The PSML format and library for norm-conserving pseudopotential data curation and interoperability

Alberto García[a,*], Matthieu Verstraete[b], Yann Pouillon[c], Javier Junquera[c]

[a] *Institut de Ciència de Materials de Barcelona (ICMAB-CSIC), Campus UAB, 08193 Bellaterra, Spain*

[b] *nanomat/Q-MAT/CESAM, Université de Liège, Allée du 6 Août 19 (B5a), B-4000 Liège, Belgium*

[c] *Departamento de Ciencias de la Tierra y Física de la Materia Condensada, Universidad de Cantabria, Cantabria Campus Internacional, Avenida de los Castros s/n, 39005 Santander, Spain*

Abstract

Norm-conserving pseudopotentials are used by a significant number of electronic-structure packages, but the practical differences among codes in the handling of the associated data hinder their interoperability and make it difficult to compare their results. At the same time, existing formats lack provenance data, which makes it difficult to track and document computational workflows. To address these problems, we first propose a file format (PSML) that maps the basic concepts of the norm-conserving pseudopotential domain in a flexible form and supports the inclusion of provenance information and other important metadata. Second, we provide a software library (*libPSML*) that can be used by electronic structure codes to transparently extract the information in the file and adapt it to their own data structures, or to create converters for other formats. Support for the new file format has been already implemented in several pseudopotential generator programs (including ATOM and ONCVPSP), and the library has been linked with SIESTA and ABINIT, allowing them to work with the same pseudopotential operator (with the same local part and fully non-local projectors) thus easing the comparison of their results for the structural and electronic properties, as shown for several example systems. This methodology can be easily transferred to any other package that uses norm-conserving pseudopotentials, and offers a proof-of-concept for a general approach to interoperability.

PACS: 71.15.Dx 71.10.-w 31.15.E-

Keywords: Pseudopotential, Density functional, Electronic Structure

*Corresponding author

Email addresses: albertog@icmab.es (Alberto García), Matthieu.Verstraete@ulg.ac.be (Matthieu Verstraete), yann.pouillon@unican.es (Yann Pouillon), javier.junquera@unican.es (Javier Junquera)

PROGRAM SUMMARY

Program Title: libPSML
Journal Reference:
Catalogue identifier:
Licensing provisions: BSD 3-clause
Program summary URL:
Programming language: Fortran
Distribution format: tar.gz
Keywords: Pseudopotential Density functional Electronic Structure
PACS: PACS: 71.15.Dx 71.10.-w 31.15.E-
Classification: 7.3
External routines/libraries: xmlf90 for XML handling in Fortran (http://launchpad.net/xmlf90)

Nature of problem:
Enhancing the interoperability of electronic-structure codes by sharing pseudopotential data.
Solution method:
Create an XML-based pseudopotential format (PSML), complete with a formal schema, and a processing library (*libPSML*) that transparently connects client codes to the information in the format.
References:

1. Introduction

Within computational science, reproducibility of research goes beyond using a specific version of a code and the appropriate input files. What is really sought is to replicate a certain physical result with a different code which implements the same basic equations of the domain at hand, but with a different set of approximations or details of implementation. This latter code will most likely have a different input data format, which might not be perfectly mapped to the format used by the original code. Reproducibility is still possible if the input data are curated so that their essential ontological properties are preserved, since it can be assumed that properly implemented codes within a domain will share a basic ontology for this domain, and can properly interpret its elements. Here we are concerned with electronic structure methods in the broad sense [1, 2], and in particular density functional theory (DFT [3, 4]), which have become standard theoretical tools to analyze and explain experiments in chemistry, spectroscopy, solid state physics, material science, biology, and geology, among many other fields.

Programs which implement DFT fall into two main categories, those which treat all electrons explicitly (linear augmented plane wave, LAPW [5, 6], linear muffin tin orbitals, LMTO [5, 7], or the Korringa-Kohn-Rostoker and related multiple-scattering theory methods, KKR [8, 9]), and those which replace the effect of the chemically inert core electrons by a (usually non-local) pseudopotential [10, 11, 12].

After some historical initial attempts, seminal work by Hamann, Schlüter and Chiang [12] and Kerker [13] showed how *norm-conserving pseudopotentials* achieve

a good tradeoff between *transferability* (a pseudopotential constructed for a specific environment, usually the isolated atom, gives good results when the atom is placed in a different environment) and *softness* (as measured for example by the number of plane waves that must be used in the representation of the wave functions for a good convergence of the physical properties). Since the 1990s more sophisticated schemes have been developed to treat the basic problem of eliding the core electrons. In particular, ultrasoft pseudopotentials [14], and the Projector Augmented Wave (PAW) method [15] are in widespread use for their improved accuracy and features, although they involve a significantly higher degree of complexity in the implementation and maintenance of the algorithms for electronic structure determination and analysis.

Several well-known atomic DFT programs [16, 17, 18, 19, 20, 21] generate pseudopotentials in a variety of formats, tailored to the needs of electronic-structure codes. While some generators are now able to output data in different bespoke formats, and some simulation codes are now able to read different pseudopotential formats, the common historical pattern in the design of those formats has been that a generator produced data for a single particular simulation code, most likely maintained by the same group. This implied that a number of implicit assumptions, shared by generator and user, have gone into the formats and fossilized there. Examples include, among others, many flavours of radial grids (from linear [20, 22] to logarithmic [19, 23], including all kinds of powers [24] or geometrical series [16, 25]), different ways of storing radial functions with spherical symmetry (angularly integrated with $4\pi r^2$ factors included, where r stands for the distance to the nuclei, or multiplication by different powers of r considered), different normalization conditions, etc. This leads to practical problems, not only of programming, but of interoperability and reproducibility, which depend on spelling out quite a number of details which are not well represented for all codes in existing formats.

Moreover, pseudopotential information can be produced and used at various levels. The original work was based on a set of semi-local operators (non-local in the angular part and local in the radial part), whereas very soon a computationally more efficient form based on fully non-local projectors plus a local part [26] was developed, and is now the standard norm-conserving form used by electronic-structure codes. The transformation of a semi-local pseudopotential into a fully non-local operator is not univocally defined from the semi-local components alone. Therefore, different codes can yield different results even if reading the same semi-local information from an input file. For example, most plane waves codes make the so-called local part equal to one of the semilocal components for reasons of efficiency, whereas SIESTA optimizes the local part for smoothness, since it is the only pseudopotential part that needs to be represented in a real space grid [27]. Modern pseudopotential generators are able to produce directly a set of non-local operators plus a local part. True interoperability can be achieved only if the codes use the same final projector-based pseudopotential.

Straightfoward interoperability of codes would allow all to benefit from the best capabilities of each. On the one hand, removing the variability associated with

the pseudopotential would enable a direct test of the quality of the basis sets used in the expansion of the one-particle Kohn-Sham eigenstates by different codes (with identical pseudopotentials, and at convergency, one should get the same total energy for the same atomic structure with any code). On the other hand, we can imagine a situation where the most stable atomic configuration of a large system can be found with a given code in a cheap yet accurate way, and then passed to another for further analysis through single-shot expensive calculations for the fixed geometry.

The obvious benefits of interoperability have naturally spawned efforts at providing more appropriate data mechanisms. At the most basic level, some pseudopotential generators (e.g. ONCVPSP or APE) offer options to write different output files suitable for different electronic-structure codes. While this addresses some of the problems, it falls short of a fully satisfactory solution. More robust are efforts to provide a standard format that can be used universally. Within the PAW domain, the PAW-XML format [28] comes close to this goal, being produced by a number of PAW dataset generators and read by most PAW-enabled electronic-structure codes. The UPF (Unified Pseudopotential Format) [29] is meant to encompass the full range of pseudopotential options, including (semi)local and fully-non-local norm-conserving, ultrasoft pseudopotentials, and PAW datasets. It is used within the Quantum Espresso suite of codes and converters exist for other codes.

We believe that, indeed, the solution to the interoperability problems involves the design a data format that faithfully maps the relevant concepts of the domain's ontology at all the relevant levels (semi-local pseudopotentials, charge densities, non-local projectors, local potentials, etc). But the format must also provide appropriate metadata that represents provenance (generation and any further processing) and documents in a parseable form any details that might be needed downstream. This second aspect has not been properly addressed so far.

The need for standardisation of the pseudopotential format and the provision of richer metadata to track provenance and document computational workflows has been made more relevant by the appearance during the last few years of high-throughput simulation schemes for materials design [30, 31, 32], which need well-tested and efficient pseudopotential libraries to draw upon. Examples of the latter are the ultrasoft pseudopotential library by Garrity, Bennett, Rabe and Vanderbilt [33, 34], the PSLibrary associated with QE [35], the Jollet-Torrent-Holzwarth [36] PAW library, and the libraries being built [37, 38, 39] using multi-projector norm-conserving pseudopotentials generated by the ONCVPSP code [22]. The latter are proving competitive with ultrasoft pseudopotentials and the PAW method in accuracy. [38, 39]

Motivated by the all the previous considerations, in this paper we present a file format for pseudopotential data (PSML, for PSeudopotential Markup Language) which is designed to encapsulate as much as possible the abstract concepts in the domain's ontology, and to provide appropriate metadata and provenance information. Moreover, we provide a software library (*libPSML*) that can be used by electronic structure codes to transparently extract the information in a

PSML file and adapt it to their own data structures, or to create converters for other formats. Our initial focus is the sub-domain in which norm-conserving pseudopotentials are used, which is not restricted to legacy cases but is set to grow in importance due to the new multi-projector pseudopotentials mentioned above. As the format is based on XML (eXtensible Markup Language) [40], it is very flexible and can serve as a basis for the future accomodation of PAW datasets and ultrasoft pseudopotentials. Our work falls within the scope of CECAM's Electronic Structure Library (ESL) initiative [41], which aims at building a collection of software functionalities, including standards and interfaces, to facilitate the development of electronic structure codes.
This paper is organized as follows: Section 2 describes the basic elements and structure of a PSML file, with a more formal XML schema given in Sec. 3. The *libPSML* library is described in Sec. 4. Section 5 discusses the relationship of the current work to previous efforts, and provides a guide to the tools already available in the PSML ecosystem and the mechanisms available for development of new ones. Finally, as an example of the interoperability benefits afforded by PSML, we present in Sec. 6 a comparison of actual physical results obtained by ABINIT and SIESTA using the same PSML files.

2. The PSML file format

This section contains a design rationale and human-readable description of the format. A formal schema specification can be found in Sec. 3. The following documents version 1.1 of the PSML, current as of this writing. Any updates and complementary information are available at the ESL PSML site: *http://esl.cecam.org/PSML*.

2.1. Root element

The root element is <psml>, containing a version attribute for use by parsers, and two attributes to make explicit the (mandatory for now) units used throughout the file: energy-unit (hartree) and length-unit (bohr). In addition, the root element should contain a uuid attribute to hold a *universally unique identifier* [42].

2.2. Provenance element(s)

The file should contain metadata concerning its own origin, to aid reproducibility. As a minimum, it should contain information about the program(s) used to generate or transform the pseudopotential, ideally with version numbers and compilation options, and provide a copy of any input files fed into the program(s). The information is contained in <provenance> elements (an example is provided in Table 1). Its internal structure is subject to a minimal specification:

- The attribute creator, where the name of the generator code is specified, is mandatory.

- The attribute date is mandatory.
- If input files are provided, they should be included as <input-file> elements with the single attribute name and no children other than character data, which should be placed in a CDATA section to avoid processing of XML reserved characters. For obvious reasons, the name of the file and its detailed content will depend on the program used to generate the pseudopotential. The name should be a mnemonic reference.
- The inclusion of an <annotation> element, providing arbitrary extra information in the form of key-value pairs, is encouraged. See the description and motivation of the <annotation> element in Sec. 2.10.

There can be an arbitrary number of <provenance> elements, ordered in temporal sequence in the file, with the most recent first. Since some XML processors might not preserve the order of the elements, it is suggested that a record-number attribute be added to make the temporal ordering explicit, with "1" for the oldest operation, "2" for the second oldest, and so on.
This feature supports the documentation of successive actions taken on the file's information. For example, a pseudopotential-generation program might generate a PSML file containing only semilocal potentials. This file is then processed by another program that generates a local potential and the corresponding nonlocal projectors. The PSML file produced by the second program should keep the original provenance element, and add another one detailing the extra operations taken.

2.3. Pseudo-atom specification

The <pseudo-atom-spec> element contains basic information about the chemical element, the generation configuration, and the type of calculation.
Element identification is in principle straightforward, with either the atomic number or the chemical symbol. However, in the interest of generality, and to cover special cases, such as synthetic (alchemical) atoms, the chemical symbol can be an arbitrary label, given by the attribute atomic-label (with the convention that when possible the first two characters give the standard chemical symbol) and the atomic number a real number instead of a simple integer, given by the attribute atomic-number.
The atomic calculation can be done non-relativistically, with scalar-relativistic corrections (i.e., with the mass and Darwin terms) [43, 44], or with the full Dirac equation, including spin-orbit effects. In the latter case, the (semilocal) pseudopotentials are typically provided in two sets: the degeneracy-averaged $j = l + 1/2$ and $j = l - 1/2$ components, appropriate for scalar-relativistic use, and the spin-orbit components [45].
It is also possible to carry out a non-relativistic calculation for a spin-polarized reference configuration (using spin-DFT). Standard practice is then to keep only the population-averaged pseudopotentials, as a closed-shell frozen core should not be represented by a spin-dependent potential. (But see Ref. [46] for a different point of view.)

Table 1: An example of the <provenance> element

```
<?xml version="1.0" encoding="UTF-8" ?>
<psml version="1.1" energy_unit="hartree" length_unit="bohr"
 uuid="904272c0-e496-11e6-496d-7f30ff5a9a4e"
 xmlns="http://esl.cecam.org/PSML/ns/1.1">

<provenance creator="ATM4.1.3" date="23-JUN-17">
<annotation action="semilocal-potential-generation">
<input-file name="INP">
<![CDATA[#
#  PS generation with core corrections
#  GGA (Perdew-Burke-Ernzerhof) XC , relativistic
#  3d7 4s1 configuration
#
   pe   Fe, GGA, rcore=0.70
        tm2      3.0
 n=Fe c=pbr
       0.0       0.0       0.0       0.0       0.0       0.0
    5    4
    4    0      1.00      0.00    # 4s1
    4    1      0.00      0.00    # 4p0
    3    2      7.00      0.00    # 3d7
    4    3      0.00      0.00    # 4f0
      2.00      2.00      2.00      2.00      0.00      0.70
#                                                       |
#                                          Radius of pseudocore
]]>
</input-file>
</provenance>
```

In practice, then, one might have a primary set of l-dependent $V_l(r)$ potentials, possibly the result of averaging, and, in the case of fully-relativistic calculations, a spin-orbit set. In other cases, the generation program might output the original lj versions in the Dirac case, or the "up" and "down" components for a spin-DFT calculation.

We cover all these possibilities in PSML by using the attribute relativity, with possible values "no", "scalar" or "dirac", and the optional attribute spin-dft to indicate (with value "yes") a non-relativistic spin-DFT calculation. Further, as detailed below, we provide support for the possible presence of various *sets* of magnitudes.

The pseudopotential construction is fundamentally dependent on the core-valence split. In most cases it is clear which states are to be considered as "valence", and which ones are to be kept in the frozen core. However, there are borderline cases in which one has the option to treat as "valence" so-called "semicore" states which are relatively shallow and/or exhibit a sizable overlap with proper valence states.

This information is provided in the mandatory <valence-configuration> element, which details the valence configuration used at the time of pseudopotential generation, given by the n and l quantum numbers, and the electronic occupation

of each shell. Empty shells can be omitted. In spin-DFT calculations, the spin-up and spin-down occupations are also given. This element has a mandatory attribute total-valence-charge which contains the total integrated valence charge Q_{val} for the configuration used to generate the pseudopotential.
The core configuration can be determined easily from the knowledge of the valence shells, but for completeness it can be given in the optional <core-configuration> element, with the same structure. This tag would be useful, for example, if a core-hole pseudopotential has been created [47].
The difference between the number of protons in the nucleus and the sum of the populations of the core shells is the effective atomic number of the pseudo-atom Z_{pseudo}, which must be given in the mandatory attribute z-pseudo.
The "pseudization flavor" or, more properly, a succint identifier for the procedure for pseudopotential generation, is encoded in the optional flavor attribute. If not present, more specific values can be given per pseudopotential block and per pseudization channel (see below).
If non-linear core-corrections [48] are present, the optional core-corrections attribute must be set to "yes".
One further piece of information is needed to complete the general specification of the computational framework: the type of exchange-correlation (XC) functional(s) used. With the explosive growth in the number of functionals, it is imperative that a robust naming convention be used. In the absence of a general registry of universally-agreed names, we propose a dual naming scheme. The element <exchange-correlation> contains:

- A mandatory element <libxc-info> that maps whatever XC functional is used in the generation code to the standard set of functionals in the LIBXC library [49, 50]. This library supports a large number of functionals, and new ones are added promptly as their details are published. We thus require that pseudopotential-generation programs producing PSML files, and programs using the PSML format, provide internal tables mapping any built-in XC naming schemes to the LIBXC one. This element has the attribute number-of-functionals, and as many <functional> elements as indicated by this attribute, with attributes name, and id, as shown in the example of Table 2. These correspond to the LIBXC identification standard. The attribute type (with values "exchange", "correlation", or "exchange-correlation") is optional. Further, to support arbitrary mixtures of functionals, the optional attribute weight can also be indicated.

- An (optional) element <annotation> (see Sec. 2.10) that can contain any XC identification used by the creator of the file, in the form of attribute-value pairs. This information can be read in an ad-hoc fashion by client programs, but it is obviously not as complete or robust as that contained in the <libxc-info>. For maximum interoperability, client programs should thus implement an interface to LIBXC.

An optional <annotation> element can also be included inside the <pseudo-atom-spec> element. Note that the ordering of child elements is significant (see

Table 2: An example of the <pseudo-atom-spec> element

```
<pseudo-atom-spec atomic-label="Se" z-pseudo="6" atomic-number="34"
 flavor="Troullier-Martins" relativity="dirac" spin-dft="no"
 core-corrections="yes">
<exchange-correlation>
<annotation oncvpsp-xc-code="3"
 oncvpsp-xc-type="LDA -- Ceperley-Alder Perdew-Zunger" />
<libxc-info number-of-functionals="2">
<functional name="Slater exchange (LDA)" type="exchange" id="1" />
<functional name="Perdew &amp;amp; Zunger (LDA)" type="correlation" id="9" />
</libxc-info>
</exchange-correlation>
<valence-configuration total-valence-charge="6">
<shell n="4" l="s" occupation="2" />
<shell n="4" l="p" occupation="4" />
</valence-configuration>
</pseudo-atom-spec>
```

Sec. 3).

2.4. Radial functions and grid specification

At the core of this new format we face the problem of how to store a variety of different radial functions (semilocal pseudopotentials, projectors, pseudo wave-functions, pseudocore and valence charges, etc.) in a radial grid. Most pseudopotential-generation codes export their data for their radial functions $f(r)$ as a tabulation $\{f(r_i)\}$, where $\{r_i\}$ are discrete values of the radial coordinate in an appropriate mesh.

A variety of meshes with different functional forms and parametrization details are in common use. Our preferred way to handle this variety of choices is to specify the actual grid point data in the file. This is most extensible to any kind of grid, and avoids problems of interpretation of the parameters, starting and ending points, etc. Furthermore, as explained below, the PSML handling library is completely grid-agnostic, since evaluators are provided for the relevant functions $f(r)$, in such a way that a client code can obtain the value of f at any radial coordinate r, in particular at the points of a grid of its own choosing. The precision of the computed value $f(r)$ is of course dependent on the quality of the $f(r_i)$ tabulation in the first place, and producer codes should take this issue seriously. We discuss more points related to the evaluation of tabulated data in Sec. 4.6.

The format should be flexible enough to allow each radial function to use its own grid if needed, while providing for the most common case in which all radial functions use a common grid (or a subset of its points). Our solution is to encode the information about each radial function in a <radfunc> element, which contains the tabulation data $f(r_i)$ in a <data> element. The grid specification uses a cascade scheme with an (optional but recommended) top-level <grid> element, optional mid-level <grid> elements under certain grouping elements, and at the lowest level optional <grid> elements inside the individual <radfunc>

elements. The grid $\{r_i\}$ for a function is inherited from the closest <grid> element at an enclosing level if it is not specified in the local <radfunc> element. The grouping elements currently allowed to include mid-level grids are those for semilocal potentials, nonlocal projectors, local potential, pseudo-wavefunctions, valence charge, and pseudocore charge.
The <grid> elements should have a mandatory "npts" attribute providing the number of points, and a <grid-data> element with the grid point data as formatted real numbers with appropriate precision. All radial data must be given in bohr.
For convenience, it is allowed to include an <annotation> element as child of <grid>, with appropriate attributes, to provide additional information regarding the form of the grid data. The client program can process this information if needed. The <data> element may contain an optional attribute npts to indicate the number of values that follow. In its absence, the number of values must match the size of the grid. The npts attribute is useful in those cases in which the effective range of a radial function is significatively smaller than the extent of the grid. For example, a PSML file might contain a top-level grid with a large range, appropriate for the valence charge density, of which only a subset of points are used for the projectors, which have a much smaller range.
A technical point should be kept in mind. When processing a PSML file, the radial function information is typically stored internally as a table on which interpolation is performed to obtain values of the function at specific radii. In order to avoid the dangers associated with extrapolation, the radial grid must contain as first point $r = 0$, and any radial magnitudes (pseudopotentials, wavefunctions, pseudo-core or valence charges) should be given without extra factors of r that might hamper the calculation of needed values at $r = 0$. In this way the processor can unambiguously determine the function values at all radial points.
When evaluating a function at a point r beyond the maximum range of the tabulated data in the PSML file, a processor should return:

- $-Z_{\mathrm{pseudo}}/r$ for the semi-local and local pseudopotentials, in keeping with the well-known asymptotic behavior.
- Zero for projectors, pseudo-wave-functions and valence and pseudo-core charges.

2.5. Semilocal components of the pseudopotentials

When available, the l-dependent (or maybe lj-dependent) semilocal components $V_l(r)$ of the pseudopotential are classified under <semilocal-potentials> elements, with attributes:

- set: A string indicating which *set* (see below) the potentials belong to. If missing, the information is obtained from the records for the individual potentials.

Table 3: An example of the <grid> element

```
<grid npts="1186">
<annotation type="log-atom" nrval="1186" scale="0.442634317262E-04"
 step="0.125000000000E-01" />
<grid-data>
  0.000000000000E+00  5.567654309900E-07  1.120534108973E-06
1.691394123953E-06
  2.269434673967E-06  2.854746079028E-06  3.447419795234E-06
4.047548429058E-06
  ...
</grid-data>
</grid>
```

- flavor: (optional) The pseudization flavor. If missing, its value is inherited from the value in the <pseudo-atom-spec> element. It can also be superseded by the records for the individual potentials.

The set attribute allows the handling of various sets of pseudopotentials. Its value is normalized as follows, depending on the type of calculation generating the pseudopotential and the way in which the code chooses to present the results:

- "non_relativistic" for the non-relativistic, non-spin-DFT case.
- "scalar_relativistic" if the calculation is scalar-relativistic, or if it is fully relativistic and an set of lj potentials averaged over j is provided.
- "spin_orbit" if a fully relativistic code provides this combination of lj potentials.
- "lj" for a fully relativistic calculation with straight output of the lj channels.
- "spin_average" for the spin-DFT case when the generation code outputs a population-averaged pseudopotential.
- "up" and "down", for a spin-DFT calculation with straight output of the spin channels.
- "spin_difference" for the spin-DFT case when the generation code outputs also the difference between the "up" and "down" potentials. This and the previous case are retained for historical reasons, but are likely used rarely.

Note that a given code might choose to output its semilocal-potential information in two different forms (say, as scalar-relativistic and spin-orbit combinations plus the lj form). The format allows this, although in this particular case the information can easily be converted from the lj form to the other by client programs.
For extensibility, the format allows two more values for the set attribute, "user_extension1" and "user_extension2", which can in principle be used to store custom information while maintaining structural and operative compatibility with the format.

The pseudopotentials must be given in hartree.
The <semilocal-potentials> element contains child <slps> elements, which store the information for the individual semilocal pseudopotential components. The attributes of this element are:

- n: principal quantum number of the pseudized shell.
- l: angular momentum number of the pseudized shell.
- j: (compulsory for "lj" sets) j quantum number
- rc: r_c pseudization radius for this shell (in Bohr).
- eref: (optional) reference energy (eigenvalue) of the all-electron wavefunction to be pseudized (in hartree).
- flavor: (optional) To allow for different schemes for different channels, the value of this attribute, when present, takes precedence over the flavor attributes in the <pseudo-atom-spec> element and the <semilocal-potentials> elements.

The optional attribute eref might only be meaningful for certain potentials (for example, those that have been directly generated, and are not the product of any extra conversion, such as from lj to scalar-relativistic plus spin-orbit form).
Each <slps> element contains a <radfunc> element. The order in which the <slps> elements appear is irrelevant.
The <semilocal-potentials> element can contain an optional <grid> child applying to all the enclosed elements, as well as an optional <annotation> element for arbitrary extra information in key-value format.

2.6. Pseudopotential in fully non-local form.
Most modern electronic-structure codes do not actually use the pseudopotential in its semi-local form, but in a more efficient fully non-local form based on short-range projectors plus a "local" potential:

$$\hat{V}_{ps} = \hat{V}_{\rm local} + \sum_i |\chi_i > E^i_{\rm KB} < \chi_i| \qquad (1)$$

proposed originally by Kleinman and Bylander [26] and generalized among others by Blöchl [15], Vanderbilt [14], and Hamann [22].
The information about this operator form of the pseudopotential is split in two elements, holding the local potential and the nonlocal projectors.

2.6.1. Local potential
The <local-potential> element has the attribute

- type We suggest the string "l=X" when the local potential is taken to be the semi-local component for channel "X", or any other succint comment if not.

and contains a <radfunc> element with the actual data for the local pseudopotential.
Optionally, the <local-potential> element can contain a child <local-charge> element, describing a radial function $\rho_{\mathrm{local}}(r)$ related to $V_{\mathrm{local}}(r)$ by Poisson's equation. That is, $\rho_{\mathrm{local}}(r)$, integrating to Z_{pseudo} and localized in the core region, is the effective charge that would generate $V_{\mathrm{local}}(r)$. The <local-charge> element is optional because not all $V_{\mathrm{local}}(r)$ functions are representable as originating from a charge distribution ($V'_{\mathrm{local}}(0)$ must be zero for this). When it is present, however, it can save some client programs (such as SIESTA, which uses $\rho_{\mathrm{local}}(r)$ to generate a very convenient localized neutral-atom potential) the task of computing numerical derivatives of $V_{\mathrm{local}}(r)$.
The <local-potential> element can also contain an optional <grid> child applying to all the enclosed elements, as well as an optional <annotation> element for arbitrary extra information in key-value format.

2.6.2. Non-local projectors
The information about the non-local projectors is stored in <nonlocal-projectors> elements, with the optional attribute set, as above, containing <proj> elements with attributes

- ekb: Prefactor of the projector in the corresponding term in Eq. 1 (in hartree).
- eref: (optional) Reference energy used in the generation of the projector (in hartree).
- l: angular momentum number
- j: (compulsory for "lj" sets) j quantum number
- seq: sequence number within a given l (or lj) shell, to support the case of multiple projectors.
- type: Succint comment about the kind of projector

and a <radfunc> element containing the data for the χ_i functions in Eq. 1. These functions are formally three-dimensional, including the appropriate spherical harmonic for the angular coordinates, and a radial component: $\chi_i = \chi_i(r)Y_{lm}(\theta,\phi)$. What is actually stored in the file is the function $\chi_i(r)$, normalized in the one-dimensional sense:

$$\int_0^\infty r^2 \left|\chi_i(r)\right|^2 dr = \int_0^\infty \left|r\chi_i(r)\right|^2 = 1, \tag{2}$$

and proportional to r^l near the origin.
There can be several <nonlocal-projectors> elements, using different values for the set attribute, as explained in Sec. 2.5. For projectors in the "dirac" case, the functionality provided by the handling of set attributes can be very useful, as it

is not straightforward to convert the lj information into scalar-relativistic and spin-orbit combinations. Unlike in the semi-local case, this conversion is not reversible, so the lj form is more fundamental. For maximum interoperability, producer codes should store both the lj and the combination sets.
The optional attribute eref might only be meaningful for certain projectors (for example, those that have been directly generated, and are not the product of any extra conversion, such as from lj to scalar-relativistic plus spin-orbit form).

2.7. Pseudo-wave functions

Pseudo-wavefunctions are typically produced at an intermediate stage in the generation (and testing) of a pseudopotential, but they are not strictly needed in electronic-structure codes, except in a few cases:

- When atomic-like initial wavefunctions are needed to start the electronic-structure calculation.
- When the code uses internally a fully-nonlocal form of the pseudopotential which is constructed from the semilocal form and the pseudo-wavefunctions.

While these pseudo-wavefunctions could be generated by the client program, we allow for the possibility of including them explicitly in the PSML file. Any (optional) wavefunction data must be included in <pseudo-wave-functions> elements, with a set attribute and as much extra metadata as needed (which we do not try to standardize at this point, so it should be given in the form of annotations).
The extra metadata might indicate whether the data is for actual pseudized wave-functions, or for the pseudo-valence wave functions generated with the obtained pseudopotential. There might be subtle differences between them, notably regarding relativistic effects, as some generation codes use a non-relativistic scheme to "test" the pseudopotential and generate pseudo wavefunctions, instead of a scalar or fully relativistic version.
Each pswf is given in a <pswf> element, with attributes that identify the quantum numbers for the shell and the apropriate energy level (which could be the eigenvalue in the original pseudization, or another energy level used in the integration leading to the wavefunction):

- n: principal quantum number of the shell.
- l: angular momentum number of the shell.
- j: (compulsory for "lj" sets) j quantum number
- energy_level

and a <radfunc> element.

The data is for the standard radial part of the wave function $R_{n,l}(r)$, and (for bound states) should be normalized as

$$\int_0^\infty r^2 \left|R_{n,l}(r)\right|^2 dr = \int_0^\infty \left|u_{n,l}(r)\right|^2 = 1. \tag{3}$$

Within our PSML format $R(r)$ is given, rather than $u(r)$, due to the extrapolation issues detailed in the section covering the grid.
The <pseudo-wave-functions> elements can contain an optional <grid> child applying to all the enclosed elements.

2.8. Valence charge density

Some electronic-structure codes might need information about the valence charge density of the pseudo-atom. This can be the pseudo-valence charge density used to unscreen the ionic potential during the pseudopotential generation process, or the pseudo-valence charge computed from the pseudopotential itself. In PSML it is given under the <valence-charge> element, whose child <radfunc> element holds a solid-angle-integrated form $q(r)$ normalized so that:

$$\int_0^\infty r^2 q(r) dr = Q. \tag{4}$$

Here Q is the total charge output, which must be stored in the total-charge attribute. The attributes is-unscreening-charge (with value "yes" or "no") and rescaled-to-z-pseudo ("yes" or "no") are optional.
In combination with the information in the <valence-configuration> element, these attributes will help some client codes process the valence charge density data appropriately, particularly in the case in which the pseudopotential generation used an ionic configuration. Some codes output in this case the unscreening charge rescaled to Z_{pseudo}. The <valence-charge> can also contain an optional annotation element.

2.9. Pseudocore-charge density

An (optional) smoothed charge density matching the density of the core electrons beyond a certain radius, for use with a non-linear core correction scheme [48], is contained within the <pseudocore-charge> element, with the data in the same solid-angle-integrated form (and implicit units) as the valence charge, and with the extra optional attributes:

- matching-radius: The point r_{core} at which the true core density is matched to the pseudo-core density.
- number-of-continuous-derivatives: In the original scheme by Louie *et al.* [48] the pseudo-core charge was represented by a two-parameter formula, providing continuity of the first derivative only. Other typical schemes provide continuity of the second and even higher derivatives. It is expected that the number of continuous derivatives, rather than the detailed form of the matching, is of more interest to a client program.

An optional annotation element with appropriate attributes might be given to document any extra details of the model-core generation. The <pseudocore-charge> element must appear if the core-corrections attribute of the <pseudo-atom-spec> element has the value "yes".

2.10. The handling of annotations

XML provides for built-in extensibility and client programs can use as much or as little information as needed. For the actual mapping of a domain ontology to a XML-based format, however, clients and producers have to agree on the terms used. What has been described in the above sections is a minimal form of such a mapping, containing the basic concepts and functions needed. The extension of the format with new fixed-meaning elements and attributes would involve an updated schema and re-coding of parsers and other programs. A more light-weight solution to the extensibility issue is provided by the use of *annotations*, which have the morphology of XML empty elements (containing only attributes) but can appear in various places and contain arbitrary key-value pairs. Annotations provide immediate information to human readers of the PSML files, and can be exploited informally by client programs to extract additional information. For the latter use, it is clear that some degree of permanence and agreed meaning should be given to annotations, but this task falls not on some central authority, but on specific codes.
Annotations are currently allowed within the following elements: <provenance>, <pseudo-atom-spec>, <exchange-correlation>, <valence-configuration>, <core-configuration>, <semilocal-potentials>, <local-potential>, <nonlocal-projectors>, <pseudo-wave-functions>, <valence-charge>, <core-charge>, and <grid>. Top-level annotations are not allowed. They properly belong in the <provenance> elements.

3. Formal specification of the PSML format

We provide a formal XML schema for the PSML format, given in the very readable RELAX NG compact form [51]. This schema can be used directly for validation of PSML files, or converted to an W3C schema file (using respectively the `jing` and `trang` tools of the RELAX NG project).
This is the overall structure of a PSML document, showing the main building blocks. The definitions of the grammar elements are given in the next section, when the closely related API is discussed.

```
default namespace = "http://esl.cecam.org/PSML/ns/1.1"

PSML =  element psml {
            Root.Attributes
          , Provenance+              # One or more provenance elements
          , PseudoAtomSpec
          , Grid?                    # Optional top-level grid
          , ValenceCharge
          , CoreCharge?              # Optional pseudo-core charge
                , (
```

```
                    (SemiLocalPotentials+ , PSOperator?)
                    |
                    (SemiLocalPotentials* , PSOperator )
                  )
          , PseudoWaveFunctions*          # Zero or more Pseudo Wavefunction groups
        }

PseudoAtomSpec =  element pseudo-atom-spec {    PseudoAtomSpec.Attributes
                                              , Annotation?
                                              , ExchangeCorrelation
                                              , ValenceConfiguration
                                              , CoreConfiguration?
                                            }

PSOperator =   (   LocalPotential             # Local potential
                 , NonLocalProjectors* )      # Zero or more fully nonlocal groups
```

The quantifiers '*', '+', and '?' mean "zero or more", "one or more", and "at most one", and the '|' sign expresses an exclusive "or" (choice) operation. Even though RELAX NG can accommodate interleaved elements, this feature is not fully representable in W3C schema, and the ordering of the elements above is strict. We use a URI in the "esl.cecam.org" domain to identify the schema namespace, but note that it is not a resolvable location.
The main feature not discussed above in Sec. 2 is the constraint of *non-emptiness* of the PSML file: it must contain at least either a set of semilocal-potentials, or a complete pseudopotential operator consisting of a local potential and a set of nonlocal projectors. It is also possible to have a single local potential as a degenerate form of pseudopotential operator. Beyond the minimal requirements, a PSML file can contain multiple occurrences of any of these elements.

4. The PSML library

We provide a companion library to the PSML format, *libPSML*, that provides transparent parsing of and data extraction from PSML files, as well as basic editing and data conversion capabilities.
The library is built around a data structure of type *ps_t* that maps the information in a PSML file. Instances of this structure are populated by PSML parsers, processed by intermediate utility programs, and used as handles for information retrieval by client codes through accessor routines. The library provides, in essence:

(i) A routine to parse a PSML file and produce a *ps* object of type *ps_t*.
(ii) A routine to dump the information in a *ps* object to a PSML file.
(iii) Accessor routines to extract information from *ps* objects.
(iv) Some setter routines to insert specific blocks of information into *ps* objects. These might be used by intermediate processors or by high-level parsers.

The library is written in modern Fortran and provides a high-level Fortran interface. A C/C++ interface is in preparation.
An example of use of the library is provided in Table 4.

In what follows we describe the basic exported data structures and procedures. Full documentation for the library, as well as general information about the PSML format ecosystem, is available at the PSML reference page under the Electronic Structure Library project website: *http://esl.cecam.org/PSML*.

Table 4: An example of the idioms used in the *libPSML* library

```
use m_psml
type(ps_t)   :: ps

call psml_reader(filename,ps)

! Set up our grid
npts = 400; delta = 0.01
allocate(r(npts))
do ir = 1, npts
   r(ir) = (ir-1)*delta
enddo

call ps_Potential_Filter(ps,set=SET_SREL,indexes=idx,number=npots)
do i = 1, npots
   call ps_Potential_Get(ps,idx(i),l=l,n=n,rc=rc)
   ...
   do ir = 1, npts
      val = ps_Potential_Value(ps,idx(i),r(ir))
      ...
   enddo
enddo
```

4.1. Exported types

The library exports a few fortran derived types to represent opaque handles in the routines:

- ps_t

  This is the type for the handle which should be passed to most routines in the API

- ps_annotation_t

  The associated handle is used in the routines that create annotations or extract the data in them (see Section 4.9).

- ps_radfunc_t

  This corresponds to the internal implementation of a radial function. Its use is mostly reserved for low-level operations, as the API provides convenience evaluators for most functions.

Additionally, and to avoid ambiguities in real types, the library exports the integer parameter ps_real_kind that represents the *kind* of the real numbers accepted and returned by the library.

4.2. Parsing

- psml_reader (*filename, ps, debug*)

  parses the PSML file *filename* and populates the data structures in the handle *ps*. An optional *debug* argument determines whether the library issues debugging messages while parsing.

- ps_destroy (*ps*)

  is a low-level routine provided for completeness in cases where a pristine *ps* is needed for further use.

4.3. Library identification

- *function* ps_GetLibPSMLVersion() *result*(*version*)

  The version is returned as an integer with the two least significant digits associated to the patch level (for example: 1106 would correspond to the typical dot form 1.1.6).

4.4. Data accessors

The API follows closely the element structure of the PSML format. Each section in the high-level document structure of Sec. 3 is mapped to a group of routines in the API. Within each, there are routines to query any internal structure (attributes, existence, number, or selection of child elements) and routines to obtain specific data items (attributes, content of child elements).

4.4.1. Root attributes

```
Root.Attributes =  attribute energy_unit { "hartree" }
                 , attribute length_unit { "bohr" }
                 , attribute uuid { xsd:NMTOKEN }
                 , attribute version { xsd:decimal }
```

- ps_RootAttributes_Get (*ps,uuid,version,namespace*)

  As in all the routines that follow, the handle *ps* is mandatory. All other arguments are optional, with "out" intent, and of type character(len=*). *version* returns the PSML version of the file being processed. A given version of the library is able to process files with lower version numbers, up to a reasonable limit. For our purposes, the NMTOKEN specification refers to a string without spaces or commas.

4.4.2. Provenance data

```
Provenance =  element provenance {
                attribute record-number { xsd:positiveInteger }?
              , attribute creator { xsd:string }
              , attribute date { xsd:string }

              , Annotation?
              , InputFile*    # zero or more input files
              }

InputFile =  element input-file {
               attribute name { xsd:NMTOKEN }, # No spaces or commas allowed
               text
             }
```

As there can be several <provenance> elements, the API provides a function to enquire about their number (*depth* of provenance information), and a routine to get the information from a given level:

- *function* ps_Provenance_Depth(*ps*) *result*(*depth*)

  The argument *depth* returns an integer number.

- ps_Provenance_Get(*ps,level,creator,date,annotation,number_of_input_files*)

  The integer argument *level* selects the provenance depth level (1 is the deepest, or older, so to get the latest record the routine should be called with *level*=*depth* as returned from the previous routine). All other arguments are optional with "out" intent. *creator* and *date* are strings. Here and in what follows, *annotation* arguments are of the opaque type ps_annotation_t (see Section 4.1). If there is no annotation, an empty structure is returned. The information in an annotation object can be accessed using routines described in Section 4.9.

4.4.3. Pseudo-atom specification attributes and annotation

```
PseudoAtomSpec.Attributes =
       attribute atomic-label { xsd:NMTOKEN },
       attribute atomic-number { xsd:double },
       attribute z-pseudo { xsd:double },
       attribute core-corrections { "yes" | "no" },
       attribute relativity { "no" | "scalar" | "dirac" },
       attribute spin-dft { "yes" | "no" }?,
       attribute flavor { xsd:string }?
```

- ps_PseudoAtomSpec_Get (*ps, atomic_symbol, atomic_label, atomic_number, z_pseudo, pseudo_flavor, relativity, spin_dft, core_corrections, annotation*)

  The arguments *spin_dft* and *core_corrections* are boolean, and the routine returns an empty string in *flavor* if the attribute is not present (recall that flavor is a cascading attribute that can be set at multiple levels). The arguments *atomic_number* and *z_pseudo* are reals of kind ps_real_kind (see Sec. 4.1).

4.4.4. Valence configuration

```
ValenceConfiguration =  element valence-configuration {
                          attribute total-valence-charge { xsd:double },
                          Annotation?,
                          ValenceShell+
                        }

ValenceShell =   Shell

Shell =  element shell {
           attribute_l,
           attribute_n,
           attribute occupation { xsd:double },
           attribute occupation-up { xsd:double }?,
           attribute occupation-down { xsd:double }?
         }

attribute_l = attribute l { "s" | "p" | "d" | "f" | "g" }
attribute_n = attribute n { "1" | "2" | "3" | "4" | "5" | "6" | "7" | "8" | "9" }
```

- ps_ValenceConfiguration_Get(*ps,nshells,charge,annotation*)

  This routine returns (as always, in optional arguments), the values of the top-level attributes, any annotation, and the number of `Shell` elements, which serves as upper limit for the index i in the following routine, which extracts shell information:

- ps_ValenceShell_Get(*ps,i,n,l,occupation,occ_up,occ_down*)

  The n and l quantum number arguments are integers (despite the use of spectroscopic symbols for the angular momentum in the format), and the occupations real.

4.4.5. Exchange and correlation

```
ExchangeCorrelation =  element exchange-correlation {
                         Annotation?
                         , element libxc-info {
                              attribute number-of-functionals { xsd:positiveInteger },
                              LibxcFunctional+
                           }
                        }

LibxcFunctional =   element functional {
                      attribute id { xsd:positiveInteger },
                      attribute name { xsd:string },
                      attribute weight { xsd:double }?,

                      # allow canonical names and libxc-style symbols

                      attribute type { "exchange" | "correlation" | "exchange-correlation" |
                                       "XC_EXCHANGE" | "XC_CORRELATION" |
                                          "XC_EXCHANGE_CORRELATION" }?
                    }
```

The routines follow the same structure as those in the previous section.

- ps_ExchangeCorrelation_Get(*ps,annotation,n_libxc_functionals*)

- ps_LibxcFunctional_Get(*ps,i,name,code,type,weight*)

  The argument *type* corresponds to the type of Libxc functional, and *code* to the id number.

4.4.6. Valence and Core Charges

```
ValenceCharge =  element valence-charge {
                   attribute total-charge { xsd:double },
                   attribute is-unscreening-charge { "yes" | "no" }?,
                   attribute rescaled-to-z-pseudo { "yes" | "no" }?,
                   Annotation?,
                   Radfunc
                 }

# =========
CoreCharge =  element pseudocore-charge {
                attribute matching-radius { xsd:double },
                attribute number-of-continuous-derivatives { xsd:nonNegativeInteger },
                Annotation?,
                Radfunc
              }
```

These are radial functions with some metadata in the form of attributes, an optional annotation, and a Radfunc child. The accessors have the extra optional argument *func* that returns a handle to a ps_radfunc_t object, which can later be used to get extra information.

- ps_ValenceCharge_Get(*ps,total_charge, is_unscreening_charge, rescaled_to_z_pseudo, annotation,func*)

  The routine returns an emtpy string in *is_unscreening_charge* and *rescaled_to_z_pseudo* if the attributes are not present in the PSML file.

- ps_CoreCharge_Get(*ps,rc,nderivs,annotation,func*)

  rc corresponds to the matching radius and *nderivs* to the continuity information. Negative values are returned if the corresponding attributes are not present in the file.

The *func* object can be used to evaluate the radial functions at a particular point *r*:

- function ps_GetValue(*func,r*) result(*val*)

but the API offers some convenience functions

- function ps_ValenceCharge_Value(*ps,r*) result(*val*)
- function ps_CoreCharge_Value(*ps,r*) result(*val*)

4.4.7. Local Potential and Local Charge Density

```
LocalPotential =  element local-potential {
                    attribute type { xsd:string },
                    Annotation?,
                    Grid?,
                    Radfunc,
                    LocalCharge?  # Optional local-charge element
                  }

LocalCharge =  element local-charge {
                 Radfunc
               }
```

- ps_LocalPotential_Get(*ps,type,annotation,func,has_local_charge,func_local_charge*)

In this version of the API, the optional <local-charge> element is not given a first-class status. To evaluate it (if the boolean argument *has_local_charge* is true), the *func_local_charge* argument has to be used in the ps_GetValue routine above. The local potential can be evaluated via the *func* object or with the convenience function

- function ps_LocalPotential_Value(*ps,r*) result(*val*)

4.4.8. Semilocal potentials

```
SemiLocalPotentials =  element semilocal-potentials {
                         attribute_set,
                         attribute flavor  { xsd:string }?,
                         Annotation?,
                         Grid?,
                         Potential+
                       }

Potential =   element slps {
                attribute flavor { xsd:string }?,
                attribute_l,
                attribute_j ?,
                attribute_n,
                attribute rc { xsd:double },
                attribute eref { xsd:double }?,
                Radfunc
              }
```

As explained in Sec. 2, there can be several <semilocal-potentials> elements corresponding to different sets. Internally, the data is built up in linked lists during the parsing stage and later all the data for the <slps> child elements are re-arranged into flat tables, which can be queried like a simple database. The table indexes for the potentials with specific quantum numbers, or set membership, can be obtained with the routine

- ps_SemilocalPotentials_Filter(*ps,indexes_in,l,j,n,set,indexes,number*)

  Here, the optional argument (of intent "in") *indexes_in* is an integer array containing a set of indexes on which to perform the filtering operation. If not present, the full table is used. The optional arguments *l,j,n,set* are the

values corresponding to the filtering criteria. Upon return, the optional argument *indexes* would contain the set of indexes which match all the specified criteria, and *number* the total number of matches.

The *set* argument has to be given using special integer symbols exported by the API: SET_SREL, SET_NONREL, SET_SO, SET_LJ, SET_UP, SET_DOWN, SET_SPINAVE, SET_SPINDIFF, or the wildcard specifier SET_ALL. An example of the use of this routine has already been given in Table 4.

The appropriate indexes can then be fed into the following routines to get specific information:

- ps_Potential_Get(*ps,i,l,j,n,rc,eref,set,flavor,annotation,func*)

  All arguments except *ps* and *i* are optional. The value returned in *set* is an integer which can be converted to a mnemonic string through the str_of_set convenience function. The annotation returned corresponds to the optional <annotation> element of the parent block of the <slps> element.

  The routine returns a very large positive value in *eref* if the corresponding attribute is not present in the file.

- function ps_Potential_Value(*ps,i,r*) result(*val*)

4.4.9. Nonlocal Projectors

```
NonLocalProjectors =  element nonlocal-projectors {
                        attribute_set,
                        Annotation?,
                        Grid?,
                        Projector+
                      }

Projector =   element proj {
                attribute ekb { xsd:double },
                attribute eref { xsd:double }?,
                attribute_l,
                attribute_j ?,
                attribute seq { xsd:positiveInteger },
                attribute type { xsd:string },
                Radfunc
              }+
```

The ideas are exactly the same as for the semilocal potentials. The relevant routines are:

- ps_NonlocalProjectors_Filter(*ps,indexes_in,l,j,seq,set,indexes,number*)

- ps_Projector_Get(*ps,i,l,j,seq,set,ekb,eref,type,annotation,func*)

  The routine returns a very large positive value in *eref* if the corresponding attribute is not present in the file.

- function ps_Projector_Value(*ps,i,r*) result(*val*)

4.4.10. Pseudo Wavefunctions

```
PseudoWaveFunctions =  element pseudo-wave-functions {
                         attribute_set,
                         Annotation?,
                         Grid?,
                         PseudoWf+
                       }

PseudoWf =  element pswf {
              attribute_l,
              attribute_j ?,
              attribute_n,
              attribute energy_level { xsd:double} ?,
              Radfunc
            }
```

Again, the same strategy:

- ps_PseudoWavefunctions_Filter(*ps,indexes_in,l,j,set,indexes,number*)
- ps_PseudoWf_Get(*ps,i,l,j,n,set,energy_level,annotation,func*)

  The routine returns a very large positive value in *energy_level* if the corresponding attribute is not present in the file.
- function ps_PseudoWf_Value(*ps,i,r*) result(*val*)

4.5. Radial function and grid information

```
Radfunc =  element radfunc {
             Grid?,               # Optional grid element
             element data {
               list { xsd:double+ }       # One or more floating point numbers
             }
           }

Grid =  element grid {
          attribute npts { xsd:positiveInteger },
          Annotation?,
          element grid-data {
                 list { xsd:double+ }   # One or more floating point numbers
          }
        }
```

In keeping with the PSML philosophy of being grid-agnostic, the basic API tries to discourage the direct access to the data used in the tabulation of the radial functions. The values of the functions at a particular point r can be generally obtained through the ps_(Name)_Value interfaces, or through the ps_GetValue interface using *func* objects of type ps_radfunc_t.
It is nevertheless possible to get annotation data for the grid of a particular radial function, or for the top-level grid, through the function

- function ps_GridAnnotation(*ps,func*) result(*annotation*)

  If a radial function handle *func* is given, the annotation for that radial function's grid is returned. Otherwise, the return value is the annotation for the top-level grid.

4.6. The evaluation engine

In the current version of the library the evaluation of tabulated functions is performed by default with polynomial interpolation, using a slightly modified version of an algorithm borrowed (with permission) from the ONCVPSP program by D.R. Hamann [20]. By default seventh-order interpolation, as in ONCVPSP, is used. If the library is compiled with the appropriate pre-processor symbols, the interpolator and/or its order can be chosen at runtime, but we note that this should be considered a debugging feature, as the reproducibility of results would be hampered if client codes change the interpolation parameters at will. Generator codes should instead strive to produce data tabulations that will guarantee a given level of precision when interpolated with the default scheme, using appropriate output grids on which to sample their internal data sets. For example, our own work on enabling PSML output in ONCVPSP (see below) includes diagnostic tools to check the interpolation accuracy.

Most codes use internally a non-uniform grid (e.g. logarithmic). We have found that a good choice of output grid is a subset of the producer's working grid points that leaves out most of the very close points near the origin but maintains the rest. This can be achieved by imposing a minimum inter-point separation δ. This parameter δ can be smaller than the typical linear-grid step used currently by most codes, and still lead to smaller grids (in terms of number of points) that preserve the accuracy of the output.

High-order interpolation can lead to *ringing* effects (oscillations of the interpolating polynomial between points), notably near edge regions when the shape of the function changes abruptly. This is the case, for example, if the function drops to zero within the interpolation range as a result of cutting off a tail. The actual interpolated values will typically be very small, but might cause undesirable effects in the client code. To avoid this problem, the *libPSML* evaluator works internally with an effective end-of-range that is determined by analyzing the data values after parsing.

If needed for debugging purposes, the evaluator engine can be configured by the routine:

- ps_SetEvaluatorOptions(*quality_level,debug, use_effective_range, custom_interpolator*)

  All arguments are optional, and apply globally to the operation of the library. The *custom_interpolator* argument is not allowed if the underlying Fortran compiler does not support procedure pointers. *quality_level* (an integer) is by default and will typically be the interpolation order, but its meaning can change with the interpolator in use. The evaluator uses an effective range by default, as discussed above, but this feature can be turned off by setting *use_effective_range* to `.false.`. The *debug* argument will turn on any extra printing configured in the evaluator. By default, no extra printing is produced.

Finally, in case it is necessary to look at the raw tabular data for debugging purposes, the library also provides a low-level routine:

- ps_GetRawData(*func,rg,data*)

  It accepts a radial function handle *func*, and the grid points and the actual tabulated data are returned in *rg* and *data*, which must be passed as allocatable real arrays.

4.7. Editing of ps structures

The PSML library has currently some limited support for editing the content of ps_t objects from user programs. For example, such an editing might be done by a KB-projector generator to insert a new provenance record (and KB and local-potential data) in the ps_t object, prior to dumping to a new PSML file. Editing operations not yet supported directly by a given version can still be carried out by a direct handling of the internal structure of the ps_t object, which is for now also visible to client programs.

- ps_RootAttributes_Set(*ps,version,uuid,namespace*)
- ps_Provenance_Add(*ps,creator,date,annotation*)

  Annotations can be created by client programs using routines exported by the PSML API (see Section 4.9).

- ps_NonlocalProjectors_Delete(*ps*)
- ps_LocalPotential_Delete(*ps*)

  Only "deletion" operations are supported as yet.

4.8. Dump of ps structures

The contents of a (possibly edited) ps_t object can be dumped to a PSML file using the routine

- ps_DumpToPSMLFile (*ps,fname,indent*)

  Here *fname* is the output file name, and *indent* is a logical variable that determines whether automatic indenting of elements is turned on (by default it is not).

In principle, there could be *dumpers* for other file formats, but their implementation is better left to specialized programs that are clients of the library.

4.9. Annotation API

To support the annotation functionality (see Sec. 2.10), the library contains a module with a basic implementation of an association list (a data structure holding key-value pairs). The library exports the ps_annotation_t type, an empty annotation object EMPTY_ANNOTATION, and the following routines:

- reset_annotation(*annotation*)

  Cleans the contents of the ps_annotation_t object *annotation* so that it can be reused.

- insert_annotation_pair(*annotation,key,value,stat*)

  Inserts the *key, value* pair of character variables in the ps_annotation_t object *annotation*. Internally, *annotation* can grow as much as needed.

- function nitems_annotation(*annotation*) result(*nitems*)

  Returns the number of key-value pairs in the annotation object

- get_annotation_value(*annotation,key,value,stat*)

- get_annotation_value(*annotation,i,value,stat*)

  This routine has two interfaces. The first gets the *value* associated to the *key*, and the second gets the *value* associated to the *i*'th entry in the annotation object.

- get_annotation_key(*annotation,i,key,stat*)

  Gets the *key* of the *i*'th entry in the annotation object.

  Together with the second form of get_annotation_value, this routine can be used to scan the complete annotation object. The first form of get_annotation_value is appropriate if the key(s) are known.

In all the above routines a non-zero *stat* signals an error condition.

5. Discussion: the PSML model and ecosystem

Our vision for the role of PSML in addressing the interoperability and documentation problems is as follows:

- Databases offer PSML files (with smart searching made possible by the clear internal structure) and most codes use them directly. As they have full provenance and a uuid tag built in, calculations can properly document the pseudopotentials used. In some cases, specific legacy formats can also be produced from PSML files.

- PSML files are produced directly by most pseudopotential generators, either *de novo* or through conversion of existing formats (with extra metadata added).

- The PSML ideas and technology are extended to include PAW datasets and ultra-soft pseudopotentials.

The PSML format is able to span a wide range of uses for norm-conserving pseudos, from semilocal-only data files, up to full-operator datasets, with flexibility (e.g.: cascading grids), robustness (e.g.: comprehensive exchange-correlation specification) and full provenance. It does so in an extensible manner, and we plan to adapt it to include PAW and USPP in the future.
To appreciate the place of PSML in relation to similar work, we can list its strengths compared to the corresponding subset of Quantum Espresso's UPF format (which however already supports PAW datasets and ultra-soft pseudopotentials):

- The PSML format is accompanied by a complete stand-alone processing library that eases its adoption by client codes.
- Support for alternative forms of datasets in the same file (i.e., "scalar-relativistic" plus "spin-orbit" and/or "lj" form).
- Support for different grids for different radial functions.
- API based on interpolation that can suit any client grid, without having to adapt the client code to the dataset grid.
- Full provenance specification, even when various codes are involved in different stages of the pseudopotential generation (i.e., semilocal generation followed by KB transformation).
- A more complete specification of the exchange-correlation functional(s) used through a LIBXC-compatible scheme.
- A complete and parseable specification of the valence configuration used for generation, and of the reference energies used for the projectors.

The above list could serve also in a comparison of PSML with the QSO XML-based norm-conserving pseudopotential format (See Ref. [38]).
As a proof of concept of the above vision, we have modified two different atomic pseudopotential generation codes to generate PSML files, and interfaced *libPSML* to two electronic-structure programs.
The first generator is the open-source ONCVPSP code implemented by D. Hamann [22] to generate optimized multiple-projector norm-conserving pseudopotentials. The projectors are directly stored in the PSML format together with the local potential. In addition, a set of semi-local potentials, a by-product of the ONCVPSP algorithm, is also included in the PSML file. The patches needed to produce PSML output in ONCVPSP are available in the Launchpad code development platform [52]. To ease the production of XML, a special library (*wxml*, part of the xmlf90 project maintained by one of the authors (A.G.)) [53] is used.
The second generator enabled for PSML output is the ATOM code, originally developed by S. Froyen, later modified by N. Troullier and J. L. Martins, and currently maintained by one of us (A. G.) within the SIESTA project. ATOM, freely distributed to the academic community, generates norm-conserving pseudopotentials in the semilocal form. We have developed a post-processing tool (psop) which takes as input the semilocal components and computes a smooth local potential and the KB projector functions in the same way as it is done within the SIESTA code. These new elements, together with a new provenance record, are incorporated in a new PSML file, which describes a well-defined, client-code independent and unique operator.
We have thus already two different generators of PSML files, their specific idiosyncrasies being describable by a common standard. Our plans are to enable PSML output in other pseudopotential-generation codes.

Table 5: Reference configuration and cutoff radii of the optimized norm-conserving pseudopotentials generated with the ONCVPSP code by Hamann [22]. Units in bohr.

Reference		Au $5s^2, 5p^6, 5d^{10}, 6s^1$	Fe $3s^2, 3p^6, 3d^6, 4s^2$
Cutoff radius	s	1.456	1.163
	p	1.564	1.109
	d	1.564	1.310
	f	1.602	-
Exchange-correlation		GGA-PBE [55]	GGA-PBE [55]
Scalar relativistic?		yes	yes
Core corrections (NLCC)?		yes	yes
NLCC cutoff radius		0.783	0.417

On the client side, we have incorporated the *libPSML* library in SIESTA (version 4.2, soon to be released) and ABINIT (version 8.2 and higher). PSML files can then be directly read and digested by these codes as described in Sec. 4, achieving pseudopotential interoperability between these codes, as exemplified in Sec. 6 below.
Our immediate plans include the development of a UPF-to-PSML converter and to encourage the adoption of *libPSML* to enable the interoperability of SIESTA and the codes in the Quantum Espresso suite.
We are aware that a complete deployment of the PSML road-map will require the development of interfaces to *libPSML* in other languages, such as C and Python, which are in progress.

6. Interoperability example: local-orbital and PW calculations with the same pseudopotential

We present in this section two examples of interoperability between SIESTA and ABINIT using PSML files: (i) the test of the convergence of a numerical atomic orbital basis set with respect to the asymptotic limit of a converged basis of plane waves, and (ii) the equation-of-state (energy versus volume profiles) for elemental crystals, a test that has been proposed as a benchmark for the comparison of different codes [54].
Four paradigmatic systems are chosen as a testbed: a standard semiconductor (bulk Si in the diamond structure), an *sp*-metal (bulk Al in the fcc structure), a noble metal (bulk Au in the fcc structure), and a 3*d* ferromagnetic transition metal (bulk Fe in the bcc structure).
The reference electronic configurations, cutoff radii, core corrections flags, and other parameters required to generate the pseudopotentials with the ONCVPSP code (version 3.3.0) are taken from the Pseudo-Dojo database [39], and summarized in Table 5. The pseudopotentials for Au and Fe include the semicore states (5*s* and 5*p* for Au, and 3*s* and 3*p* for Fe) in the valence.

Table 6: Reference configuration and cutoff radii of the norm-conserving pseudopotentials generated with the ATOM code following the Troullier-Martins scheme [23]. LDA refers to the local density approximation functional of Ceperley and Alder [56] as parametrized by Perdew and Zunger [57]. GGA refers to the generalized gradient approximation functional proposed by Perdew, Burke and Ernzerhof. [55] Units in bohr.

Reference		Si $3s^2, 3p^2$	Al $3s^2, 3p^1$	Fe $4s^2, 3d^6$
Cutoff radius	s	1.90	2.30	2.25
	p	1.90	2.30	2.75
	d	1.90	2.30	2.00
	f	1.90	2.30	2.00
Exchange-correlation		LDA [56, 57]	LDA [56, 57]	GGA [55]
Relativity option		non-relativistic	non-relativistic	scalar-relativistic
Core corrections (NLCC)?		no	no	yes
NLCC cutoff radius		-	-	0.70

For the Troullier-Martins pseudopotentials generated with the ATOM code, we use the parameters reported in Table 6. The rest of the technical details of the calculations, that are common to both SIESTA and ABINIT simulations, are given in Table 7.

Table 7: Parameters used in the simulations of bulk Si, Al and Fe that are kept the same in SIESTA and ABINIT. FD stands for Fermi-Dirac smearing.

	Si	Al	Au	Fe
Atomic structure	Diamond	FCC	FCC	BCC
Exchange-correlation	LDA [56, 57]	LDA [56, 57]	GGA [55]	GGA [55]
Spin polarized?	No	No	No	Yes
Monkhorst-Pack mesh	$6 \times 6 \times 6$	$20 \times 20 \times 20$	$20 \times 20 \times 20$	$20 \times 20 \times 20$
Occupation option	–	FD	FD	FD
Smearing temperature	–	0.01 Ha	0.01 Ha	0.01 Ha

In SIESTA, the electronic density, Hartree, and exchange correlation potentials, as well as the corresponding matrix elements between the basis orbitals, were calculated on a uniform real space grid, controlled by an energy cutoff [58] which was well converged at values of 300 Ry for Si, 400 Ry for Al and Fe, and 600 Ry for Au. The reader has to keep in mind that these cutoffs are not directly comparable to plane-wave cutoffs, and that they control the operation count of stages of the calculation which typically represent a small fraction of the total computation time.

Figure 1 shows the convergence of the total energy for Al, and Fe, using ATOM-generated pseudopotentials, and for Au using ONCVPSP pseudopotentials, as a function of the basis-set quality. For ABINIT, the latter can simply be measured by the plane-wave energy cutoff. For SIESTA, the hierarchy and the nomenclature of the basis sets of numerical atomic orbitals is described in Ref. [59]. For a given tier within the basis hierarchy, the NAO basis sets used to produce the results of Fig. 1 were generated using the default parameters implemented in SIESTA. The only exceptions are those marked with an "opt" suffix for the ATOM pseudopotentials, that were optimized following the recipe given in Ref. [60], and are available on the SIESTA web page [61]. The "optimized" NAO basis sets of Au and Fe with the ONCVPSP pseudopotentials were generated following the automatic procedure implemented in SIESTA with the cutoff radii for all the shells defined by a unique parameter: the energy shift [62] (0.005 Ry for Au and 0.002 Ry for Fe). The calculations are made at lattice constants of 3.97 Å for Al, 2.87 Å for Fe, and 4.16 Å for Au.

Although the convergence of NAO results is not *a priori* systematic with respect to the size of the basis, the sequence of bases presented in Fig. 1 shows a uniform convergence of the total energy with respect the basis size. This is specially remarkable for fully optimized basis sets, such as the one used with the DZP quality for Fe and Al, where several eV can be gained. But significant reductions in the total energy can be obtained simply by tuning a reduced subset of the parameters that define the atomic orbitals. This is exemplified here in the case of metallic bulk Au, where the total energy is lowered by almost 2 eV increasing the range of the atomic orbitals for the same DZP size of the basis. In any case, we can observe how the polarization orbitals are important for convergence, much more so than doubling the basis set. A basis of relatively modest size (DZP) is equivalent, from the total energy point of view, to the PW basis cutoff which would be used in realistic calculations (9 Ha for Al, 35 Ha for Fe, or 23 Ha for Au).

It is important to stress that, when using the same pseudopotential, the total energies are given with respect to the same reference and, therefore, can be *directly* compared. The use of a common pseudopotential format thus allows a more straightforward and detailed analysis of convergence.

In related work, a comparison of energy differences, ionic forces and average pressures for water monomers, dimers, two phases of ice and liquid water at ambient and high density have been presented in Ref. [63]. They were obtained with SIESTA and ABINIT using the same pseudopotentials (with an early version of the PSML framework). Highest order bases are shown to give accuracies comparable to a plane-wave kinetic energy cutoff of around 1000 eV.

Figure 2 shows the comparison of the equations of state for bulk Si, Al, Fe and Au, computed with a basis set of double-zeta polarized NAOs and for a PW basis set of comparable quality (see Fig. 1). Both the position of the minimum energy and the curvature of the energy as a function of volume are very similar, indicating that for the same quality of the basis we can obtain essentially the same structural information.

The distance between the two curves, for the volume range plotted in Fig. 2, can

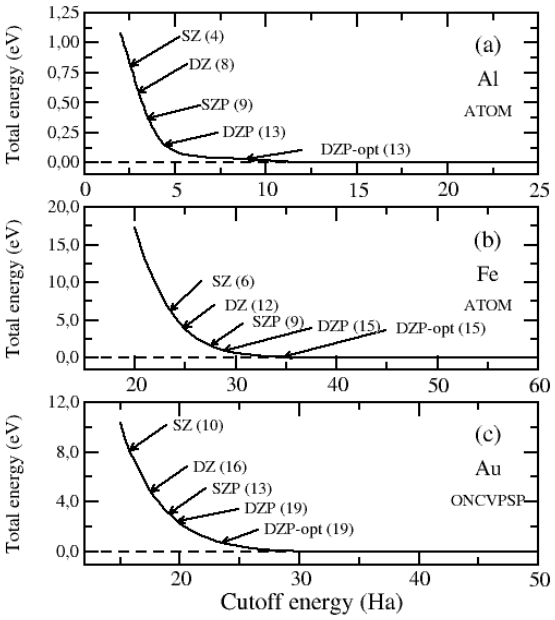

Figure 1: Comparison of the convergence of a basis set of numerical atomic orbitals (NAOs) and a basis set of plane waves for (a) bulk Al in the fcc structure, (b) bulk Fe in the ferromagnetic bcc structure, and (c) bulk Au in the fcc structure. The pseudopotentials for Al and Fe were generated within the Troullier-Martins schema as implemented in the ATOM code, while the pseudo for Au was generated with the ONCVPSP code. SZ and DZ stand, respectively, for single-ζ and double-ζ quality of the basis set. P stands for polarized. When semicore states are included in the valence, the corresponding shells are always treated at the SZ level. The total energy of a well converged PW calculation (25 Ha for Al, 60 Ha for Fe, and 50 Ha for Au) has been taken as the reference zero energy (dashed line). The number in parenthesis indicate the number of NAOs considered per atom.

be further quantified using the delta-factor[64]. Taking the plane wave equation of state as a reference, we quantify the delta values that are reported in Table 8. In every case, the delta factor is smaller than 1 meV/atom, demonstrating the excellent agreement obtained between the two codes, and highlighting the level of interoperability achievable.

7. Conclusions

We have presented the PSML norm-conserving pseudopotential file format and the associated open source *libPSML* library for parsing and data handling. PSML is based on XML and implements provenance and flexibility in a widely applicable and extensible format. We demonstrate its potential for enabling interoperability among electronic-structure codes by comparing results from a plane wave (ABINIT) and an atomic orbital code (SIESTA), using the same input. We

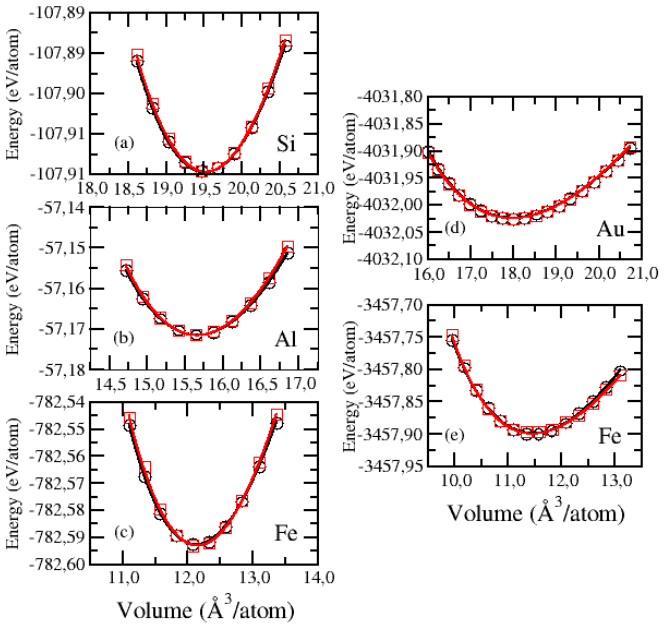

Figure 2: (Color online) Equation of state (energy versus volume) for crystalline solids using the same pseudopotential operator but different basis sets (plane waves as implemented in ABINIT (black solid lines and circles), and numerical atomic orbitals (DZP) as implemented in SIESTA (red solid lines and squares)). Left column: Troullier-Martins norm-conserving pseudopotentials obtained from the ATOM code. Right columne: optimized norm-conserving pseudopotential computed from the ONCVPSP code. The cutoff energies used in the PW calculations were those corresponding to the optimized NAO basis set of DZP quality, according to Fig. 1. For ATOM pseudopotentials: 13 Ha for Si, 9 Ha for Al and 35 Ha for Fe. For ONCVPSP pseudopotentials: 51 Ha for Fe, and 23.5 Ha for Au. Total energies per atom are shown to highlight the reproducibility of the results in absolute terms.

find a systematic convergence in absolute values of energies, and a delta factor of less than 1 meV.

8. Acknowledgements

We thank Xavier Gonze in particular for backing this project and providing useful comments. Many constructive discussions are acknowledged with Don Hamann, Matteo Giantomassi, Michiel Van Setten, Paolo Giannozzi, Gian-Marco Rignanese, and François Gygi. This work was supported by CECAM through the Electronic Structure Library (ESL) initiative and the ETSF through the libpspio project. MJV acknowledges support from ULg and CfWB through ARC projects AIMED and TheMoTherm (GA 15/19-09 and 10/15-03) and a FNRS PDR project (GA T.1077.15-1/7). A.G. was funded by EU H2020 grant 676598 ("MaX: Materials at the eXascale" CoE), Spain's MINECO (grants

Table 8: Lattice constant (a) and bulk modulus (B) for bulk Si, Al, Fe and Au obtained after fitting the equation of state, Fig. 2, to a Birch-Murnaghan equation. For Fe we also show the magnetic moment (M) at the minimum energy structure. Δ refers to the delta factor [64, 54], taking the plane wave results as reference. Experimental numbers from Ref. [65].

	PW	NAO	Expt.
Si			
a (Å)	5.384	5.385	5.430
B (GPa)	95.7	100.2	98.8
Δ (meV/atom)	0.93		
Al			
a (Å)	3.971	3.971	4.05
B (GPa)	80.9	87.4	72.2
Δ (meV/atom)	0.51		
Fe (ATOM)			
a (Å)	2.895	2.896	2.870
B (GPa)	137.6	150.6	168.3
M (μ_B)	2.32	2.35	2.22
Δ (meV/atom)	0.82		
Fe (ONCVPSP)			
a (Å)	2.841	2.840	2.870
B (GPa)	175.9	172.4	168.3
M (μ_B)	2.19	2.18	2.22
Δ (meV/atom)	0.24		
Au (ONCVPSP)			
a (Å)	4.158	4.158	4.08
B (GPa)	139.3	139.4	173
Δ (meV/atom)	0.14		

FIS2012-37549-C05-05 and FIS2015-64886-C5-4-P, and the "Severo Ochoa" Program grant SEV-2015-0496), and GenCat (2014 SGR 301). JJ and YP acknowledge support from Spain's MINECO (grants RTC-2016-5681-7 and FIS2015-64886-C5-2-P).

[1] R. M. Martin, Electronic Structure. Basic Theory and Practical Methods, Cambridge University Press, Cambridge, 2004.

[2] J. Kohanoff, Electronic structure calculations for solids and molecules, Cambridge University Press, Cambridge, 2006.

[3] P. Hohenberg, W. Kohn, Inhomogeneous electron gas, Phys. Rev. 136 (1964) B864.

[4] W. Kohn, L. J. Sham, Self-consistent equations including exchange and correlation effects, Phys. Rev. 140 (1965) A1133.

[5] O. K. Andersen, Linear methods in band theory, Phys. Rev. B 12 (1975) 3060–3083.

[6] D. Singh, Planewaves, pseudopotentials and the LAPW method, Kluwer Academic Publishers, Dordrecht, 1994.

[7] H. Skriver, The LMTO method, Springer, New York, 1984.

[8] J. Korringa, On the calculations of the energy of a bloch wave in a metal, Physica 13 (1947) 392.

[9] W. Kohn, N. Rostoker, Solution of the schrodinger equation in periodic lattices with an application to metallic lithium, Phys. Rev. 94 (1954) 1111.

[10] J. C. Phillips, L. Kleinman, New method for calculating wave functions in crystals and molecules, Phys. Rev. 116 (1959) 287–294.

[11] J. Harris, R. O. Jones, Pseudopotentials in density functional theory, Phys. Rev. Lett. 41 (1978) 191–194.

[12] D. R. Hamman, M. Schlüter, C. Chiang, Norm-conserving pseudopotentials, Phys. Rev. Lett. 43 (1979) 1494–1497.

[13] G. P. Kerker, Non singular atomic pseudopotentials for solid state applications, J. Phys. C: Solid St. Phys. 13 (1980) L189–94.

[14] D. Vanderbilt, Soft self-consistent pseudopotentials in a generalized eigenvalue formalism, Phys. Rev. B 41 (1990) 7892–7895.

[15] P. E. Blöch, Projector augmented-wave method, Phys. Rev. B 50 (1994) 17953–17979.

[16] The FHI98PP pseudopotential program, `http://www.fhi-berlin.mpg.de/th/fhi98md/fhi98PP/`, accessed July 2017.

[17] Atomic Pseudopotentials Engine (APE), `http://tddft.org/programs/APE/`, accessed July 2017.

[18] OPIUM - pseudopotential generation project, `http://opium.sourceforge.net`, accessed July 2017.

[19] ATOM code for the generation of norm-conserving pseudopotentials. The version maintained by the SIESTA project can be accessed at `http://icmab.es/siesta/Pseudopotentials/index.html`. An alternative version is available at `http://bohr.inesc-mn.pt/~jlm/pseudo.html`. (Accessed July 2017).

[20] Open-source pseudopotential code ONCVPSP, `http://www.mat-simresearch.com`, accessed July 2017.

[21] See the atomic package, included in the QUANTUM ESPRESSO distribution `http://www.quantum-espresso.org`, accessed July 2017.

[22] D. R. Hamann, Optimized norm-conserving Vanderbilt pseudopotentials, Phys. Rev. B. 88 (2013) 085117.

[23] N. Troullier, J. L. Martins, Efficient pseudopotentials for plane-waves calculations, Phys. Rev. B. 43 (1991) 1993.

[24] M. Teter, Additional condition for transferability in pseudopotentials, Phys. Rev. B 48 (1993) 5031–5041.

[25] M. Fuchs, M. Scheffler, Ab initio pseudopotentials for electronic structure calculations of poly-atomic systems using density-functional theory, Comput. Phys. Commun. 119 (1999) 67–98.

[26] L. Kleinman, D. M. Bylander, Efficacious form for model pseudopotentials, Phys. Rev. Lett. 48 (1982) 1425.

[27] J. M. Soler, E. Artacho, J. D. Gale, A. García, J. Junquera, P. Ordejón, D. Sánchez-Portal, The Siesta method for ab initio Order-N materials simulations, J. Phys.: Condens. Matter 14 (2002) 2745–2779.

[28] See `https://wiki.fysik.dtu.dk/gpaw/setups/pawxml.html`, accessed July 2017.

[29] See `http://www.quantum-espresso.org/pseudopotentials/unified-pseudopotential-format/`, accessed July 2017.

[30] The Materials Project, `https://www.materialsproject.org`, accessed July 2017.

[31] G. Ceder, K. Persson, How supercomputers will yield a golden age of materials science, Scientific American 309 (2013) 36–40.

[32] G. Pizzi, A. Cepellotti, R. Sabatini, N. Marzari, B. Kozinsky, AiiDa: automated interactive infrastructure and database for computational science, Comp. Mat. Sci. 111 (2016) 218–230.

[33] K. F. Garrity, J. Bennett, K. Rabe, D. Vanderbilt, Pseudopotentials for high-throughput dft calculations, Comput. Mater. Sci. 81 (2014) 446.

[34] GBRV (Garrity-Bennett-Rabe-Vanderbilt) high-throughput pseudopotentials, `http://www.physics.rutgers.edu/gbrv/`, accessed July 2017.

[35] See `http://www.quantum-espresso.org/pseudopotentials/pslibrary/`, accessed July 2017.

[36] F. Jollet, M. Torrent, N. Holzwarth, Generation of projector augmented-wave atomic data: A 71 element validated table in the XML format, Comput. Phys. Commun. 185 (4) (2014) 1246 – 1254.

[37] M. Schlipf, F. Gygi, Optimization algorithm for the generation of ONCV pseudopotentials, Comput. Phys. Commun. 196 (2015) 36.

[38] See http://www.quantum-simulation.org/potentials/sg15_oncv/, accessed July 2017.

[39] See the PSEUDO-DOJO web page: http://www.pseudo-dojo.org, accessed July 2017.

[40] See https://www.w3.org/XML/, accessed July 2017.

[41] See http://esl.cecam.org/, accessed July 2017.

[42] See https://en.wikipedia.org/wiki/Universally_unique_identifier, accessed July 2017.

[43] D. D. Koelling, B. N. Harmon, A technique for relativistic spin-polarised calculations, J. Phys. C 10 (1977) 3107.

[44] T. Takeda, The scalar relativistic approximation, Z. Physik B 32 (1978) 43–48.

[45] G. B. Bachelet, D. R. Hamman, M. Schlüter, Pseudopotentials that work: From H to Pu, Phys. Rev. B 26 (1982) 4199.

[46] S. C. Watson, E. A. Carter, Spin-dependent pseudopotentials, Phys. Rev. B 58 (1998) R13309–R13313.

[47] S. García-Gil, A. García, P. Ordejón, Calculations of core level shifts within dft using pseudopotentials and localized basis sets, Eur. Phys. J. B 85 (2012) 239.

[48] S. G. Louie, S. Froyen, M. L. Cohen, Nonlinear ionic pseudopotentials in spin-density-functional calculations, Phys. Rev. B 26 (1982) 1738.

[49] M. A. L. Marques, M. J. T. Olivera, T. Burnus, Libxc: A library of exchange and correlation functionals for density functional theory, Comp. Phys. Comm. 10 (2012) 2272–2281.

[50] http://octopus-code.org/wiki/Libxc, accessed July 2017.

[51] RELAX NG home page: http://www.relaxng.org, accessed July 2017.

[52] See http://launchpad.net/pspgenpatch, accessed July 2017.

[53] See http://launchpad.net/xmlf90, accessed July 2017.

[54] K. Lejaeghere, G. Bihlmayer, T. Björkman, P. Blaha, S. Blügel, V. Blum, D. Caliste, I. E. Castelli, S. J. Clark, A. Dal Corso, S. de Gironcoli, T. Deutsch, J. K. Dewhurst, I. Di Marco, C. Draxl, M. Dułak, O. Eriksson, J. A. Flores-Livas, K. F. Garrity, L. Genovese, P. Giannozzi, M. Giantomassi, S. Goedecker, X. Gonze, O. Grånäs, E. K. U. Gross, A. Gulans, F. Gygi, D. R. Hamann, P. J. Hasnip, N. A. W. Holzwarth, D. Iuşan, D. B. Jochym, F. Jollet, D. Jones, G. Kresse, K. Koepernik, E. Küçükbenli,

Y. O. Kvashnin, I. L. M. Locht, S. Lubeck, M. Marsman, N. Marzari, U. Nitzsche, L. Nordström, T. Ozaki, L. Paulatto, C. J. Pickard, W. Poelmans, M. I. J. Probert, K. Refson, M. Richter, G.-M. Rignanese, S. Saha, M. Scheffler, M. Schlipf, K. Schwarz, S. Sharma, F. Tavazza, P. Thunström, A. Tkatchenko, M. Torrent, D. Vanderbilt, M. J. van Setten, V. Van Speybroeck, J. M. Wills, J. R. Yates, G.-X. Zhang, S. Cottenier, Reproducibility in density functional theory calculations of solids, Science 351 (6280). doi:10.1126/science.aad3000.

[55] J. P. Perdew, K. Burke, M. Ernzerhof, Generalized gradient approximation made simple, Phys. Rev. Lett. 77 (1996) 3865.

[56] D. M. Ceperley, B. J. Alder, Ground state of the electron gas by a stochastic method, Phys. Rev. Lett. 45 (1980) 566–569.

[57] J. P. Perdew, A. Zunger, Self-interaction correction to density-functional approximations for many-electron systems, Phys. Rev. B 23 (1981) 5048.

[58] P. Ordejón, E. Artacho, J. M. Soler, Self-consistent Order-N density-functional calculations for very large systems, Phys. Rev. B 53 (1996) R10441.

[59] J. Junquera, O. Paz, D. Sánchez-Portal, E. Artacho, Numerical atomic orbitals for linear-scaling calculations, Phys. Rev. B 64 (2001) 235111.

[60] E. Anglada, J. M. Soler, J. Junquera, E. Artacho, Systematic generation of finite-range atomic basis sets for linear-scaling calculations, Phys. Rev. B 66 (2002) 205101.

[61] See the SIESTA web page: http://www.icmab.es/siesta, accessed July 2017.

[62] E. Artacho, D. Sánchez-Portal, P. Ordejón, A. García, J. M. Soler, Linear-scaling ab-initio calculations for large and complex systems, Phys. Stat. Sol. (b) 215 (1999) 809.

[63] F. Corsetti, M.-V. Fernández-Serra, J. M. Soler, E. Artacho, Optimal finite-range atomic basis sets for liquid water and ice, J. Phys.: Condens. Matter 25 (2013) 435504.

[64] K. Lejaeghere, V. V. Speybroeck, G. V. Oost, S. Cottenie, Error estimates for solid-state density-functional theory predictions: an overview by means of the ground-state elemental crystals, Crit. Rev. Solid State 39 (2014) 1–24.

[65] C. Kittel, Introduction to Solid State Physics, John Wiley & Sons, New York, 1986.