juRS User Manual

1.0.1 Building the Application

juRS is a pure Fortran code with coding standards newer than FORTRAN90, i.e. it uses some features of the '95 and 2003 standards. Before compiling, the build directory should contain the following items: Makefile, system.make, README and the directory src/ containing the .F90 and .F95 source files.

The Makefile usually does not need to be changed for building the application. All configurations such as preprocessor, compiler linker and their flags are defined in the architecture-specific include file system.make. If there is an example include file as e.g. system.make.INTEL, you can copy that file or use a soft link to it (recommended):
ln -s system.make.INTEL system.make
For preprocessing the .F90 source files, a C-preprocessor (e.g. cpp) is applied. Some source files carry the extension .F95. These source files contain simultaneous code for real/complex and will be preprocessed twice with an additional flag R1_C2. The first time cpp -D R1_C2=1 $< generates the head part of the final .f90-source file with routines that accept real as data type for wave functions, the second time cpp -D R1_C2=2 $< generates routines for complex wave functions and the tail part of the preprocessed .f90-file. Each source file contains exactly one module.

The code has been developed for double precision that is activated via the compiler flags (Intel:-r8, IBM:-qrealsize=8). In principle, one could also compile in single precision, for that, -D SINGLE_PRECISION should be added to the preprocessor flags in order to replace library calls (MPI, LAPACK, ScaLAPACK) with the proper interface.

On the IBM BlueGene class supercomputers and similar architectures, add -D BLUEGENE to the preprocessor flags. Furthermore, the MPI library can be replaced with dummy routines leading to a serial version by defining -D NOMPI. On systems where ScaLAPACK is not installed, -D NOScaLAPACK will replace the ScaLAPACK calls by calls to the corresponding LAPACK routines. -D NOLAPACK can be used for testing the compilation and linking of all the others code parts but does not lead to a fully usable application. -D TIME will show additional timings in the log-output. During development, some additional checks for NaN (Not-a-Number) can be included into the code with -D NaN_SEARCH. Finally, but not recommended for production runs, the user may want to activate additional and partially also experimental checks and extra outputs in all modules with -D DEBUG_ALL. -D USE_SYMMETRY is neccessary for special symmetries of small unit cells that can be exploited when run without domain decomposition.

The binary executable will be called ./paw.

1.0.2 Get Started

Once the application `./paw` was successfully compiled and linked, we may start with the help function `./paw --help` which explains how to generate an example input file and more command line options.

```
usage: ./paw  <inputfile>  [options]
  options:
  -o   --output [<file>]  redirect output
  -l   --load             load existing wave functions
  -L   --Load             load existing density files
  -ps  --preset <line>    lines read before input file
  -ov  --override <line>  append line to input file
       --real             enforce real wave functions
       --complex          enforce complex wave functions
  -i   --ignore <arg>     the next arg will not be parsed

  -g   --generate         generate PAW data file
  -e   --element <Z>      show default element configuration
  -t   --test <module>    debug function testing modules
  -cm  --check [<np>]     only check the input file
       --example          writes an example input file
  -kw  --keywords         lists all input file keywords
  -v   --version          shows the version number
  -h   --help             this usage help
```

The first eight of these command line options are parameters which alter the run properties, the others have a toolbox functionality, i.e. the application will perform the desired action and quit.

1.0.3 Input Files

`juRS` v14.* tries to load the required chemical element specific PAW data stored in `pawdata.00Z` where `00Z` is the three-digit atomic number Z^a (integer), e.g. `pawdata.047` contains the PAW data for silver. These files are `ASCII`-formatted. A description of the format can be found in the module `pawdatafile`.

The system to be calculated is described in a single input file. Three different elements of the input syntax are to be distinguished:

- Keyword statements 1.0.4
- Variable definitions 1.0.5
- Blocks 1.0.6

The following subsection will describe each syntax element in detail.

1.0.4 Keyword statements

A keyword statement consists of one line in the input file that is lead by the keyword and followed by one or three values, depending on the quantity to be set. Blanks as separation are required here, no equality-sign. Lines starting from `#`, `!` or `%` are comments and will be ignored. For all keyword statements, the latter statement will overwrite the earlier. However, multiple mentioning of a keyword will launch a warning. In vector quantities (requiring three values) all vector entries are set to the same value if the keyword is followed only by one value.

A list of all keywords can be found by `./paw --keywords` or `./paw -kw`. Their syntax is explained further in Table 1.1. Keywords have a long name, usually containing an underscore which allows to write very verbose input files as shown in Section 1.0.10. However, the short versions have the same effect. For reasons of backwards compatibility some keywords are also aliased with keyword names of older versions.

1.0.5 Variable definitions

`juRS` is equipped with a built-in variable environment that helps to generate flexible input files. All variables are scalar real values. To define a variable yourself, simply write `$<var> = <expression>` in one line. Here, `<var>` stands for a variable name of your choice, max. 32 characters long. The usual identifier for variables is the dollar sign `$`. This may lead to interference with script languages, so one may use `@` instead. The blanks to the left and right of `=` are important. A line that reads `$<var>` only will show the variable's value in the output. Here, `<expression>` must a float value, a fractional value (with `:`), a predefined constant `<cst>` (see Table 1.2), a defined variable, a function of values or a math operation. Valid math operators are `+`, `-`, `*`, `:` , `^` (or `**`). The backslash for divisions cannot be used. No operator between two words means multiplication. Operators only lead to valid expressions in a context `<value> <operator> <value>`. The minus sign therefore cannot be used as a unary operator or function, please use `0 - $<var>` instead. Valid functions are `exp`, `cos`, `sin`, `abs`, `sqrt` and `log`. Functions lead to valid expressions in a context `[<prefactor>] <function> <argument>`. An expressions may only consist of up to three strings. The usage of parenthesis is not supported. A string as `a:b` without blanks before and after the colon operator `:` will be interpreted as fractional number $\frac{a}{b}$.

1.0.6 Blocks

To input a list of dynamic length, blocks are defined by block keywords. A line containing only the block keyword opens the block or closes the block, depending on the application's previous state. In between block opening/closing keywords, block items are added to the corresponding list. Blocks are implemented for atoms and $\vec{k}$-points: Block keywords are

- `kpoints` $\vec{k}$-points for Brillouin zone sampling x y z w

<keyword>	Types	Default	Description
ascale	r	1.0	scale absolute atomic positions by this factor
bands	i	[**auto**]	total number of bands
bc	w [w w]	iso iso iso	boundary condition {isolated,periodic}
cell	i [i i]	[**required**]	extend of the orthorhombic cell
charged	r	0.0	additional electrons in the system
comment	string		show this string in the output, allows detailed comments or notes
cscale	r	1.0	scale cell by this factor
dg	i i	4 5	DoubleGrid order N and meshrefinement M
domains	i [i i]	1 1 1	number of MPI processes for grid parallelization in domain decomposition
efield	r [r r]	.0 .0 .0	electric field, only directions where bc is isolated
element	string		projector augmented wave configuration, paw -e <ChemSym> shows defaults
grid	i [i i]	cell/spacing	total numbers of grid points per direction
forces	w [w w]	on on on	calculate forces {on,yes,1,off,no,0}
hfield	r	.000	magnetic field, shifts local spin-dependent potential by this energy up or down
kmesh	i [i i]	[32·cell^{-1}]	automatic $\vec{k}$-point sampling in periodic directions
kshift	r [r r]	.0 .0 .0	origin shift for kmesh
md	crit		criteria for molecular dynamics or structural relaxation
mixing	r [w [i]]	0.25	mixing ratio, method and history length optional

<keyword>	Types	Default	Description
nfd	i [i i]	4 4 4	FiniteDifference order N_{fd} for KohnSham, Nabla and Poisson
nscale	i	2	ratio of potential grid g_{e} over the density grid g_{d}
spacing	r	0.25 Ang	grid spacing suggestion (to be adjusted)
origin	r [r r]	.0 .0 .0	shift of the coordinate system in fractions of cell
params	w	light	choose a set of convergence parameters {light,right,tight}
path	w	.	path to the pawdata.00Z files (use \ instead of /)
pdos	w	none	projected DoS { alm, al, a, slm, sl, s, z }
poisson	crit		criteria for iterative solution of the electrostatic equation
scf	crit		criteria for self-consistency iterations
spin	w	1	treat electron spin {1,integrated,polarized,2}
solver	w	diis	eigensolver method or scheme {diis,cg,sr,speed,stable}
symmetry	w	none	do not use!
temp	r	.001 Ha	electronic smearing temperature
units	w [w]	Ang eV	output units { aB, Ang, nm, pm } and { Ha, eV, Ry, kJ, ... }
wfs	crit	1 1 1E-4	criteria for eigensolver iterations
xc	w	PZ	exchange-correlation functionals {VWN,PZ,PBE}

Table 1.1: Keywords and their syntax. The required key word cell has no default values since the application will not run without its specification. The types are abbreviated as i:integer, r:real numbers and w:words. Type crit stands for the input syntax of convergence criteria which are specified e.g. poisson max 333 min 3 < 1E-7 for the max. and min. number of iterations and the convergence threshold. Default values depend on the params setting.

<cst>		Internal Value	Name
Ang	Å	1.8897261247728	Ångström
nm	nm	18.897261247728	nanometer
pm	pm	0.0188972612477	picometer
Pi	π	3.1415926535898	
Deg	$\frac{2\pi}{360}$	0.0174532925199	degrees, $^\circ$
sqrth	$1/\sqrt{2}$	0.7071067811865	
sqrt2	$\sqrt{2}$	1.4142135623731	
sqrt3	$\sqrt{3}$	1.7320508075689	
sqrt5	$\sqrt{5}$	2.2360679774998	
Kel	K	$3.1668294 \cdot 10^{-6}$	Kelvin
eV	eV	27.210282768626	eVolt
meV	meV	0.0272102827686	milli eVolt
Ry	Ry	2	Rydberg
Ha	Ha	1	Hartree
aB	Bohr	1	Bohr radius

Table 1.2: Predefined constants for easy input.

- `kpath` edges for a band structure path sampling $x\ y\ z$
- `atoms` for absolute atomic positions $Z\ x\ y\ z\ [m_x\ m_y\ m_z\ \sigma]$
- `atoms_fractional` for relative positions $Z\ \frac{x}{L_x}\ \frac{y}{L_y}\ \frac{z}{L_z}\ [m_x\ m_y\ m_z\ \sigma]$

where Z may be either the atomic number (`integer`) or the chemical symbol (e.g. `Mg` instead of `12`, element symbols are case sensitive). L_i with $i \in \{x, y, z\}$ are the extends of the orthorhombic cell (see `cell`). For the input of atoms, we may optionally add three logic values $m_i \in \{$`T`,`F`$\}$ for geometry relaxations. The default m_i is `T`, so atoms are usually free to move. If the three m_i are defined, the user may also define a spin-flip number $\sigma \in [-1, 1]$. If the element data contain a magnetic orbital occupation (run `grep PartialWave pawdata.*` to reassure a different occupation of `up` and `dn`), the spin-flip number σ determines the sign of the magnetization of the inital orbitals in a spin-resolved calculation. Default σ is `+1`.

A single block keyword followed by the name of an existing external file will open the file and try to read block items from it.
Useful examples: `atoms coords.xyz` or `kpoints kpts`

In order to check the input file for syntax and perform some quick analysis of the atomic geometry we can run with `--CheckMode` (most useful in serial). Invoke the executable as `./paw <inp> --CheckMode` or `-cm` where `<inp>` is your input file. Then search the output for the occurrence of `WARNING!`. For the distribution of parallel MPI processes, we add a positive integer, i.e. `--CheckMode <np>`, which stands for the total number of processes. The output will show how the parallelization would factorize this number.

A continued calculation may load existing densities (`--Load`) and/or wave functions (`--load`) from the file system. See the following Section 1.0.7 for a detailed description.

1.0.7 Output Files

The name of the input file is the project name. All non-temporary output files will have the same name as the project plus an extension that depends on the type of output. Usual file name extensions are

- `.dos` Density of states (`ASCII`)
- `.bst` Plotable band structure (`xmgrace` v5.0+)
- `.frc` Atomic positions and forces (`ASCII` in atomic units)
- `.adm` Atomic density matrices $D^a_{ij\sigma}$ (`ASCII`)
- `.rho` Smooth valence density $\tilde{n}^\sigma_{\mathrm{v}}(\vec{r})$ (`binary`)
- `.wfs` Smooth valence wave functions $\tilde{\Psi}_{n\sigma\vec{k}}(\vec{r})$ (`binary`)

In `CheckMode` some more files for the visualization are created

- `.xyz` Atomic positions (`ASCII` in Å)
- `.pov` Geometry for `povray` (`ASCII`)

Both types of binary files (density and wave functions) are created by `MPI`-collectives that write files in `BIG ENDIAN`. A leading header of 1024 Bytes proceeds the data arrays. The header is encoded in `ASCII` and holds information about the calculation such as file name, stored quantity, numbers of grid points, cell extends, number of bands, number of spins, number of $\vec{k}$-points and a time-stamp of the file generation time. When densities are to be loaded, the number of grid points needs to match the number of grid points specified in the density file. Significant deviations of the cell sizes will launch `WARNING!`s. For loading wave functions, the number of bands, spins and $\vec{k}$-points found in the file may be larger than the requested number, however, this will also be warned.

1.0.8 Global Application Configuration

Most configurations are defined in `mod_configurations.F90` and, if necessary, have to be adjusted before compiling.

1.0.9 Limitations

The restriction to nearest-neighbor communication during data-exchange introduces a limitation. The lower limit for the number of grid points per domain is the number of finite-difference neighbors N_{fd} used for the kinetic energy stencil in the Kohn-Sham Hamiltonian. This defines a lower boundary to the parallelization in domain decomposition. The finite-difference order of the Laplacian in the Poisson equation of the electrostatic part may be twice as large since the density grid contains at least twice the number of grid points per direction in each domain compared to the wave function grid. The syntax for controlling N_{fd} is explained in Table 1.1 (`nfd`).

1.0.10 Example input

Run `./paw --example` to get a simple example input file. Fur further practice with the variable environment, look at this example: create a file called `inp` with this content:

```
cell 12 Ang

$r = 76 pm
$a = 45 Deg
$x1 = $r cos $a
$y1 = $r sin $a
$x2 = 0 - $x1
$y2 = 0 - $y1

atoms
Li  $x1  $y1  0.0
Li  $x2  $y2  0.0
atoms
```

Now invoking `./paw inp --CheckMode` or equivalently `./paw inp -cm` tells the user that `./pawdata.003` could not be found. Re-running `./paw inp -cm --gen` generates a PAW data file for Lithium with the default PAW configuration for this species if extended features are activated in your version. Check `./paw -e Li` to see the default PAW configuration string for Lithium. Running again `./paw inp -cm` performs a check of the geometry specified in the input file `inp`. To start the calculation, remove the CheckMode flag `-cm`. The file name does not need to be `inp` but `inp` is a placeholder for the project name. The input file of a project carries the project name `<projectname>` only. Any output file will be named `<projectname>.<ext>` where the file extension `<ext>` depends on the kind of output.

Version 14.03 (2014-03-04) juRS_bug_report@fz-juelich.de

PAW generation

The configuration string for the details of the PAW data set should be specified in each input file to make sure the results are the same. This string contains information about the inital occupation numbers in the atomic self-consistent calculation, numbers of projectors in each ℓ and the states which contribute to the frozen core density. Finally, also the augmentation radii (may vary slightly with ℓ) are specified. For example Stronium will appear as

```
element  Sr  4s 2 5s 2 4p* 6 4d | 2.44
```

The keyword `element` expects an element symbol (like `Sr`, case-sensitive) or an integer atomic number. The symbol or atomic number is followed by the electronic configuration string. Here, the orbital specifyer `4s` sets the $4s$-state to be included in the set of valence states. Automatically, $1s$, $2s$ and $3s$ are set to be core states. The following `2` is an occupation number for the all-electron self-consistent atomic calculation, interpreted as float 2.0. If no occupation number is given, the occupation is zero, if a single number is given, both spin-channels are given half of the occupation and if two numbers are specified, we may define a magnetization, as e.g. in the d-cannel of Osmium.

```
element  Os  6s* 2 5p 6 6p 5d 5 1 | 2.55 2.64 2.35
```

A second valence orbitals with the same ℓ as in `element Sr 4s 2 5s 2` [...] effects that two partial waves are used for this ℓ. A second partial wave without an extra orbital can also be added by a star as in `element Os 6s* 2` [...]. Here, an energy derivative is used as an unorthogonalized start guess for the second partial wave. Note that `element Sr 4s 2 5s*` or `element Sr 4s* 2 5s` would activate a third set of partial waves and projectors which is not recommended. The example of Osmium shows how different augmentation radii for s, p, d, ... are specified after the `|`. The radii are always given in `Bohr` atomic units, multiplyiers like `Ang` cannot be used here! When a magnetization is induced in the occupation numbers, please make sure that the first number is the majority spin as e.g. `5d* 5 1`. A spin-polarized atomic all-electron calculation will be performed, if spin is activated in the input file and any orbitals occupation number differs in $\uparrow$ vs. $\downarrow$.

When a `pawdata.00Z` file is generated (always in `./` in constrast to reading the file during the code execution where we can specify a path to find the `pawdata`-files), there is a possiblity to store the self-consistent potential. Create a folder `pot/` in the current work directory before the generation. A file `rV.00Z` with the local spherically symmetric potential $r \cdot V(r)$ is output to this folder and re-used in a later generation process.

When you want to compare the logarithmic derivatives of the true and of the augmented potential in your newly generated PAW data set, run the generation combined with the CheckMode which is activated by the command line option `-cm 2` (or higher integers). Look for the file `Sy_logder` with `Sy` being the corresponding element symbol. Furthermore, when `pawdata.00Z` is present and a folder `partial_waves/`

is detected running with `-cm 2`, the radial representation of projectors, true and smooth partial waves is written to separate plotable files in this folder.

Usage of xml PAW setups

It is possible to load PAW setups in xml format (extened markup language) defined here:
`https://wiki.fysik.dtu.dk/gpaw/setups/pawxml.html`
To use the xml PAW setups invoke the program with the `-xml` flag. `./paw copper -xml` Just after loading, informations about the pseudopotential are printed to standard output. The program looks for xml files with name `pawdata.0xx.xml` for element number `xx`.

Check of the PAW setup

When invoking the applicaiton in CheckMode, the PAW setups are checked. The logarithmic derivative is computed before ad after projector filtering and the spherically symmetric PAW hamiltonian is diagonalized. This allows to check quickly if the pseudopotential has ghost states. For example `./paw copper -xml -cm 2` or `./paw copper -cm 2`

SoftSwitches

As mentioned above, there is a variable environment such that the user may define variables and use them in the input but also everywhere in the code after the input files are parsed. Therefore, SoftSwitches are used to control add-on features of the code, such as `$visualize_all_wfs = 1` write a phase-to-color-coded `.bmp`-file for each wave function and each x-y-plane of grid points. Similarly, `$visualize_all_wfs_x = 1` and `$visualize_all_wfs_y = 1` work for y-z-planes and x-z-planes, respectively, but require more memory.

The SoftSwitch `$indep = 8` allows a parallel calculation of 8 different lattice parameters around the one defined in `cell_size`. The step size is controlled by `$bulk` which defaults to 0.01, i.e. one percent, so will `$indep = 8` calculate from -3% to +4% in steps of 1%. Using less than `$indep = 4` parallel independent calculations will fail to fit a cubic curve and also the parabola fit will be fully determined.

The SoftSwitch `$xsf_density = 1` write a `.xsf`-file of the structure with a three-dimensional data grid of the smooth density. This action is performed instead of starting a calculation and an existing density file is required. Internally, one may switch `$xsf_max_z_dist = 13.3` to limit the density in z-direction e.g. by 13.3 Bohr and `$xsf_use_norm = 1` to scale the density down by the cell volume.

After the self-consistency cycles have reached convergence or the maximum number of iterations the Kohn-Sham eigenvalues are displayed in the output unless there are more than 1000. Change this limit by defining `$display_states` with a positive or negative integer value. For the display in the output, only the absolute value counts.

Zero or negative values will cause a file `<project>.eig` to be created with a detailed list of all eigenvalues.

The SoftSwitch `$write_wfs` defaults to 1 when not specified. Specifying it with zero or negative values will disable the output of the Kohn-Sham wave functions at the end of a calculation.

Minimal Executable

Depending on the circumstances of the computer system it can be useful to have a small executable file `./paw`. In particular in the case of statically linked binaries the preprocessor option `-D EXTENDED` should not be defined. This omits a large set of test functionality and a number of add-on modules which are not needed during production runs, including the build-in PAW data generation and the module self-tests.