3rd International ABINIT Developer Workshop

LIÈGE - BELGIUM

ETSF_IO, *a new library to access electronic structure calculation files*

Damien Caliste

PCPM - UCL Belgium

30 January 2007

Main goals of the implementation

Even with simple specifications, a new norm won't be used without an implementation.

The present work is an attempt to:

- give an unified and complete implementation ;
- achieve simplicity and briefness in calls ;

`write(file) array` (plain Fortran) should become
`call etsf_io_write(file, array)`

- create a code-independent implementation (users should not adapt their codes to the library) ;
- follow high coding standards

- Fortran90 modules for better compilation checks,
- automatic tests,
- inline documentation,
- and tutorials.

Framework of the library

The library is made of different levels, each of them characterised by one library file (*e.g.* `libetsf_io.a`) in association with one module file (*e.g.* `etsf_io.mod`).

the `src` directory of the package

- `low_level`: several wrapper routines to simplify and automate most common NetCDF calls.
- `group_level`: the core module of the library. It gives routines to read/write data related to ETSF specifications in very few calls.
- `utils`: contains some related tools to handle ETSF files and data (*e.g.* a merging tool for files created by a MPI run, or validity checkers).
- `tutorials`: several highly commented examples for common use of the library (*e.g.* how to write wavefunctions one *k* point per one *k* point).

Inside the low level module

Simplicity and briefness

With pure NetCDF calls, a full-proof code should have several calls to access a single value.

`error_id = nf90_inquire_variable(varid);` *treat error_id*

`error_id = nf90_inquire_variable(ndims);` *treat error_id*

`error_id = nf90_inquire_variable(dimids);` *treat error_id*

checks on `ndims` *and* `dimids`

`error_id = nf90_put_variable();` *treat error_id*

With low level access everything is done in one call.

```
call etsf_io_low_write_var(ncid, varname, var, lstat)
```

- Using Fortran interfaces, the type of `var` is generic (integer, double precision, 1D array to 7D array, character...).
- All `etsf_io_low` calls returns optional verbose error structure, detailling which action failed, for what reason, in which routine and on which variable or dimension.

Data storage in group level

- The etsf_dims type stores all dimensions related to ETSF variables.

```
 ...
integer ::  max_number_of_states = 320
integer ::  number_of_atoms = 44

...
```

- Several etsf_*<grpname>* types gathering variables of related meaning.

```
 etsf_wavedata
double precision, pointer ::  kinetic_energy_cutoff => null()
integer, pointer ::  number_of_coefficients(:)  => null()

...
```

All type fields are pointers to avoid memory duplication.

Actions in group level

Possible actions are define, write, read and copy:

- `define` is used to declare the variables in the files (following values in `etsf_dims`). All variables of each group are declared and space on disk is allocated.
- `write/read` are used to access variables. Only associated pointers of the group will result in an action.
- `copy` is used to copy a group of variable from one file to another (variable in the destination file should have been defined).

All these action required to have an opened file.

High level routines prefixed with `etsf_io_data_` work as one-call routines, being able to open a file, acess several groups and close the file in one call.

Characteristics of data storage in group level

- ✔ Briefness , using high level calls and `etsf_<grpname>` ;
- ✔ (mostly) code-independent, using custom generic pointers ;
- ✔ strict and complete about specifications, with
  - possible sub access (see tutorial) ;

```
! We associate the data
main%coefficients_of_wavefunctions%data2D => main_prog_array
! We set the sub access.
main%wfs_pw__kpoint_access = i_kpt
! We use the group level write routine.
call etsf_io_main_put(ncid, main, lstat, error_data)
```

  - transparent support for attributes (*e.g.* units are converted on the fly or on demand) ;
  - support for split variables.
- ✔ build by scripts (easy integration of modifications in the specs) ;

Conclusion and outlooks

Current usage:

- In ABINIT,
  - Read/write the electronic density (requires the geometry group and the main group) ;
  - Read/write the coefficient of wavefunctions (done spin by spin and *k* point per *k* point, using sub access) ;
  - Write the data related to GW.
- In `src/utils` of the package,
  - Read several files with split definitions and create a new merged file using the copy actions and the split support.

Roadmap, version 1.0 targetted on 1st Marsh

- add validity checkers and miscelaneous routines ;
- implement two more tutorials.

Goal of stability with an API freeze for all 1.x versions (existing routines will be kept unchanged, only new routines may be added).