

Utilities

ADF Program System
Release 2002.02

SCIENTIFIC COMPUTING & MODELLING NV
Vrije Universiteit; Theoretical Chemistry
De Boelelaan 1083; 1081 HV Amsterdam; The
Netherlands
E-mail: support@scm.com



28 June, 2002

Copyright © 1993–2002: SCM, Vrije Universiteit, Theoretical Chemistry, Amsterdam, The Netherlands
<http://www.scm.com>
All rights reserved.

1	INTRODUCTION	3
2	GENERAL UTILITIES	4
2.1	KF command line utilities	4
	pkf.....	4
	cpkf.....	5
	dmpkf.....	5
	udmpkf.....	6
3	PROPERTIES, ANALYSIS	8
3.1	Densf: Volume Maps	8
	Input.....	8
	Result: TAPE41.....	14
	Notes.....	16
	Contents of TAPE41.....	16
3.2	Cntrs: Contour Plots	18
	Input.....	18
	Result.....	20
3.3	ADFPLT: GUI for volume Maps	21
3.4	Dos: Density of States	23
	Introduction.....	23
	Mulliken population analysis.....	24
	Density of states analyses based on Mulliken population analysis.....	26
	Generalizations of OPDOS, GPDOS, PDOS.....	27
	Input.....	28
4	OTHER	33
4.1	Dirac: Relativistic Potentials	33
	Implied options.....	33
	Input.....	34
4.2	BOBIT	36
	Introduction.....	36
	Initialization.....	36
	Include Menu.....	36
	Other Menus.....	36
	Click on Marked Items.....	37
	Customizing BOB.....	37
	REFERENCES	38

1 INTRODUCTION

The ADF package contains two main programs – ADF and BAND for calculations on, respectively, molecules and periodic systems – and a suite of auxiliary utility and property programs, which are used to prepare special input data or to process results. This manual describes the usage of the utility and property programs. Consult also the *User's Guide*. You should be familiar with the *User's Guide* anyway before reading this *Utilities* document because some concepts from the *User's Guide* will be assumed known here. Among these are the use, format, and general types of keywords applied in the input files as well as in the input for some of the utilities. Most of the utility programs relate only to ADF, in the current version of the package.

The utilities are installed automatically when the ADF package is installed. See the *Installation Manual*.

Some of the files that will be mentioned are KF files. A KF file (Keyed-File) is a special type of Direct-Access Fortran file used in programs of the ADF suite. They have a keyword driven organization and can easily be processed with the KF-utilities that come with the ADF package.

Overview

The utilities discussed in this manual are:

- KF utilities: UNIX command-line utilities to process KF files;
- *dirac*: a program to generate relativistic potentials of the isolated atom, and of its frozen core, if present. These are used in an ADF calculation when relativistic options are applied.
- *densf*: a program to generate values of the charge density, potential, molecular orbitals in a user-specified regular grid (2-D or 3-D);
- *cntrs*: a program to generate contours for data computed by *densf*;
- *dos*: a program to create density-of-states (DOS) type info (Total DOS, Projected DOS, ...), to be plotted or printed.
- *nmr*: a program to compute NMR chemical shifts. It requires the TAPE21 result file of an ADF calculation.
- *disper*: a program to compute dispersion coefficients for the long-range interaction between two molecules. It requires the results from two ADF calculations on the interacting molecules.
- *bobit*: a program which provides a graphical user interface to browse through ADF standard ("print") output files.

2 GENERAL UTILITIES

2.1 KF COMMAND LINE UTILITIES

There are four utility programs for manipulating KF files from the command shell. Two of them convert KF files from binary to ASCII and vice versa. See the *pkf* and *dmpkf* utilities for a description of the ASCII format of a KF file. An ASCII version of a KF file can be useful to inspect its contents in detail. The conversion back and forth between binary and ASCII may be necessary when a binary KF file generated on a particular platform is to be used on another platform that is not binary compatible. In such case: convert to ASCII, transfer to the other platform and transfer back to binary. Although a bit tedious, it occasionally is the only way to avoid recomputing a TAPE21 result file only because you need it on another machine.

The KF software (KF= Keyed File) has been developed at the Vrije Universiteit in Amsterdam as a general-purpose package to store data on files and retrieve it again by keyword-driven procedures. For more information about the KF package (usage, implementation) consult the SCM web site (<http://www.scm.com>) where information about the ADF software is available.

pkf

```
| pkf file1 { file2 ... fileN }
```

pkf prints a summary of the contents of the KF files *file1... fileN*.

The output should be more or less self-documenting: all variables are listed by name, type (integer, real, character, logical) and size (number of array elements) and they are grouped together in named sections.

To put the results in an ASCII file for later inspection:

```
| pkf file > ascii_result
```

Each section on the file contains an index of its variables and their associated values. All data are organized in blocks. Each section may have any number of index blocks and any number of data blocks (this depends simply on the amount of data to be stored in such a block). In addition there is one special section, the SuperIndex, which is an index of all sections on the file.

The output of *pkf* consists of:

1. General information about the file (name of the file, internally used unit numbers during processing the file...)
2. A summary of the SuperIndex: an index of blocks on the file and the sections they are associated with.
3. A summary: total numbers of blocks associated with the different types of blocks.
4. For each section a list of its variables with for each variable:
 - a. Its name;
 - b. Its length: the amount of space reserved on the file for the variable;
 - c. Its size ('used'): the amount of data associated with the variable; for reals, integers and logicals: the number of such elements; for strings: the number of characters;
 - d. The (logical) index of the data block it is stored on;
 - e. Off-set: its position within the data block in which it is stored;
 - f. Its value (for an array: the value of the first element);

Remark: 'length' and 'size' are usually the same, but not necessarily.

cpkf

```
| cpkf    file1 file2 {key1 .. keyn}
```

cpkf copies the sections and/or variables `key1 .. keyn` from `file1` to `file2`. If a referenced section or variable already exists on `file2` it is overwritten, else it is created. Sections and variables which are already present on `file2` but which are not referenced in the command are not affected.

If no sections and/or variables are explicitly mentioned at all the copy is carried out for *all* sections and variables on `file1`.

A side effect of the copy is that any 'holes' that may be present on the first file are not copied to the second file so that they no longer take up any space. The data copied to the second file is rearranged for optimum storage efficiency. "Holes" may be due to sections and variables that have existed in the past but that have formally been deleted later; one of the KF functionalities is to delete variables or sections. Such activity does not actually rearrange the KF file, but simply deletes the corresponding entries in index tables.

dmpkf

A utility to extract information from a KF file and make it available in ASCII format.

```
| dmpkf  file {key1 .. keyn}
```

dmpkf prints the sections and/or variables from the file *file* indicated by *key1* .. *keyn* on standard output. If no sections and/or variables are explicitly mentioned the complete file is printed.

The format to be used for *key1* et cetera is:

Sec%Var

where *Var* is the variable, which must exist in section *Sec*. If no *Var* is mentioned, the complete section *Sec* is dumped.

By redirecting the result to another file you get an ASCII version of *file*:

```
| dmpkf file > ascii_result
```

The output contains for each printed variable:

- One line with the name of the section it belongs to;
- One line with the name of the variable itself;
- One line with three integers:
 - The amount of space reserved for the variable on the file (this aspect is relevant for the software that handles KF files and is mentioned here only for completeness);
 - The amount of data associated with the variable: for reals, integers, logicals: the number of such elements; for strings: the number of characters;
 - An integer code for the data type of the variable: 1=integer, 2=real, 3=character, 4=logical;
- The values of the variable (on as many lines as necessary): for scalar variables only one value, for arrays as many values as the array contains.

udmpkf

A utility to put information read from standard input into a KF file.

```
| udmpkf file
```

udmpkf reads an ASCII file in the format created by *dmpkf* from standard input and creates the KF file *file* from this data if it does not already exist, otherwise it adds the sections and/or variables in the input file to the existing KF file. If sections or variables on the input file already exist in the target file, that data on the target file is overwritten. Other data on the target file are not affected.

The combination of *dmpkf* and *udmpkf* makes it easy to modify KF files with a normal text editor:

```
dmpkf TAPE21 > t21_ASCII
```

If necessary, edit and modify *t21_ASCII*, and then

```
udmpkf < t21_ASCII TAPE21_new
```

Note carefully that *dmpkf* and *udmpkf* do NOT have two arguments, but only one each. The ASCII “argument” is standard input, respectively output, which you may of course redirect to a file.

3 PROPERTIES, ANALYSIS

3.1 DENSF: VOLUME MAPS

densf is an auxiliary program to generate values of molecular orbitals, charge densities and potentials in a user-specified grid, to be used typically for plotting or graphical display. The TAPE41 result file can be used directly by the ADFview program to visualize these properties.

densf requires an ASCII input file where the user specifies the grid and the items that he wishes to see calculated in the grid, plus the standard result file TAPE21 from an *adf* calculation. TAPE21 must be present as a local file in the directory where *densf* runs. *densf* writes a summary of the items that have been requested to standard output, together with some general information.

densf produces a (binary) KF file TAPE41. Any pre-existing TAPE41 will be deleted by *densf* before it creates the new file. TAPE41 is a KF file and all KF utilities can be used to inspect and process its data.

Furthermore TAPE41 can be processed by *cntrs* to generate contour plot data. TAPE41 can also be processed by *adfplt* to display orbitals or densities on your screen, or to get a picture printed. The ADFview program can be used to view the data available in TAPE41 in a variety of ways. *Cntrs* and *adfplt* are separate utility programs, see later sections in this documentation. For the ADFview program separate documentation is available.

Examples of using *densf* are contained in the set of sample runs; see the *Examples* document.

Input

The input for *densf* is keyword oriented. The order of keywords in input is immaterial. Reading input by *densf* ends when it encounters the record `ENDINPUT` or the end-of-file, whichever comes first.

The current version of *densf* does have reasonable defaults for all input. That means that in many cases you probably will not need to specify any input at all.

Follows a list of the admissible keywords with their meaning.

Grid

The Grid key is available either as simple key, or as block key.

The simple key options are as follows:

```
| GRID {save} {coarse|medium|fine}
```

If the word `save` is specified, the program will store all grid points on TAPE41 (in addition to the specification of the grid that is always stored). The default is NOT to store all grid points.

Either `coarse`, `medium` or `fine` may be specified. This instructs the program to generate the grid automatically within a box enclosing all atoms of the molecule. The distance between grid points is 0.5, 0.2 or 0.1 bohr for respectively a coarse, medium or fine grid. Evidently the size of the result file TAPE41 depends strongly on this specification. The default value (used when the user does not specify the grid) is to generate a coarse grid.

If GRID is used as a block key it must be followed by the word `END` in a later record. The records until the `END` are the data for the `Grid` keyword:

```
| GRID {save}  
  x0, y0, z0  
  n1, n2, n3  
  v1x, v1y, v1z, length1  
  v2x, v2y, v2z, length2  
  v3x, v3y, v3z, length3  
| END
```

If the word `save` is specified, the program will store all grid points on TAPE41 (in addition to the specification of the grid that is always stored). The default is NOT to store all grid points.

The records in the data block must contain (*in order!!*):

- 1 Three coordinates for the 'origin' (lower-left corner) of the grid;
- 2 Three integers: the numbers of points in three independent directions. If fewer integers are supplied the grid will accordingly be less-dimensional;
- 3 Three records each containing the coordinates for the direction of the independent vector (size irrelevant) and the total length of the grid in that direction. If a lower-dimensional grid is requested (see item #2), then fewer such direction-records are read and the redundant ones, if any, are ignored.

The unit of length in which the grid size is input is by default Angstrom. The default (Angstrom) can be overridden by using the input key UNITS, see below.

Notes:

- The second record ('three integers...') specifies the number of grid *points* in the different directions. The corresponding number of steps or intervals is one less!
- If the TAPE41 result file is to be used by the contour generating program *cntrs*, the grid used in the *densf* calculation must be *two*-dimensional.
- If the TAPE41 result file is to be used by ADFview, the grid used must be an three-dimensional orthogonal grid, with a single step size for all three dimensions.
- The unit of length used in the input file has no relation to how the data are stored on the result file and how the program processes the data internally. Internal processing and storage on file is in bohr (atomic units).

Units

As in the ADF main program, the unit of length can be set with the block type key

```
| UNITS
|   Length      unit_of_length
| END
```

In *densf* the only item that can be specified in the UNITS block is the length, so it seems a bit pointless to make UNITS a *block* type key rather than a simple key. However, to make its usage identical to the application in the *adf* main program the block form has been chosen to apply also here. The unit-of-length will apply to the grid specification in the input file. Default is angstrom.

Density

Generates the charge density in the grid. It is a simple keyword (not block-type).

```
| DENSITY  {fit}  {frag}  {ortho}  {scf}
```

Occurrence of the word *fit* specifies that all densities specified in this record will be computed from the fit functions (an approximation to the exact density), rather than from the occupied molecular orbitals.

frag, *ortho*, and *scf* causes each of the corresponding densities to be computed. *frag* stands for the sum-of-fragments (i.e. the initial) density, *scf* for the final result of the ADF calculation, and *ortho* for the orthogonalized fragments (orthogonalization to account for the Pauli repulsion, see the ADF User's Guide) .

If both the exact and the fit-densities are required the density keyword must be repeated, once with and once without the *fit* option specified.

The default (when the DENSITY key does not occur in the input file) is to calculate the final SCF density and the sum-of-fragments density.

Potential

Generates the coulomb and/or exchange-correlation potential in the grid.

```
| POTENTIAL   {coul / XC}  {frag}  {ortho}  {scf}
```

`frag`, `ortho`, and `scf` are as for the density: at least one must be specified. `coul` and `xc` specify that the Coulomb potential, respectively the exchange-correlation potential must be computed. Precisely one of these options must be specified in the record. If both potential types are required, another input record with the `POTENTIAL` key must be used.

In the present release the `xc` option is not yet operational.

The default (when the `POTENTIAL` key does not occur in the input) is to calculate the SCF Coulomb potential.

Orbitals

A block type key in which the required molecular orbitals are specified. The key can be repeated in input any number of times; all occurrences are read and applied.

```
| ORBITALS    type  
| (data)  
| END
```

The argument of the `ORBITALS` key (`type`) must be `SCF` (for the `SCF` orbitals) or `FRAG` (for the fragment orbitals) or `LOC` (for the localized molecular orbitals, see the ADF User's Guide).

The `FRAG` option is not operational in the present release.

In many data records in the `ORBITALS` block, as noted in the description of these data records, you may specify a HOMOLUMO range.

A HOMOLUMO range is the following:

```
{HOMO{{-}n}} {LUMO{{+}n}}
```

HOMO: the highest occupied orbital

HOMO-*n*, with *n* an integer: the highest (*n*+1) occupied orbitals

LUMO: the lowest virtual orbital

LUMO+*n*, with *n* an integer: the lowest (*n*+1) virtual orbitals.

The HOMO part, or the LUMO part, or both must be specified. The integer *n* with sign is always optional, and the sign is always optional (and has no meaning, it is intended to enhance readability).

Thus, as an example,

```
HOMO-1 LUMO+1
```

means a range of 4 orbitals: the two highest occupied ones, and the two lowest virtuals.

Each data record in the `ORBITALS` block must have either of the following formats:

1 the word `alpha` or `beta`.

This specifies that subsequent records refer to spin-alpha or spin-beta orbitals respectively. In a restricted calculation this has no meaning and `beta` must not be specified.

`alpha` and/or `beta` may occur any number of times in the `ORBITALS` block. All records until the first occurrence of `alpha` or `beta` are assumed to refer to spin-alpha orbitals.

2 `label n1, n2, n3, ...`

`label` is one of the subspecies of the point group symmetry used in the *adf* calculation and `n1` etc. are indices of the molecular orbitals (in that subspecies) that are to be computed. This format is meaningless and must not be used for the `LOC` orbitals type, because *localized* orbitals do not (necessarily) belong anymore to a particular symmetry representation.

3 `label HOMOLUMO`

`label` is one of the subspecies of the point group symmetry used in the calculation, the orbitals follow from the HOMOLUMO range.

4 `label occ`

or

`label virt`

`occ` specifies all orbitals (in that symmetry representation) up to and including the highest occupied one. `virt` specifies all orbitals above the highest occupied one. In this context *partially* occupied orbitals are considered occupied. Note carefully that if in a particular symmetry representation an empty orbital is computed below the highest occupied one in that same representation (excited state), that particular empty one is included in the list of `occ`.

Again, this format is meaningless and must therefore not be used for the `LOC` type of orbitals.

5 ALL OCC

or

ALL VIRT

OR

ALL HOMOLUMO

Specifies *for each symmetry representation*:

- all orbitals up to and including the highest occupied one (in that symmetry), or
- all orbitals above the highest occupied one, or
- all orbitals defined by the HOMOLUMO range.

This form is not to be used for the `LOC` type of orbitals. However, using this for `LOC` will not result in an error but will be interpreted as identical to the following format.

6 ALL

This format must be used only for the `LOC` type of orbitals and simply means: all computed localized orbitals (irrespective of occupation numbers).

7 n1, n2, ...

a simple list of integer indices. This format must be used only for the `LOC` type of orbitals since no reference is made to any symmetry representation. The indices refer of course to the list of localized orbitals as computed by *adf*, see the *User's Guide*.

The default value used when the ORBITALS key is not present is:

```
Orbitals SCF
  All HOMO-1 LUMO+1
End
```

Memory usage

The default value for the maximum amount of memory that can be used by the program to perform the required computations is chosen and defined at the installation of the whole package and is identical to the value for the main programs (*adf*, *band*). As for these programs, the actual value can be adjusted for any particular run with certain keywords, in exactly the same way as for *adf* and *band*.

Apart from the maximum total amount of memory you can also adjust some related technical parameters. These should only affect efficiency issues and in general you should not adjust them.

Follows a description of the pertaining keys. All input keys pertaining to dynamical allocation are simple keys and specify the corresponding quantities in Mbytes. For the total amount of memory:

	MAXMEMORYUSAGE	n
--	----------------	---

The technical parameters, defining how chunks of memory are allocated:

	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. Their normal values are; for

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.

Input control of memory usage is applicable only if *dynamical* memory allocation has been turned on in the installation (which is the default, but you may have adjusted that).

Result: TAPE41

Follows a description of the contents of TAPE41. We start with a brief discussion of the sections. At the end you can find an uncommented list of all variables and sections. Note that some data are only generated when certain keywords are provided.

Grid

This is a general section. It contains the grid data and some more general info.

The grid characteristics are stored as:

- 1 The 'origin' of the grid;
- 2 The numbers of points in three independent directions;
- 3 Three vectors, called 'x-vector', 'y-vector' and 'z-vector'. They are the *steps* in the three independent directions that define the grid.

If the `save` option was used in input (key `GRID`) also all grid coordinates are stored: for each point three coordinates (xyz), also if only a 2-dimensional or 1-dimensional grid has been generated (a 2D grid does not necessarily lie in the xy-plane).

Note that the grid values are now stored in a simpler manner than in previous values of densf, because the “x values”, “y values”, and “z values” now each have their own, separate sections.

The remaining (general) data in this section comprises:

- The number of subspecies ('symmetries') for which data such as Molecular Orbitals may be present;
- The names of the subspecies;
- A logical with the name 'unrestricted', which flags whether the data pertain to an unrestricted calculation;
- The total number. of grid points.

SumFrag

Contains grid data of the Sum-of-fragments (charge density, coulomb potential, ...)

Ortho

Contains similarly data for the orthogonalized-fragments.

SCF

Contains the (spin) density, potential, etc. of the final (SCF) solution.

SCF_label

'Label' must be one of the symmetry subspecies.

Each such section contains the total number of orbitals in that subspecies (as used in the ADF calculation), with their occupation numbers and energy eigenvalues.

In addition it contains the grid-values of the (user-specified subset of) MOs in that subspecies. The variable name corresponding to an orbital is simply its index in the energy-ordered list of all orbitals (in that subspecies): '1', '2', etc.

LocOrb

Values of the localized orbitals.

Geometry

Some general geometric information: the number of atoms (not counting any dummy atoms that may have been used in the *adf* calculation), their Cartesian coordinates (in bohr) and nuclear charges.

Note: the *order* of the atoms here is not necessarily identical to the input list of atoms: they are grouped by atom type.

Notes

- In an unrestricted calculation the section `SCF_label` is replaced by `SCF_label_A` and `SCF_label_B` for the spin-alpha and spin-beta data, respectively, and similarly for `LocOrb`: `LocOrb_A` and `LocOrb_B`.
- One or more subspecies may not have been used in the *adf* calculation. This happens when the basis set used in that calculation does not contain the necessary functions to span symmetry-adapted combinations of basisfunctions for that subspecies. In such a case the corresponding section on TAPE41 will not be created by *densf*.
- If you want to verify the contents of TAPE41, use the *pkf* utility to obtain a survey or *dmpkf* to get a complete ASCII printout.

Contents of TAPE41

Section names are printed leftmost on the page, variable names are printed with an indent. *Italic* text is comment. In case of array variables the number of elements is given between parentheses to the right side of the page. String lengths are indicated as (*length), an array of such strings as (n*length). Variables of type logical are marked by an uppercase L at the right side of the page.

Note that the name of a section of variable may consist of more than one word and that blanks in such names are significant. Furthermore, they are case-sensitive. Each line below contains the name of only one section or variable.

Grid

```
Start-point (3)
nr of points x
nr of points y
nr of points z
total nr of points
x-vector
y-vector
z-vector
nr of symmetries
labels (nr of symmetries*80)
unrestricted (L)
```

SumFrag

```
CoulPot (total nr of points)
XCPot_A (idem)
XCPot_B (idem)
(in a spin-restricted calculation: XCPot rather than XCPot_A, _B)
Density_A (idem)
Density_B (idem)
(in a spin-restricted calculation: Density)
```

Ortho

same variables as in SumFrag

SCF

Same variables as in SumFrag and Ortho

SCF_label_A *label is a symmetry subspecies. Spin-restricted:* SCF_label

nr of orbitals

Occupations

(nr of orbitals)

Eigenvalues

(nr of orbitals)

1

2

3

(as many as there are Molecular Orbitals in that symmetry representation for the indicated spin)

SCF_label_B *(if unrestricted)*

same variable as in SCF_label_A

LocOrb_A *if unrestricted, otherwise* LocOrb

nr of orbitals

1

2

(etc)

Geometry

nnuc *nr of nuclei, omitting dummy atoms used in the calculation.*

xyznuc

(nnuc times 3)

the atoms are not in the same order as in the ADF input file. Rather they are grouped by atomtype.

qtch *nuclear charges*

(nnuc)

x values

x values

(total nr. of points)

y values

y values

(total nr. of points)

z values

z values

(total nr. of points)

3.2 CNTRS: CONTOUR PLOTS

cntrs is an auxiliary program to generate plot data from TAPE41 produced by *densf*. In the future *cntrs* will be superseded by the ADF-GUI, which will offer both two-dimensional and three-dimensional visualization possibilities.

cntrs requires an ASCII input file where the user specifies which items should be plotted and what scan values are to be used, plus the standard result file TAPE41 from *densf*. TAPE41 must be present as a local file in the directory where *cntrs* executed. For usage by *cntrs* TAPE41 must have been generated in a *densf* run using a two-dimensional grid.

cntrs produces as result one or more ASCII files with plot data. An example of using *cntrs* is contained in the set of sample runs (NO₂), see the *Examples* document.

Input

The (ASCII) input for *cntrs* is keyword oriented. *The order of keywords in input is relevant for cntrs.*

Scan

```
| SCAN  
|   scanvalues  
| END
```

With the `SCAN` key you read in values for which contours are to be generated for the items that are specified subsequently in input. Up to a maximum number of 20 scan values can be supplied. `SCAN` may occur any number of times in input. Each occurrence resets the scan values for the subsequent items. The initial values, which apply until the first occurrence, if at all, of `SCAN` are the eleven values 0, $\pm 2e-2$, $\pm 5e-2$, $\pm 1e-1$, $\pm 2e-1$, $\pm 5e-1$.

Dash

```
| DASH length
```

contours corresponding to positive scan values are plotted as solid lines, the zero-contour is plotted by a dash-dot-dash line, and negative contours are dash-lines. The `DASH` key defines the length of a dash. Default: 0.2 bohr. `DASH` may occur any number of times in input, each occurrence resets the dash-length for the items that follow.

Items to be plotted

The remaining part of input has the format:

```
FILE filename
item {factor}
item {factor}
...
FILE filename
item {factor}
...
(etc.)

END INPUT
```

Each `FILE` key requires the name of a file. This file must not yet exist and will be created by `cntrs` as an ASCII file on which to write plot data. All 'items' until the next occurrence of `DASH` will be combined into one quantity for the contours of which the plot data are generated.

Each `item` must be of the form `Section%Variable` and must in this way correspond to one of the variables on `TAPE41` (*case-sensitive!*), see the description in the chapter about `densf`. All items that belong to one `FILE` will be added up, each one multiplied by its `factor` (default: 1.0), to the quantity to be plotted. In this way you can generate contours for instance of density differences or a summation of densities.

Memory usage

The default value for the maximum amount of memory that can be used by the program to perform the required computations is chosen and defined at the installation of the whole package and is identical to the value for the main programs (`adf`, `band`). As for these programs, the actual value can be adjusted for any particular run with certain keywords, in exactly the same way as for `adf` and `band`.

Apart from the maximum total amount of memory you can also adjust some related technical parameters. These should only affect efficiency issues and in general you should not adjust them.

Follows a description of the pertaining keys. All input keys pertaining to dynamical allocation are simple keys and specify the corresponding quantities in Mbytes. For the total amount of memory:

```
MAXMEMORYUSAGE      n
```

The technical parameters, defining how chunks of memory are allocated:

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. Their normal values are, for

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.

Input control of memory usage is applicable only if *dynamical* memory allocation has been turned on in the installation (which is the default, but you may have adjusted that).

Result

Each of the ASCII result files, the names of which are defined in input (key `FILE`), consists of a sequence of data blocks.

Each block consists of a number of records that contain two values ('x' and 'y') and it ends with a blank line.

Each block defines a contour by plot instructions as follows:

for the first {x,y}: start to plot at that point;

for each next {x,y}: (continue to) draw the contour to that point;

the blank line signals the end of the contour.

The last block does not correspond to a contour, but draws a rectangle around the whole picture.

3.3 ADFPLT: GUI FOR VOLUME MAPS

Adfplt is a tool [Autschbach, 1999 #1110] to graphically display orbitals, densities or potentials computed in a 2D or 3D grid. You can view the picture on your screen, or put it in a postscript file for printing, or send it to another device. This auxiliary program is installed with the other programs of the ADF package, but only if you have previously installed a few graphical libraries and you have correctly assigned the appropriate Environment Variable GKSGR_LIB to point to the directory where these libraries reside. See the Installation manual for details.

For more extensive graphics possibilities we recommend the ADF-GUI, which will be actively supported and developed. Details on the ADF-GUI are available from SCM (info@scm.com).

These (graphical) libraries are not part of ADF, but are available for free from their developers in Jülich, Germany. The libraries may not be available for all platforms, which then implies that cannot use *adfplt* on such machines. Consequently we cannot guarantee the continuity of their availability in any respect.

Adfplt requires a TAPE41 file – with that name – computed by *densf*, using a 3D grid. You start the program by invoking its executable with as argument the data you want to display.

```
| $ADFBIN/adfplt Sec%Var
```

Sec%Var must reference existing data on your TAPE41 file. To get a survey of all data on TAPE41, use the *pkf* facility.

```
| $ADFBIN/pkf TAPE41 > t41_ascii
```

You can then inspect the ascii dump of the TAPE41 contents to determine what you can display.

After having typed the command, you get a list of available output devices to send the picture to. Select one of them by typing the indicated number, for instance 211 to get the picture on your screen. (The appropriate choice may depend on your system settings, but a little try-and-error will soon help you on your way).

You may add options in the call of *adfplt*.

```
| $ADFBIN/adfplt -opt1 -opt2 [...] Sec%Var
```

You get a list of all available options by typing

```
| $ADFBIN/adfplt -help
```

In the current implementation, each run is one-time shot. To get another picture you have to start the program again.

3.4 DOS: DENSITY OF STATES

The auxiliary program *dos* computes various types of densities-of-states (DOS) for a user-specified energy interval.

dos requires an ASCII input file where the user specifies the items to be calculated and computational details, plus the standard result file TAPE21 from an *adf* calculation. The latter file must be present as a local file with name TAPE21 in the directory where *dos* is executed.

dos produces as result one or more ASCII files with the density-of-states values. Error messages and computational info (if any) are written to standard output.

Introduction

The program *dos* gives information on the number and character of one-electron levels (molecular orbitals) as a function of the (orbital) energy. The total density of states $N(E)$ is a well known concept in electronic structure theory of infinite systems (crystals). $N(E)dE$ denotes the number of one-electron levels (orbitals) in the infinitesimal energy interval dE . The total density of states (TDOS) at energy E is usually written as

$$N(E) = \sum_i \delta(E - \varepsilon_i) \quad (3.3.1)$$

where the ε_j denote the one-electron energies. So the integral of $N(E)$ over an energy interval E_1 to E_2 gives the number of one-electron states in that interval. Usually the δ -functions are broadened to make a graphical representation possible.

When the δ -functions are multiplied by a weight factor that describes some property of the one-electron state ϕ_i at energy ε_j various types of densities-of-states are obtained that provide a graphical representation of the state character (orbital character) as a function of one-electron energy.

In calculations on finite molecules the total density of states as a function of (orbital) energy may also be useful, but the main use of various types of densities-of-states is to provide a pictorial representation of Mulliken populations. The weight factors employed are related to the orbital character determined by means of a Mulliken population analysis *per orbital* (see below). The program *dos*, therefore, provides the same information as can be generated by the ADF program – a population analysis per orbital – but *dos* enables an easy graphical representation and is particularly useful when there are many one-electron levels, for instance in calculations on

clusters. You can obtain a simple view of the character of the orbitals in a certain energy range. You can also find out in which orbitals (at which energies) certain basis functions or fragment orbitals give a large contribution, and whether such contributions are bonding, nonbonding or antibonding with respect to particular bonds. Such information is provided by *dos* in the form of (weighted) density of states values over a user-specified energy range, which can for instance be plotted by *gnuplot*.

The following options are available for computations by *dos*:

- TDOS: Total Density of States
- GPDOS: Gross Population Density of States
- OPDOS: Overlap Population Density of States
- PDOS: Projected Density of States

The total density of states (TDOS) has large values at energies where there are many states per energy interval.

The GPDOS (Gross Population based Density Of States) of a function χ_μ (or a sum of such functions) has large values at energies where this function (these functions) occur(s) in the molecular orbitals.

The PDOS of a function χ_μ provides similar information, but with the projection of χ_μ onto the orbital ϕ_i as weight factor for the importance of χ_μ in the orbital ϕ_i .

The OPDOS (Overlap Population based Density Of States) between χ_μ and χ_ν has large positive values at energies where the interaction between them is bonding, and negative values where the interaction is antibonding. An example of the use of these plots is provided in `ADDIN ENRfu [1]`.

We review below the Mulliken population analysis, and then describe the forms of density of states analysis performed by *dos*. Finally an input description of *dos* is given.

Mulliken population analysis

The orbitals `EMBED Equation.3 * MERGEFORMAT` with energies e_i are expanded in basis functions `EMBED Equation.3 * MERGEFORMAT`, which leads to the definition of density matrices P_i describing orbital densities, from which the total density matrix can be constructed:

$$\text{EMBED Equation.3} \tag{3.3.2a}$$

$$\text{EMBED Equation.3} \tag{3.3.2b}$$

$$\square \text{EMBED Equation.3} \square \square \square \quad (3.3.2c)$$

Here μ and ν run over the basis functions, which may be either primitive functions, or combinations of primitive functions, for instance the SCF orbitals of atoms or larger fragments.

The Mulliken population analysis provides a partitioning of either the total charge density or an orbital density. The total density is written as

$$\rho = \sum_{\mu\nu} P_{\mu\nu} \chi_{\mu}(\mathbf{r}) \chi_{\nu}(\mathbf{r}) = \sum_{A \leq B} \sum_{\mu \in A} \sum_{\nu \in B} P_{\mu\nu} \chi_{\mu} \chi_{\nu} = \sum_{A \leq B} \rho_{AB} \quad (3.3.3a)$$

$$\rho_{AB} = \sum_{\mu \in A} \sum_{\nu \in B} P_{\mu\nu} \chi_{\mu} \chi_{\nu} \quad (3.3.3b)$$

The total number of electrons, $N = \int \rho(\mathbf{r}) d\mathbf{r}$, is now partitioned over the atoms by assigning an overlap population $P_{\mu\nu} S_{\mu\nu} + P_{\nu\mu} S_{\nu\mu}$ for one half to the atom A of χ_{μ} and one half to atom B of χ_{ν} ,

$$N = \int \rho(\mathbf{r}) d\mathbf{r} = \sum_{\mu\nu} P_{\mu\nu} S_{\mu\nu} = \sum_{\mu} GP_{\mu} \quad (3.3.4a)$$

$$GP_{\mu} = \sum_{\nu} P_{\mu\nu} S_{\mu\nu} \quad (3.3.4b)$$

GP_{μ} is the gross population of χ_{μ} . It contains the net population $P_{\mu\mu}$ and half of each total overlap population $P_{\mu\nu} S_{\mu\nu} + P_{\nu\mu} S_{\nu\mu}$ between χ_{μ} and χ_{ν} . Summing the gross populations over the functions $\mu \in A$ yields the total number of electrons assigned to atom A , or the gross population of atom A , GP_A , and hence the gross charge Q_A of atom A ,

$$GP_A = \sum_{\mu \in A} GP_{\mu} \quad (3.3.5a)$$

$$Q_A = Z_A - GP_A \quad (3.3.5b)$$

The overlap population $OP_{\mu\nu}$ between two functions and the overlap population Q_{AB} between two atoms are defined in an analogous manner,

$$OP_{\mu\nu} = P_{\mu\nu} S_{\mu\nu} + P_{\nu\mu} S_{\nu\mu} \quad (3.3.6a)$$

$$Q_{AB} = \sum_{\mu \in A} \sum_{\nu \in B} OP_{\mu\nu} \quad (3.3.6b)$$

These quantities can be evaluated for a single orbital density, $N = 1 = \int |\phi_i(\mathbf{r})| d\mathbf{r}$. The gross population $GP_{i,\mu}$ of a function in a specific orbital density $|\phi_i(\mathbf{r})|^2$ is then associated with the fraction of the orbital density belonging to that function (or the percentage χ_μ character of orbital ϕ_i), and the overlap population $OP_{i,\mu\nu}$ gives an indication of the strength of bonding or antibonding between χ_μ and χ_ν in orbital ϕ_i ,

$$GP_{i\mu} = \sum_{\nu} P_{i,\mu\nu} S_{\mu\nu} = \sum_{\nu} C_{\mu i} C_{\nu i} S_{\mu\nu} \quad (3.3.7a)$$

$$OP_{i,\mu\nu} = P_{i,\mu\nu} S_{\mu\nu} + P_{i,\nu\mu} S_{\nu\mu} = 2C_{\mu i} C_{\nu i} S_{\mu\nu} \quad (3.3.7b)$$

Density of states analyses based on Mulliken population analysis

Total density of states

The total density of states TDOS at energy E is written as

$$\text{TDOC: } N(E) = \sum_i \delta(E - \varepsilon_i) \quad (3.3.8)$$

so the integral of $N(E)$ over an energy interval E_1 to E_2 gives the number of one-electron states in that interval. In practice the delta functions are approximated by Lorentzians,

$$\text{TDOS: } N(E) = \sum_i L(E - \varepsilon_i) = \sum_i \frac{\sigma/\pi}{(E - \varepsilon_i)^2 + \sigma^2} \quad (3.3.9)$$

A plot of $N(E)$ versus E reveals energetic regions where many levels are located. The width parameter σ determines of course the appearance of the plot. A typical value is 0.25 eV (used as default in *dos*).

Partial (gross population and projected) density of states

In order to find out if a given function χ_μ contributes strongly to one-electron levels at certain energies, one may weigh a one-electron level with the percentage χ_μ

character. We usually determine the χ_μ character by the gross populations, obtaining the GPDOS form of the partial density of states,

$$\text{GPDOS: } N_\mu(E) = \sum_i GP_{i,\mu} L(E - \varepsilon_i) \quad (3.3.10)$$

If the weight factor is determined by projection of ϕ_i against χ_μ , we obtain the projected density of states PDOS,

$$\text{PDOS: } N_\mu(E) = \sum_i \left| \langle \chi_\mu | \phi_i \rangle \right|^2 L(E - \varepsilon_i) \quad (3.3.11)$$

Overlap population density of states (OPDOS)

If the delta function representing orbital ϕ_i is weighed with the overlap population between χ_μ and χ_ν in ϕ_i , the overlap population density of states OPDOS is obtained,

$$\text{OPDOS: } N_{\mu\nu}(E) = \sum_i OP_{i,\mu\nu} L(E - \varepsilon_i) \quad (3.3.12)$$

If an orbital ϕ_i at energy ε_j is strongly bonding between χ_μ and χ_ν the overlap population is strongly positive and OPDOS(E) will be large and positive around $E = \varepsilon_j$. Similarly, OPDOS(E) will be negative around energy ε_j when there is antibonding between χ_μ and χ_ν in ϕ_i .

The OPDOS(E) has been used under the name COOP (crystal orbital overlap population) in Extended-Hückel solid state calculations by Hoffmann and coworkers [2].

Generalizations of OPDOS, GPDOS, PDOS

As observed above, the basis functions in the above expressions may be primitive basis functions ('Slater type orbitals'), but of course the formulas are equally applicable for other types of MO expansions. In *dos* the user may select either the expansion in primitive basis functions ('BAS') or the expansion in SFOs (Symmetrized Fragment Orbitals) for the DOS analyses.

It is also possible in DOS to treat a *set* of basis functions simultaneously. For instance, the GPDOS for a set of basis functions μ_1, μ_2, \dots is simply defined as the summation of the corresponding single-function GPDOS(E) values

$$N_{\mu\text{-set}}(E) = \sum_{\mu \in \mu\text{-set}} \sum_i GP_{i\mu} L(E - \varepsilon_i) \quad (3.3.13)$$

In a similar fashion the OPDOS can be defined for *two sets* of basis functions μ_1, μ_2, \dots and ν_1, ν_2, \dots as

$$N_{\mu\text{-set}, \nu\text{-set}}(E) = \sum_{\mu \in \mu\text{-set}} \sum_{\nu \in \nu\text{-set}} \sum_i OP_{i,\mu\nu} L(E - \varepsilon_i) \quad (3.3.14)$$

and finally for the PDOS we get in similar fashion

$$N_{\mu\text{-set}}(E) = \sum_{\mu \in \mu\text{-set}} \sum_i |\langle \chi_\mu | \varphi_i \rangle|^2 L(E - \varepsilon_i) \quad (3.3.15)$$

Input

The (ASCII) input for *dos* is keyword oriented. Reading input by *dos* terminates whenever it finds a line `END INPUT` or the end-of-file, whichever comes first.

Follows a list of keywords with their meaning. Generally keys may occur more than once and *the order in which they appear is relevant in some cases*. For instance the key `ENERGYRANGE` (which defines for what energy values to compute densities-of-states, see below) applies to all items that come after it in input until the next occurrence of `ENERGYRANGE`.

Energy scan values

ENERGYRANGE	{Npoint=nr}	{E-start=e1}	{E-end=e2 / E-
step=de}			

This specifies for which energy values the densities-of-states are computed that are specified *after* it in the input file and *until* the next occurrence of `ENERGYRANGE`.

`ENERGYRANGE` specifies the lower bound, upper bound and number of equidistant energy values (including end-points). All items are optional with defaults applying for those omitted.

The `E-end` and `E-step` values determine one another and must therefore not be specified both (or be consistent).

The initial defaults are:

```
nr=301
```

```
e1=-20
```

```
de=0.1
```

All energy data are in eV.

When values have been changed with the key `ENERGYVALUE`, the so-modified values are the defaults for the next occurrence of `ENERGYVALUE`.

Peak widening

The peaks in the DOS curves corresponding to the energies of the molecular orbitals are widened by a Lorentzian curve, the width of which can be adjusted.

```
| LORENTZIAN      width=width
```

Initial default width is 0.25 (eV).

As for `ENERGYRANGE`, the key `LORENTZIAN` may occur more than once and each occurrence sets the width for all items after it.

Result files

The computed densities-of-states are stored on one or more ASCII files, which have to be specified in input.

```
| FILE file
```

The key `FILE` may occur any number of times in input. Each time it occurs the specified file is opened by `dos`. The file must not yet exist and the new file will accumulate (ASCII) the densities-of-states data of all DOS items subsequently specified, until the next occurrence of `FILE`. The first occurrence of the key `FILE` must be given before any DOS specification (by the keys `TDOS`, `OPDOS`, `GPDOS`, `PDOS`, see below).

The format of the result file is such that it can be fed directly into *gnuplot*.

Densities of States

```
| TDOS { title }
```

TDOS instructs the program to compute the *total* density of states. `title` (optional) will appear as title to the section of corresponding Density-of-States data in the result file.

The other types of densities-of-states require block-type keyword input.

```

| OPDOS      { title }
| Ftype      numbers
| Ftype      numbers
| ...
| SUBEND
| Ftype      numbers
| Ftype      numbers
| ...
| END

```

Ftype

Specifies the type of basis functions to use in the MO expansions. If the primitive basis functions are to be used `Ftype` must be `BAS`. For the SFO representation `Ftype` must be one of the irreducible representations of the pointgroup symmetry. All `Ftype` values in the data block must be consistent: either all are `BAS` or all are irrep labels. The scope of this consistency requirement is the data block of the current key: in a next OPDOS data block, for instance, a different choice may be made.

numbers

Must be a sequence of integers referring to the basis functions to be selected, i.e. the ' μ -set' and ' ν -set' in (3.3.13) etc.

If `BAS`-type basis functions are selected the numbers refer to the overall list of all basis functions as printed in the output file of the *adf* run. If SFOs are selected the numbers refer to the SFO list of the pertaining symmetry representation *without the core functions*, see the *adf* output file.

SUBEND

Must be typed as such and separates the ' μ -set' and the ' ν -set': all records before `SUBEND` specify together the ' μ -set' and all records below `SUBEND` comprise the ' ν -set'. Each of these two sections may consist of any number of records.

The input for `GPDOS` and `PDOS` are similar, but simpler because only one set of functions (' μ -set') has to be specified, so there is no `SUBEND` in the data blocks for these keys.

```

| GPDOS      { title }
| Ftype      numbers
| Ftype      numbers

```

```
...  
END
```

```
PDOS { title }  
Ftype numbers  
Ftype numbers  
...  
END
```

The keys `GPDOS`, `OPDOS`, `PDOS` and `(TDOS)` may occur any number of times in input and in any order. Each time the `DOS` key occurs the current `ENERGYRANGE` and `LORENTZIAN` settings apply and the results are written to the current `FILE`.

Memory usage

The default value for the maximum amount of memory that can be used by the program to perform the required computations is chosen and defined at the installation of the whole package and is identical to the value for the main programs (*adf*, *band*). As for these programs, the actual value can be adjusted for any particular run with certain keywords, in exactly the same way as for *adf* and *band*.

Apart from the maximum total amount of memory you can also adjust some related technical parameters. These should only affect efficiency issues and in general you should not adjust them.

Follows a description of the pertaining keys. All input keys pertaining to dynamical allocation are simple keys and specify the corresponding quantities in Mbytes. For the total amount of memory:

```
MAXMEMORYUSAGE n
```

The technical parameters, defining how chunks of memory are allocated:

```
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. Their normal values are, for reals: 1

integers: 1/2
logicals: 1/64
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.

Input control of memory usage is applicable only if *dynamical* memory allocation has been turned on in the installation (which is the default, but you may have adjusted that).

One of the standard examples of the ADF package contains an application of *dos*, including a copy of the ASCII result file from *dos*. See also the *Examples* document.

4 OTHER

4.1 DIRAC: RELATIVISTIC POTENTIALS

The auxiliary program DIRAC, which is installed together with ADF, serves to compute relativistic frozen core potentials (and densities), necessary to apply the (scalar) relativistic option in ADF. The database *atomicdata* has a subdirectory *Dirac*, which contains input files for DIRAC for all atoms of the periodic table of elements. The *names* of the input files indicate the frozen core level: *Ag.3d* for instance is the input file for a calculation on a Silver atom with a frozen core up to and including the 3d shell (i.e.: 1s, 2s, 2p, 3s, 3p, and 3d). The frozen core level used in the DIRAC calculation defines the core data computed and should therefore match the frozen core level in the ADF Create run for the atom that it will be used for.

A DIRAC run with the inputs provided in the database involves a fully relativistic calculation on the atom (spin-orbit coupling, double group symmetries). It generates a file TAPE12 with the corresponding core potential and density (a table of values for a sequence of radial distance values). Other files produced by DIRAC should be removed after the DIRAC run; they are needed for other applications of the program but play no role here.

If you run DIRAC while a file TAPE12 already exists the computed core data will be written at the end of it, after the existing data. The program will assume, however, that the existing data on the file are also core-data from DIRAC runs, and may abort otherwise.

If a `CorePotentials` file is needed for an ADF calculation with the (scalar) relativistic option, the simplest approach is to subsequently run DIRAC for each of the involved atoms types. This builds up the TAPE12 file for this particular molecule. Then, specify in the ADF input which sections correspond to the distinct atom types.

Alternatively, if you frequently perform relativistic ADF runs, with many different types of atoms, you might, once and for all, construct one big TAPE12 file, containing the core potentials of all atoms that you may ever need, and use that file again and again. Of course you need then to remember which section numbers correspond to which atoms.

Implied options

The DIRAC calculations imply the *local* Density Functional in its simple X-alpha approximation without any gradient corrections. Not the *scalar* relativistic but the *fully* relativistic Hamiltonian is used, including spin-orbit coupling. In ADF you may use the

scalar relativistic Hamiltonian and most users will employ a more sophisticated LDA than X-alpha, such as the default vwn (Vosko, Wilk, Nusair) formulas, and may in addition routinely apply gradient corrections. The core potential may not exactly match the Fock operator applied in the molecular calculation. The effect is very small and one can neglect the discrepancy.

Input

The ASCII input files for DIRAC, as available in the database directory *adfhome/atomicdata/Dirac*, have a structure as described below. With this information you should be able to construct alternative input files, with other frozen cores for instance.

- 1 Title (60 characters at most). Plays no role
- 2 Ngrid, Nshell, rmin, rmax, Z, Xion, Anuc
Ngrid=number of radial grid points, in which the core potentials are computed.
Nshell=number of atomic orbital shells
rmin, rmax=minimum and maximum radial grid values
Z=nuclear charge
Xion=net charge of the 'atom'
Anuc=atomic weight
- 3 Pinit, Pfinal, eps, del, delrv
Pinit, Pfinal= initial and final density iteration averaging factors. Each iteration cycle changes the actual averaging factor by taking the average of the previous and the final one, starting with the 'initial' one.
eps=Exp(-sqrt(eps)) is set to zero, so eps determines the exponential underflow control.
del=absolute convergence criterion for orbital eigenenergies.
delrv=convergence criterion on the potential (multiplied by the radial distance *r*).
- 4 Idirc, Nmax, Ndebu, Nprin, Ipun, Ircor, Iwcor
Idirc=zero for non-relativistic, otherwise one.
Nmax=maximum number of iterations allowed to reach convergence.
Ndebu=non-zero for additional output (for debugging purposes mainly)
Nprin=print parameter. Use 2 or larger to get the orbitals printed.
Ipun=punched output is produced if Ipun is non-zero. (out-of-date)
Ircor=number of core orbitals from the fully relativistic run, to be kept frozen in the subsequent (if any) first-order perturbation calculation.
Iwcor=number of core orbitals used to construct the core density and core potential, that are output on TAPE12. So, here you specify the relativistic core.
- 5 Xalph, Xlatt, Rnuc
Xalph= Exchange parameter in X-alpha formalism.
Xlatt=Coulomb tail parameter
Rnuc=size of nuclear radius, in bohr. If set to 1.0 or larger, it is recomputed as $0.0000208 \cdot \text{Anuc}^{1/3}$

6 For each orbital shell:

N, L, J, E, Z, D

N, L, J = The usual orbital quantum numbers. J is used only for relativistic runs.

E = Initial estimate of orbital energy, in atomic units.

Z = Number of electrons on the shell

D = Initial estimate of the error in the orbital energy

7 Icorp, Npcl, Demp, Peps

Icorp = If 1 (one): Do a first order perturbation calculation after the fully relativistic run. This option plays no role in the current application for ADF.

Npcl = Maximum number of cycles in the perturbation calculation.

Demp = Damping factor in the perturbation iterations

Peps = Convergence criterion in the perturbation iterations.

4.2 BOBIT

When the package is installed you can use the output browser

```
$ADFBIN/bobit
```

This command will start the output browser BOB showing the output file corresponding to the result file you have chosen for analysis. Following is a short description of some of the features of the Basic Output Browser BOB.

To run it you will need the `Tcl/Tk` package. The environment variable `$SCMWISH` should be set to the complete path to the `wish8.0` executable (part of the `Tcl/Tk` distribution, available from <http://www.scriptics.com>).

Also note that you need to set your `$DISPLAY` environment variable (on UNIX the `Tcl/Tk` toolkit uses `X` for its graphical display).

Introduction

In the BOB main window the complete output file is shown. Using the scroll bar you can move around in a linear version. At this point using BOB is equivalent to using any WYSIWYG-editor.

Initialization

When loading a file, either at startup or by using the File/Open command, BOB scans the entire file and locates many special points of interest in this file. All these points are linked to menu entries. These menu entries are organized to make it easier to find the entry you are looking for. Furthermore, still during this initialization phase, BOB will try to detect the different parts in an output file. For example, many outputs will consist of the output from one or more ADF Create runs followed by one or more molecular runs. BOB will find these parts and put them in its 'Include' menu.

Include Menu

The Include menu is often very informative since it provides at a glance the overall structure of the file you are looking at. If you select any of the parts BOB will jump to that part in the file, *and* it will restrict all operations (except a new Restrict command) to this part. This makes it very easy to study a single part of the output file without getting lost in another part.

Other Menus

The other menus (Properties, Iterations..., except File and Edit) contain the special markers ('Points of Interest') which have been detected when opening the file. Choosing one of their menu items will jump to that item, or beep if that is not possible (because you included only part of the output to view using the Include menu). The search will wrap around.

Using the BOB menus you can find easily what you are looking for. The property menu gives access to all calculated properties, including excitation energies, polarizabilities, NMR results and bonding energy analysis.

Click on Marked Items

The points of interest in the file have also been marked visually. Clicking on one of them will jump to the next point of interest of the same kind. Shift-Clicking will do the same but backwards.

Customizing BOB

The points of interest BOB knows about are easy to change. They live in the file `$ADFBIN/search.dat`. If you are not content with the current points of interest and would like to change them, or to add your own, just open this file and customize. The format should be easy to figure out by looking at the existing entries. In case you are wondering: `regexp` stands for 'search using regular expression syntax', and `exact` stands for 'search the following string exactly'. All searches are case sensitive.

REFERENCES

1. P.J. van den Hoek, E.J. Baerends, and R.A. van Santen, *J. Phys. Chem.* **93**, 6469 (1989).
2. R. Hoffmann, *A chemist's view of bonding in extended structures* (VCH Publishers, New York, 1988) .
3. G. Schreckenbach and T. Ziegler, *J. Phys. Chem.* **99**, 606 (1995).
4. G. Schreckenbach and T. Ziegler, *Int. J. Quantum Chem.* **61**, 899 (1997).
5. S.K. Wolff and T. Ziegler, *J. Chem. Phys.* **109**, 895 (1998).
6. S.K. Wolff, T. Ziegler, E. van Lenthe, and E.J. Baerends, *J. Chem. Phys.* , (1998 submitted).
7. H. Fukui, *Mag. Res. Rev.* **11**, 205 (1987).