energy of the fragments in the same field. The energy terms due to the field are also printed separately so that one can subtract them from the total bonding energy to obtain the energy-change without field-terms.

**Thermodynamics**

At the end of a completed Frequencies calculation, a survey is given of thermodynamic properties: Heat Capacity, Internal Energy, Entropy. The computed results assume an ideal gas, and electronic contributions are ignored. The latter is a serious omission if the electronic configuration is (almost) degenerate, but the effect is small whenever the energy difference with the next state is large compared to the vibrational frequencies.

```
| THERMO {P=pressure} {T=temp1 {temp2}} {nT=nT}
```

- **pressure**
  The Pressure in atmospheres. Default value: 1.0. A zero or negative pressure is adjusted by the program to a (very) small number 1e-4.

- **temp1, temp2**
  The endpoints of the Temperature range (in K), for which the results are computed. By default only room temperature is considered (298.15 K).
  If the option T= is used and only one value supplied (temp1), then temp2 is assumed to be equal to temp1.
  A zero or negative temparture is adjusted by the program to a (very) small number 1e-4.

- **nT**
  The number of steps by which the temperature interval is scanned. By default it is computed by the program from the temperature range (temp1, temp2), such that the step size is as close as possible to 10 K. Note that the number of temperatures for which the calculations are done is one more than the number of temperature steps.

The thermal analysis is based on the temperature dependent partition function. The energy of a (non-linear) molecule is
\[
\frac{E}{NkT} = \frac{3}{2} + \frac{3}{2} + \sum_{j=1}^{3N-6} \left( \frac{h\nu_j}{2kT} + \frac{h\nu_j}{kT(e^{h
u_j/kT} - 1)} \right) - \frac{D}{kT} 
\] (5.1.2)

The summation is over all harmonic frequencies \( \nu_j \), \( h \) is Planck’s constant and \( D \) is the dissociation energy.

\[
D = D_0 + \sum_j \frac{h\nu_j}{2} 
\] (5.1.3)
5.2 PLOTS: DENSITY, POTENTIAL, ORBITALS

To compute the electrostatic potential, charge density or molecular orbital values in a regular 2-D or 3-D grid, a separate program densf can be used. It requires the TAPE21 result file from the calculation and produces a TAPE41 files with the required data.

Other programs may process the TAPE41 data.

cntrs processes computes contours.

Adfplt displays orbitals, densities, potentials on your screen (2D, 3D) and can be used to print the pictures [96].

densf, cntrs and adfplt are auxiliary programs in the ADF package. See the UTILITIES document.
6 APPENDICES

6.1 DATABASE

The database contains standard basis sets (and fit sets, frozen core orbitals) for all chemical elements of the periodic table at different levels of accuracy. The database is partitioned in subdirectories. Some of these are special: the subdirectory Dirac contains input files for the program dirac (computation of relativistic potentials and charge densities). Most subdirectories contain files for the CREATE runs: subdirectories I through V.

The names of the files in the database consist of two parts: the standard symbol for the chemical element and the level of frozen core approximation. Mn.2p for instance is a data file for Manganese with a frozen core up to and including the 2p shell.

Polarization functions are not provided for all elements because our experience with them is limited. If you contemplate to compile more extended basis sets, by including one or more polarization functions, a good rule of thumb to choose the functional characteristics, is the following. Take the next higher \( l \)-value that does not yet occur in the function set (however, do not go beyond \( f \)-functions: the program cannot (yet) handle \( g \)-type basis functions), select the minimum value for the main quantum number \( n \) that is compatible with the \( l \)-value (i.e.: 2p, 3d, 4f), and determine the exponential decay factor such that the function attains its maximum value at somewhere between 1/3 and 1/2 times the bond length; the functional maximum for a Slater-type exponential function is at \( R=(n-1)/\alpha \).

A few all-electron basis are included in the database. In addition, some all-electron sets are provided in the subdirectory AE of the database. However, the files in AE do NOT contain fit functions so they cannot be used directly in Create runs. Fit functions for the all-electron basis sets must include more, in particular more contracted functions that the standard fit sets that are provided in the normal database files. If you would combine the all-electron basis set with an inadequate fit set the results are unreliable and absolutely inadequate, in the same fashion as when you would have used a highly inadequate basis set.

Data File for Create

The data file supplied to ADF in Create mode contains the following sections:

Title
Basis Functions
Core Expansion Functions
Core Description
Fit Functions
Start-up Fit Coefficients
Each of these items is discussed below. The data file does not define the applied density functional, the electronic configuration, precision parameters (numerical integration, SCF convergence criterion...), etcetera. These items can be set in the normal input file if the default is not satisfactory.

**Title**

is the first record of the file. It may contain any text. Only the first 60 characters are actually used. This title is (by default) printed in the output; it is also used to stamp an identification on the result file (TAPE21). The file stamp will be printed whenever you use it as a fragment file in another calculation.

**Basis functions**

A list of Slater type basis function characteristics. This part has the following format (example):

```
<table>
<thead>
<tr>
<th>BASIS</th>
</tr>
</thead>
<tbody>
<tr>
<td>1s</td>
</tr>
<tr>
<td>2s</td>
</tr>
<tr>
<td>...</td>
</tr>
<tr>
<td>(etc.)</td>
</tr>
<tr>
<td>...</td>
</tr>
<tr>
<td>END</td>
</tr>
</tbody>
</table>
```

The words `BASIS` and `END` signal the beginning and the end of this section in the data file. The records in-between list the basis functions; each record contains the main quantum number, the angular quantum number, and the exponential decay factor for a set of Slater type basis functions. A function description `3d 2.5` for instance represents the functions $r^2 e^{-2.5r} Y_{2m}(\Omega)$, $m = -2, ..., 2$.

The order of specification of the basis functions is not free. First must come the Core Functions used for core-orthogonalization, see Chapter 1.2. The CFs must be in order: s-functions first, then p-functions, then d-functions, and finally f-functions (as far as applicable). In the valence basis set there must be exactly one core-orthogonalization function for each frozen core shell (1s, 2s, 2p, ...).

Here as well as in all other function definitions below, the unit of length, implicit in the exponential decay factor, is bohr (atomic units), irrespective of the unit of length used in input for geometric items such as atomic positions (see UNITS).

**Core expansion functions**

This part has the form

```
<table>
<thead>
<tr>
<th>CORE n_s, n_p, n_d, n_f</th>
</tr>
</thead>
<tbody>
<tr>
<td>1s 7.68</td>
</tr>
<tr>
<td>...</td>
</tr>
</tbody>
</table>
```
It looks very much like the *basis functions*: a list of Slater type function descriptions, closed by `END`. The header record however (CORE...) contains in addition four integers $n_s$, $n_p$, $n_d$, $n_f$. They are the numbers respectively of $s$-, $p$-, $d$-, and $f$- frozen core shells in the atom. If you create for instance a Ruthenium atom with a frozen core up to the 4p shell, these numbers would be 4, 3, 0: four frozen $s$-shells (1s, 2s, 3s, 4s), three frozen $p$-shells (2p, 3p, 4p), one frozen $d$-shell (3d), and no frozen $f$-shells.

The core expansion sets defined in this section are used to describe the frozen core orbitals; they are not included in the valence basis set. In the list of core expansion sets all $s$-type functions must come first, then the $p$-type functions, then the $d$-functions, and then the $f$-functions (as far as applicable).

**Core description**

Describes the frozen core shells as linear combinations of the core expansion functions. This section has the form

```
| COREDESCRIPTION
| coefficients for the first frozen s-shell
| for the second s-shell
| ...
| for the $n_s$-th shell
| coefficients for the first frozen p-shell
| for the second p-shell
| ...
| for the d-shells
| for the f-shells
| pseudopotential parameters
```

END

For each of the angular momentum quantum numbers $l=s, p, d, f$ all $n_l$ frozen shells are described by giving expansion coefficients. There are as many coefficients as there are function sets with the pertaining $l$-value in the list of expansion functions. There are no separate coefficients for all $m$-values: all $m$-values are equivalent in a spherically symmetric model atom. See the Ca example below.

At the end of the (core) description section there is a record with pseudopotential parameters. The pseudopotential option, as an alternative to the frozen core approximation, is currently not supported, all values in this record must be zero, one for each frozen core shell. Equivalently you can put one zero, followed by a slash (/).
**Fit functions**

is again a list of Slater type functions. These are used for an expansion of the density. The Coulomb potential due to the electronic charge distribution is computed from this expansion, see Chapter 1.2.

The format of this section is similar to the *basis functions*:

```
FIT
1s  10.8
...
...
(etc.)
...
END
```

The program cannot handle fit functions with $l$-value higher than 4, i.e. not higher than $g$-type functions. Bear this in mind if you construct alternative fit sets.

In view of the next item, one is well advised to put the $s$-functions first.

**Start-up fit coefficients**

The initial (start-up for the SCF procedure) expansion of the atomic charge density in terms of the fit functions. Since the atom is spherically symmetric, only $s$-type functions should have non-zero coefficients. This is why the $s$-type fit functions should be listed first: the list of coefficients can then, after the $s$-set, be closed by a slash, rather than putting a long series of zeros.

The higher $l$-values (p, d...) in the fit set play no role in the creation of the basic atom, because it is spherically symmetric. They should not be omitted however as they will be needed when the atom is used as a fragment in a molecule: the charge density around the atom is then not spherically symmetric anymore.

The form of this section is simple:

```
FITCOEFFICIENTS
coefficients
END
```

**Example: Calcium**

An example may serve to illustrate the format of a Create data file (empty records inside and between the various sections are meaningless and ignored):

```
Calcium (II, 2p frozen)
BASIS
1S  15.8
2S  6.9
2P  8.1
```
3S  2.6
3S  3.9
3P  2.1
3P  3.4
4S  0.8
4S  1.35
4P  1.06
END

CORE  2  1  0  0
1S  24.4000
1S  18.2500
2S   7.4000
2S   4.8500
3S   4.0000
3S   2.5500
4S   0.7000
4S   1.0500
4S   1.6500
2P  10.8500
2P   6.4500
3P   1.8500
3P   2.7000
3P   4.0000
END

DESCRIPTION
0.2076143E+00 0.7975138E+00 -0.7426673E-04 0.1302616E-03 -0.6095738E-04
0.1508446E-04 0.1549420E-06 -0.2503155E-07 -0.1843317E-05
0.8487466E-01 -0.4505954E+00 0.1009184E+01 0.9627952E-01 -0.3093986E-01
0.1678301E-01 -0.2381843E-02 0.6270439E-02 -0.8899688E-02
0.3454503E+00 0.6922138E+00 -0.1610756E-02 0.5640782E-02 -0.5674517E-02

0/
END

FIT
7S   1.4000
7S   1.9300
7S   2.6600
6S   3.2100
6S   4.5400
6S   6.4100
5S   7.7500
5S  11.2800
4S  13.6400
4S  20.6700
3S  24.8500
2S  29.2500
1S  32.0000
6P   2.5500
6P   4.3900
5P   6.5200
4P   9.8700
3P  15.2700
2P  24.1000
6D   3.3800
6D   5.7100
5D   8.3300
4D  12.3500
3D  18.6800
5F   4.5300
4F  10.7800
5G   5.3600
END

FITCOEFFICIENTS
0.8835751E-05  0.5058920E-03  0.1741981E-01 -0.4393154E-01  0.3489301E+01
0.1975943E+03  0.7094304E+03 -0.5516550E+04  0.3390105E+03  0.3226271E+05
0.1004605E+04 -0.5203295E+03  0.5889693E+02
/
END
### 6.2 ELEMENTS OF THE PERIODIC TABLE

A few characteristics are predefined in ADF for all elements of the periodic table, as shown below. The electronic configuration defines the default occupation numbers in Create mode.

<table>
<thead>
<tr>
<th>Element</th>
<th>mass number of default isotope used for mass</th>
<th>electronic configuration</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cu</td>
<td>63</td>
<td>$3d^{10}4s^1$</td>
</tr>
<tr>
<td>Zn</td>
<td>64</td>
<td>$3d^{10}4s^2$</td>
</tr>
<tr>
<td>Ga</td>
<td>69</td>
<td>$3d^{10}4s^24p^1$</td>
</tr>
<tr>
<td>Ge</td>
<td>74</td>
<td>$3d^{10}4s^24p^2$</td>
</tr>
<tr>
<td>As</td>
<td>75</td>
<td>$3d^{10}4s^24p^3$</td>
</tr>
<tr>
<td>Se</td>
<td>80</td>
<td>$3d^{10}4s^24p^4$</td>
</tr>
<tr>
<td>Br</td>
<td>79</td>
<td>$3d^{10}4s^24p^5$</td>
</tr>
<tr>
<td>Kr</td>
<td>84</td>
<td>$3d^{10}4s^24p^6$</td>
</tr>
<tr>
<td>Rb</td>
<td>85</td>
<td>$5s^1$</td>
</tr>
<tr>
<td>Sr</td>
<td>88</td>
<td>$5s^2$</td>
</tr>
<tr>
<td>Y</td>
<td>89</td>
<td>$4d^25s^2$</td>
</tr>
<tr>
<td>Zr</td>
<td>90</td>
<td>$4d^25s^2$</td>
</tr>
<tr>
<td>Nb</td>
<td>93</td>
<td>$4d^55s^1$</td>
</tr>
<tr>
<td>Mo</td>
<td>98</td>
<td>$4d^55s^1$</td>
</tr>
<tr>
<td>Tc</td>
<td>(98)</td>
<td>$4d^55s^2$</td>
</tr>
<tr>
<td>Ru</td>
<td>102</td>
<td>$4d^{10}5s^1$</td>
</tr>
<tr>
<td>Rh</td>
<td>103</td>
<td>$4d^{10}5s^1$</td>
</tr>
<tr>
<td>Pd</td>
<td>106</td>
<td>$4d^{10}$</td>
</tr>
<tr>
<td>Ag</td>
<td>107</td>
<td>$4d^{10}5s^1$</td>
</tr>
<tr>
<td>Cd</td>
<td>114</td>
<td>$4d^{10}5s^2$</td>
</tr>
<tr>
<td>In</td>
<td>115</td>
<td>$4d^{10}5s^25p^1$</td>
</tr>
<tr>
<td>Sn</td>
<td>120</td>
<td>$4d^{10}5s^25p^2$</td>
</tr>
<tr>
<td>Sb</td>
<td>121</td>
<td>$4d^{10}5s^25p^3$</td>
</tr>
<tr>
<td>Te</td>
<td>130</td>
<td>$4d^{10}5s^25p^4$</td>
</tr>
<tr>
<td>I</td>
<td>127</td>
<td>$4d^{10}5s^25p^5$</td>
</tr>
<tr>
<td>Xe</td>
<td>132</td>
<td>$4d^{10}5s^25p^6$</td>
</tr>
<tr>
<td>Cs</td>
<td>133</td>
<td>$6s^1$</td>
</tr>
<tr>
<td>Ba</td>
<td>138</td>
<td>$6s^2$</td>
</tr>
<tr>
<td>La</td>
<td>139</td>
<td>$5d^16s^2$</td>
</tr>
<tr>
<td>Ce</td>
<td>140</td>
<td>$4f^35d^16s^2$</td>
</tr>
<tr>
<td>Pr</td>
<td>141</td>
<td>$4f^36s^2$</td>
</tr>
<tr>
<td>Element</td>
<td>Mass Number</td>
<td>Configuration</td>
</tr>
<tr>
<td>---------</td>
<td>-------------</td>
<td>---------------</td>
</tr>
<tr>
<td>Nd</td>
<td>142</td>
<td>4f^6s^2</td>
</tr>
<tr>
<td>Pm</td>
<td>145</td>
<td>4f^5s^2</td>
</tr>
<tr>
<td>Sm</td>
<td>152</td>
<td>4f^6s^2</td>
</tr>
<tr>
<td>Eu</td>
<td>153</td>
<td>4f^7s^2</td>
</tr>
<tr>
<td>Gd</td>
<td>158</td>
<td>4f^7d^16s^2</td>
</tr>
<tr>
<td>Tb</td>
<td>159</td>
<td>4f^8s^2</td>
</tr>
<tr>
<td>Dy</td>
<td>164</td>
<td>4f^9s^2</td>
</tr>
<tr>
<td>Ho</td>
<td>165</td>
<td>4f^{11} 6s^2</td>
</tr>
<tr>
<td>Er</td>
<td>166</td>
<td>4f^{12}6s^2</td>
</tr>
<tr>
<td>Tm</td>
<td>169</td>
<td>4f^{13}6s^2</td>
</tr>
<tr>
<td>Yb</td>
<td>174</td>
<td>4f^{14}6s^2</td>
</tr>
<tr>
<td>Lu</td>
<td>175</td>
<td>4f^{14}5d^16s^2</td>
</tr>
<tr>
<td>Hf</td>
<td>180</td>
<td>4f^{14}5d^26s^2</td>
</tr>
<tr>
<td>Ta</td>
<td>181</td>
<td>4f^{14}5d^36s^2</td>
</tr>
<tr>
<td>W</td>
<td>184</td>
<td>4f^{14}5d^46s^2</td>
</tr>
<tr>
<td>Re</td>
<td>187</td>
<td>4f^{14}5d^66s^2</td>
</tr>
<tr>
<td>Os</td>
<td>192</td>
<td>4f^{14}5d^66s^2</td>
</tr>
<tr>
<td>Ir</td>
<td>193</td>
<td>4f^{14}5d^76s^2</td>
</tr>
<tr>
<td>Pt</td>
<td>195</td>
<td>4f^{14}5d^86s^2</td>
</tr>
<tr>
<td>Au</td>
<td>197</td>
<td>4f^{14}5d^106s^1</td>
</tr>
<tr>
<td>Hg</td>
<td>202</td>
<td>4f^{14}5d^106s^1</td>
</tr>
<tr>
<td>Tl</td>
<td>205</td>
<td>4f^{14}5d^106s^26p^1</td>
</tr>
<tr>
<td>Pb</td>
<td>208</td>
<td>4f^{14}5d^106s^26p^2</td>
</tr>
<tr>
<td>Bi</td>
<td>209</td>
<td>4f^{14}5d^106s^26p^3</td>
</tr>
<tr>
<td>Po</td>
<td>(209)</td>
<td>4f^{14}5d^106s^26p^4</td>
</tr>
<tr>
<td>At</td>
<td>(210)</td>
<td>4f^{14}5d^106s^26p^5</td>
</tr>
<tr>
<td>Rn</td>
<td>(222)</td>
<td>4f^{14}5d^106s^26p^6</td>
</tr>
<tr>
<td>Fr</td>
<td>(223)</td>
<td>7s^1</td>
</tr>
<tr>
<td>Ra</td>
<td>(226)</td>
<td>7s^2</td>
</tr>
<tr>
<td>Ac</td>
<td>(227)</td>
<td>6d^77s^2</td>
</tr>
<tr>
<td>Th</td>
<td>232</td>
<td>6d^77s^2</td>
</tr>
<tr>
<td>Pa</td>
<td>231</td>
<td>5f^26d^77s^2</td>
</tr>
<tr>
<td>U</td>
<td>238</td>
<td>5f^36d^77s^2</td>
</tr>
<tr>
<td>Np</td>
<td>(237)</td>
<td>5f^46d^77s^2</td>
</tr>
<tr>
<td>Pu</td>
<td>(244)</td>
<td>5f^57s^2</td>
</tr>
<tr>
<td>Am</td>
<td>(243)</td>
<td>5f^77s^2</td>
</tr>
<tr>
<td>Cm</td>
<td>(247)</td>
<td>5f^6d^77s^2</td>
</tr>
<tr>
<td>Bk</td>
<td>(247)</td>
<td>5f^77s^2</td>
</tr>
<tr>
<td>Cf</td>
<td>(251)</td>
<td>5f^{10}7s^2</td>
</tr>
<tr>
<td>Es</td>
<td>(252)</td>
<td>5f^{11}7s^2</td>
</tr>
<tr>
<td>Fm</td>
<td>(257)</td>
<td>5f^{12}7s^2</td>
</tr>
<tr>
<td>Md</td>
<td>(258)</td>
<td>5f^{13}7s^2</td>
</tr>
<tr>
<td>No</td>
<td>(259)</td>
<td>5f^{14}7s^2</td>
</tr>
<tr>
<td>Lr</td>
<td>(260)</td>
<td>5f^{14}6d^17s^2</td>
</tr>
</tbody>
</table>

Default (most abundant) isotope, used to set atomic mass (nr. of brackets gives mass directly). Default electronic configurations used in Create mode.
6.3 SYMMETRY

Schönfliess symbols and symmetry labels

A survey of all point groups that are recognized by ADF is given below. The table contains the Schönfliess symbols together with the names of the subspecies of the irreducible representations as they are used internally by ADF. The subspecies names depend on whether single-group or double-group symmetry is used. Double-group symmetry is used only in relativistic spin-orbit calculations.

Note that for some input of TDDFT (Response) calculations, other conventions apply for the subspecies. This is explicitly mentioned in the discussion of that application.
<table>
<thead>
<tr>
<th>Point Group</th>
<th>Schönfliess Symbol</th>
<th>Irreducible representations in single-group symmetry</th>
<th>Irreducible representations in double-group symmetry</th>
</tr>
</thead>
<tbody>
<tr>
<td>C1</td>
<td>NOSYM</td>
<td>A</td>
<td>A1/2</td>
</tr>
<tr>
<td>R3</td>
<td>ATOM</td>
<td>s p d f</td>
<td>s1/2 p1/2 p3/2 d3/2 d5/2 f5/2</td>
</tr>
<tr>
<td>T_d</td>
<td>T(D)</td>
<td>A1 A2 E T1 T2</td>
<td>E1/2 U3/2 E5/2</td>
</tr>
<tr>
<td>O_h</td>
<td>O(H)</td>
<td>A1.g A2.g E.g T1.g T2.g A1.u A2.u E.u T1.u T2.u</td>
<td>E1/2.g U3/2.g E5/2.g E1/2.u U3/2.u E5/2.u</td>
</tr>
<tr>
<td>Cxx</td>
<td>C(LIN)</td>
<td>Sigma Pi Delta Phi</td>
<td>J1/2 J3/2 J5/2 J7/2</td>
</tr>
<tr>
<td>D_zdh</td>
<td>D(LIN)</td>
<td>Sigma.g Sigma.u Pi.g Pi.u Delta.g Delta.u Phi.g Phi.u</td>
<td>J1/2.g J1/2.u J3/2.g J3/2.u J5/2.g J5/2.u J7/2.u</td>
</tr>
<tr>
<td>C_i</td>
<td>C(I)</td>
<td>A.g A.u</td>
<td>A1/2.g A1/2.u</td>
</tr>
<tr>
<td>C_s</td>
<td>C(S)</td>
<td>AA AAA</td>
<td>A1/2 A1/2*</td>
</tr>
<tr>
<td>C_n</td>
<td>C(N), n must be 2</td>
<td>A B E1 E2 ... odd n: without B</td>
<td>A1/2 A1/2*</td>
</tr>
<tr>
<td>C_nh</td>
<td>C(NH), n must be 2</td>
<td>even n: A.g B.g A.u B.u E1.g E1.u E2.g E2.u ...</td>
<td>A1/2.g A1/2.g* A1/2.u A1/2.u*</td>
</tr>
<tr>
<td></td>
<td></td>
<td>odd n: AA AAA EE1 EE2 ...</td>
<td></td>
</tr>
<tr>
<td>C_nv</td>
<td>C(NV), n&lt;8</td>
<td>A1 A2 B1 B2 E1 E2 E3 ... odd n: without B1 and B2</td>
<td>E1/2 E3/2 E5/2 ... for even n also: An2 An2*</td>
</tr>
<tr>
<td>D_n</td>
<td>D(N), n&lt;8</td>
<td>n=2: A B1 B2 B3 other: A1 A2 B1 B2 E1 E2 E3</td>
<td>E1/2 E3/2 ... for odd n also: An2 An2*</td>
</tr>
<tr>
<td></td>
<td></td>
<td>... odd n: without B1 B2</td>
<td></td>
</tr>
<tr>
<td>D_nh</td>
<td>D(NH), n&lt;8</td>
<td>n=2: A1.g B1.g B2.g B3.g A1.u B1.u B2.u B3.u</td>
<td>even n: E1/2.g E1/2.u E3/2.g E3/2.u ...</td>
</tr>
<tr>
<td></td>
<td></td>
<td>even n (≠2): A1.g A2.g B1.g B2.g E1.g E2.g E3.g ... A1.u A2.u B1.u ...</td>
<td>odd n: E1/2 E3/2 E5/2 ...</td>
</tr>
<tr>
<td></td>
<td></td>
<td>odd n: AA1 AAA2 EE1 EE2 ...</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>AAA1 AAA2 EE1 EE2 ...</td>
<td></td>
</tr>
<tr>
<td>D_nd</td>
<td>D(ND), n&lt;8</td>
<td>n=2: A1 A2 B1 B2 E1 other: A1.g A2.g E1.g E2.g ... E(n-1)/2.g A1.u A2.u E1.u E2.u ...</td>
<td>even n: E1/2 E3/2 ...</td>
</tr>
<tr>
<td></td>
<td></td>
<td>odd n: E1/2.g E1/2.u E3/2.g E3/2.u ... An2.g An2.u</td>
<td>odd n: E1/2.g E1/2.u E3/2.g</td>
</tr>
<tr>
<td></td>
<td></td>
<td>... An2.g* An2.u*</td>
<td></td>
</tr>
</tbody>
</table>

Schönfliess symbols and the labels of the irreducible representations.

Most labels are easily associated with the notation usually encountered in literature. Exceptions are AA, AAA, EE1, EEE1, EE2, EEE2, etc. They stand for A’, A”, E1’, E1”, and so on. The AA, etc. notation is used in ADF to avoid using quotes in input files in case the subspecies names must be referred to.
The symmetry labeling of orbitals may depend on the choice of coordinate system. For instance, B1 and B2 representations in C$_{2v}$ are interchanged when you rotate the system by 90 degrees around the z-axis so that x-axis becomes y-axis and vice-versa (apart from sign).

Labels of the symmetry subspecies are easily derived from those for the irreps. For one-dimensional representations they are identical, for more-dimensional representations a suffix is added, separated by a colon: For the two- and three-dimensional E and T representations the subspecies labels are obtained by adding simply a counting index (1, 2, 3) to the name, with a colon in between; for instance, the EE1 irrep in the Dnh pointgroup has EE1:1 and EE1:2 subspecies. The same holds for the two-dimensional representations of C$_{\infty V}$ and D$_{\infty h}$.

For the R$^3$ (ATOM) point group symmetry the subspecies are p:x, p:y, p:z, d:z$^2$, d:x$^2$-y$^2$, etc. All subspecies labels are listed in the Symmetry section, very early in the ADF output. To get this, perform a quick run of the molecule using the STOPAFTER key (for instance: STOPAFTER CONFIG).

**Molecular orientation requirements**

ADF requires that the molecule has a specific orientation in space, as follows:

- The origin is a fixed point of the symmetry group.
- The z-axis is the main rotation axis, xy is the $\sigma_h$-plane (axial groups, C(s)).
- The x-axis is a C$_2$ axis (D symmetries).
- The xz-plane is a $\sigma_v$-plane (C$_{nv}$ symmetries).
- In T$_d$ and O$_h$ the z-axis is a fourfold axis (S$_4$ and C$_4$, respectively) and the (111)-direction is a threefold axis.

If the user-specified symmetry equals the true symmetry of the nuclear frame (including electric field and point charges) the program will adapt the input coordinates to the above requirements, if necessary. If no symmetry has been specified at all ADF assumes you have specified the symmetry of the nuclear frame, accounting for any fields. If a subgroup has been specified for the molecular symmetry the input coordinates will be used as specified by the user. If a Z-matrix input is given this implies for the Cartesian coordinates: first atom in the origin, second atom on the positive x-axis, third atom in the xy-plane with positive y value.
REFERENCES

7. Versluis, L., The determination of molecular structures by the HFS method, . 1989, University of Calgary


27. Snijders, J.G., E.J. Baerends, and P. Ros, *A perturbation theory approach to relativistic calculations. II. Molecules*. Molecular Physics, 1979, **38**: p. 1909


30. DeKock, R.L., *et al.*, *Electronic structure and bonding of Hg(CH₃)₂, Hg(CN)₂, Hg(CH₃)(CN), Hg(CCCH₃)₂, and Au(PMe)₃(CH₃)*. Journal of the American Chemical Society, 1984, **106**: p. 33387


41. Pye, C.C. and T. Ziegler, Theoretical Chemistry Accounts, 1999, **101**: p. 396-408


56. van Gisbergen, S.J.A., _et al._, _Journal of Chemical Physics_, accepted


58. Rosa, A., _et al._, submitted


60. Champagne, B., _et al._, _Assessment of conventional density functional schemes for computing the polarizabilities and hyperpolarizabilities of conjugated oligomers: An ab initio investigation of polyacetylene chains_. Journal of Chemical Physics, 1998, 109: p. 10489-10498


64. van Lenthe, E., P.E.S. Wormer, and A. van der Avoird, _Journal of Chemical Physics_, 1997, 107: p. 2488

van Lenthe, E., in preparation


Edmiston, C. and K. Rüdenberg, Rev. Mod. Phys., 1963, **35**: p. 457

Boys, S.F., Rev. Mod. Phys., 1960, **32**: p. 253

von Niessen, W., Journal of Chemical Physics, 1967, **47**: p. 253

Hirshfeld, F.L., Theoretica Chimica Acta, 1977, **44**: p. 129


Bickelhaupt, F.M., *et al.*, *The Carbon-Lithium Electron Pair Bond in (CH\(_3\)Li)\(_n\) (n = 1, 2, 4)*. Organometallics, 1996, **15**: p. 2923
96. Autschbach, J., On the calculation of relativistic effects and how to understand their trends in atoms and molecules, in Chemistry. 1999, University of Siegen: Siegen