Introduction to *Quantum ESPRESSO* for would-be developers

*P. Giannozzi*

Università di Udine and IOM-Democritos, Trieste, Italy

25 March 2013
Developers’ Resources

*When everything else fails, read the manual!*

There isn’t a real **QUANTUM ESPRESSO** manual, but there are some useful resources for developers:

- the pseudo-manual `Doc/developers-man.pdf`, plus other files in `Doc/`, `install/`, `include/`;
- scattered documentation in source files;
- `www.quantum-espresso.org`, **Resources** menu, under **Developers’ Resources**
- the **q-e** project on `qe-forge.org`, and in particular the bug tracker, forum, mailing list archives, web-svn interface.
The QUANTUM ESPRESSO ecosystem (zoo?)

Quantum ESPRESSO package portfolio

(core modules: CP, PWscf, PHonon, TDDFPT, NEB, Atomic)

packages: core, porpolio

GIPAW, SaX, WanT, Wannier90, PLUMED, Xspectra, YAMBO, GWL, PWCOND, PLUMED, YAMBO

(the above slide courtesy of Filippo Spiga)
Core QUANTUM ESPRESSO distribution

In the following I will refer to the core distribution, including

- scripts, installation tools, libraries, common source files;
- basic packages
  - PWscf: self-consistent calculations, structural optimization, molecular dynamics on the ground state;
  - CP: Car-Parrinello molecular dynamics;
  - PostProc: data analysis and plotting (requires PWscf)

All of the above is stored into a single ('main') SVN repository. Some libraries are however downloaded on demand from the web.

Most of what will be said is valid also for additional packages.
More quantum ESPRESSO distribution

Additional packages that can be downloaded and installed on demand:

- atomic: pseudopotential generation
- PHonon: Density-Functional Perturbation Theory
- NEB: reaction pathways and energy barriers
- PWCOND: ballistic conductance
- XSPECTRA: calculation of X-Ray spectra
- TDDFPT: Time-dependent DFPT (requires PHonon)

All these require that the core distribution is already installed.
Stored in the main SVN repository along with the core distribution.
Additional **Quantum ESPRESSO** pieces

More packages that can be downloaded and installed on demand:

- **GIPAW**: calculation of NMR coefficients and chemical shifts. Stored in separate SVN repository that can be downloaded into the main SVN repository

- **EPW**: electron-phonon (under development, requires PHonon). SVN: as above

A further package that depends upon the core **Quantum ESPRESSO** distribution (must be separately downloaded and installed):

- **GWL**: GW calculations using Lanczos chains.
## Directory structure of QUANTUM ESPRESSO

Common files and directories in the espresso/ directory:

<table>
<thead>
<tr>
<th>Directory</th>
<th>Content</th>
</tr>
</thead>
<tbody>
<tr>
<td>install/</td>
<td>configure</td>
</tr>
<tr>
<td>include/</td>
<td>make.sys</td>
</tr>
<tr>
<td>archive/</td>
<td>Makefile</td>
</tr>
<tr>
<td>dev-tools/</td>
<td>License</td>
</tr>
<tr>
<td>pseudo/</td>
<td>README</td>
</tr>
<tr>
<td>Doc/</td>
<td>environment_variables</td>
</tr>
<tr>
<td>clib/</td>
<td>flib/</td>
</tr>
<tr>
<td>Modules/</td>
<td>upftools/</td>
</tr>
<tr>
<td>PW/</td>
<td>PP/</td>
</tr>
<tr>
<td>CPV/</td>
<td></td>
</tr>
</tbody>
</table>

PW/, PP/, CPV/ are specific for the respective packages: PWscf, PostProc, CP. The rest is the common part.
Common Files and Directories

Apart from License and README whose meaning is obvious, the other files and directories are related to

- **Installation** (i.e. compilation and linking):
  - install/,
  - dev-tools/,
  - archive/,
  - configure,
  - make.sys

- **Testing** (running tests and examples):
  - pseudo/,
  - environment_variables

- **General documentation** (not package-specific):
  - Doc/

- **C and Fortran Libraries, modules** (F95):
  - clib/,
  - flib/,
  - Modules/
Files and Directories in a Packages

Typical subdirectories of a directory containing a package (e.g. PW/):

- Makefile
- examples/
- tests/
- Doc/
- src/

Note that:

- tests/ contains automated post-installation tests (only in PW/ and CPV/) while examples/ are not suitable for automated checks
- other packages may have a slightly different structure (in particular, PHonon has three directories for sources and none is called src/ )
Compilation and linking

**Quantum ESPRESSO** requires a number of machine-dependent pre-processing, compilation, linking options to be set; plus a number of libraries to be available. Packages can be downloaded, compiled and linked on demand. This is achieved using `configure` and `make` utilities.

Files related to compilation and linking:

- **install/**: documentation and utilities for compilation and linking
- **configure**: wrapper for `install/configure` script
- **make.sys**: file produced by running `configure`, used by `Makefile`, contains machine-specific compilation and linking options
- **Makefile**: contains dependencies and targets used by command `make`
- **include/**: files to be included into sources, to be pre-processed
Compilation and linking (2)

"./configure options" runs install/configure, produces file make.sys. Its behavior can be changed by modifying file install/configure.ac and running (in install/) command autoconf. This produces a new version of install/configure.

"make target" checks for dependencies, recursively goes into subdirectories executing make again. The behavior of make is thus determined by many Makefile's in the various directories. The most important files are Makefile's in the directories containing sources, e.g. Modules/Makefile, PW/src/Makefile.

Dependencies of Fortran files are contained in make.depend files in each source directory. These files must be updated if you change the sources, using script install/makedeps.sh or running "make depend"
The make.sys file

Produced by configure, contains all system-specific information on

- C and Fortran compilers name, pre-processing and compilation options
- whether the Fortran compiler performs C-style preprocessing or not
- whether compiling for parallel or serial execution
- available optimized mathematical libraries, libraries to be downloaded
- Miscellaneous stuff

Written for usage into Makefile's, using the corresponding syntax.
The best documentation for the make.sys file is the file itself.
The top-level Makefile contains the instructions to download, unpack, compile and link what is required. Sample contents:

```makefile
include make.sys

Contains machine- and quantum ESPRESSO-specific definitions

default :
    @echo ’to install, type at the shell prompt:’
    ...

If no target specified, ask for one, giving a list of possibilities

pw : bindir mods liblapack libblas libs libiotk libenviron
    if test -d PW ; then \n        ( cd PW ; if test "$(MAKE)" = "" ; then make $(MFLAGS) TLDEPS= all ; fi ) ; fi
```

Makefile
Target pw: first check the list of dependencies bindir mods ... etc., do what is needed; then go into PW/ and give command make all

neb : bindir mods libs pw
    cd install ; $(MAKE) $(MFLAGS) -f plugins_makefile $@

Target neb: do all of the above, then go into directory install/ where make neb using plugins_makefile as Makefile will check if NEB is there, download from the network if not, compile and link it

libblas : touch-dummy
    cd install ; $(MAKE) $(MFLAGS) -f extlibs_makefile $@

Target libblas: this is an external library, that may or may not be needed, depending upon what is written in make.sys. If needed, go into directory install/ where make libblas using extlibs_makefile as Makefile will check if BLAS are there, download from the network if not, compile and build the library
Second-level Makefile contains only targets related to a given subdirectory or package. Sample contents:

default : all
all: pw pwtools
pw:
   if test -d src ; then \
      ( cd src ; if test "$(MAKE)" = "" ; then make $(MFLAGS) $@; \n        else $(MAKE) $(MFLAGS) ; fi ) ; fi ; \n
Target pw: go into src/ if it exists, and (apart from make wizardry you don't want to know about) give command make pw. Other targets are quite similar: go into a subdirectory, e.g. Doc/ and 'make something', e.g. make clean.
The most important and most complex Makefile is the one in the source directory. It is also the one you need to modify if you add something.

```
include ..../make.sys
```

Contains machine- and *Quantum ESPRESSO*-specific definitions

```
MODFLAGS= $(MOD_FLAG)../iotk/src
    $(MOD_FLAG)../Modules $(MOD_FLAG).
```

Location of needed modules; MOD_FLAG is defined in make.sys

```
PWOBJ = \npwscf.o
```

Object file containing main program (this is actually redundant)
PWLIBS = \n a2fmod.o \n ...
 wannier_enrg.o

List of objects - add here new objects, or delete from this list. Do not forget the backslash! It ensure continuation of the line

QEMODS=../../Modules/libqemod.a

Objects from Modules/ are available from the above archive. The directory where F95 modules are must also be specified to the compiler!

TLDEPS=bindir mods libs liblapack libblas libenviron

TLDEPS=Top-Level DEPendencies: a machinery to ensure proper compilation with correct dependencies also if compiling from inside a package directory and not from top level
LIBOBS = ../../../flib/ptools.a ../../../flib/flib.a
       ../../../clib/clib.a ../../../iotk/src/libiotk.a

All needed QE-specific libraries

all : tldeps pw.x generate_vdW_kernel_table.x

Targets that will be build - add here new executables

pw.x : $(PWOBJS) libpw.a $(LIBOBS) $(QEMODS)
       $(LD) $(LDFLAGS) -o $@
       $(PWOBJS) libpw.a $(QEMODS) $(LIBOBS) $(LIBS)
       - ( cd ../../../bin; ln -fs ../../../PW/src/$@ . )

Target pw.x - produces executable with the same name. It also produces
a link to the executable in espresso/bin/. Do not forget tabulators
even if you do not see them! All variables (introduced by $) are either
defined locally in Makefile or imported from make.sys
libpw.a : $(PWLIBS)
    $(AR) $(ARFLAGS) $@ $? 
    $(RANLIB) $@

This builds the library libpw.a - again, do not forget tabulators

tldeps:
    test -n "$(TLDEPS)" && ( cd ../.. ;
    $(MAKE) $(MFLAGS) $(TLDEPS) || exit 1 ) || : 

second part of the TLDEPS machinery

clean :

There should always be a "clean" target, removing all compiled (*.o) or
preprocessed (*.F90) stuff - compiled F95 modules may have different
filenames: the four last items cover most cases
include make.depend

Contains dependencies of objects upon other objects. Sample content of file make.depend (can be produced by install/makedep.sh):

a2fmod.o : ../../Modules/io_global.o
a2fmod.o : ../../Modules/ions_base.o
a2fmod.o : ../../Modules/kind.o
a2fmod.o : pwcom.o
a2fmod.o : start_k.o
a2fmod.o : symm_base.o

tells us that the listed objects must have been compiled prior to compilation of a2fmod.o - make will take care of this.

BEWARE: the Makefile system is in a stable but delicate equilibrium, resulting from many years if experiments on many different machines. Handle with care: what works for you may break other cases.
Directories containing sources

- **clib/**: a few routines written in C: FFTW, system calls. May include 
  
  *.h files in include/, notably c_defs.h, using the CPP syntax
  
  #include 'c_defs.h'.

- **flib/**: Spherical Bessel functions, Spherical Harmonics, exch-corr
  
  functionals, basic and utility routines written in Fortran-95

- **Modules/**: basic Fortran-95 modules used by all packages

- **PW/src/**: package-specific Fortran-95 main and subroutines

Order of compilation: external libraries (e.g. BLAS, LAPACK) first; 
then clib/, Modules/, flib/, PW/src/

*Note*: distinction between flib/ and Modules/ is by now obsolete
Preprocessing

Fortran-95 source code contains pre-processing option with CPP syntax. Most F95 compilers understand pre-processing options -D ... or some similar form. Pre-processing is useful to

- account for machine dependency in a unified source tree
- distinguish between parallel and serial execution when they follow different paths (i.e. there is a substantial difference between serial execution and parallel execution on a single processor)
- introduce experimental or special-purpose stuff

Use with care and only when needed. See file include/defs.README for a list of pre-processing options. Please keep that list updated.

Note: include/f_defs.h is obsolete and must not be used any longer.
Examples of preprocessing (1)

flib/flush_unit.f90:

#if defined(__XLF) || defined(__ABSOFT)
    #define flush flush_
#endif

[...]

! IF ( opnd ) CALL flush( unit_tobeflushed )
! 

in XLF and ABSOFT compilers, the system routine flush, emptying output buffers, needs an underscore at the end. __XLF or __ABSOFT are defined in make.sys as pre-processing variables at compile time.
Examples of preprocessing (2)

flib/sph_bes.f90:

#if defined (__MASS)
   qr = q * r
   call vcos( cos_qr, qr, msh)
   jl = cos_qr / qr
#else
   jl (:)= cos (q * r (:)) / (q * r (:))
#endif

if __MASS is defined, use the IBM mathematical fast routines MASS. configure finds out whether these routines actually exists for the machine you are compiling on.
if __MPI is defined, the code is compiled for parallel execution. A different routine is called wrt the serial case. __OPENMP accounts for OpenMP parallelization. Note that __PARA and __MPI are used as synonyms, but the former shouldn’t be used any longer.