PSI4 Programming Workshop: Py-side

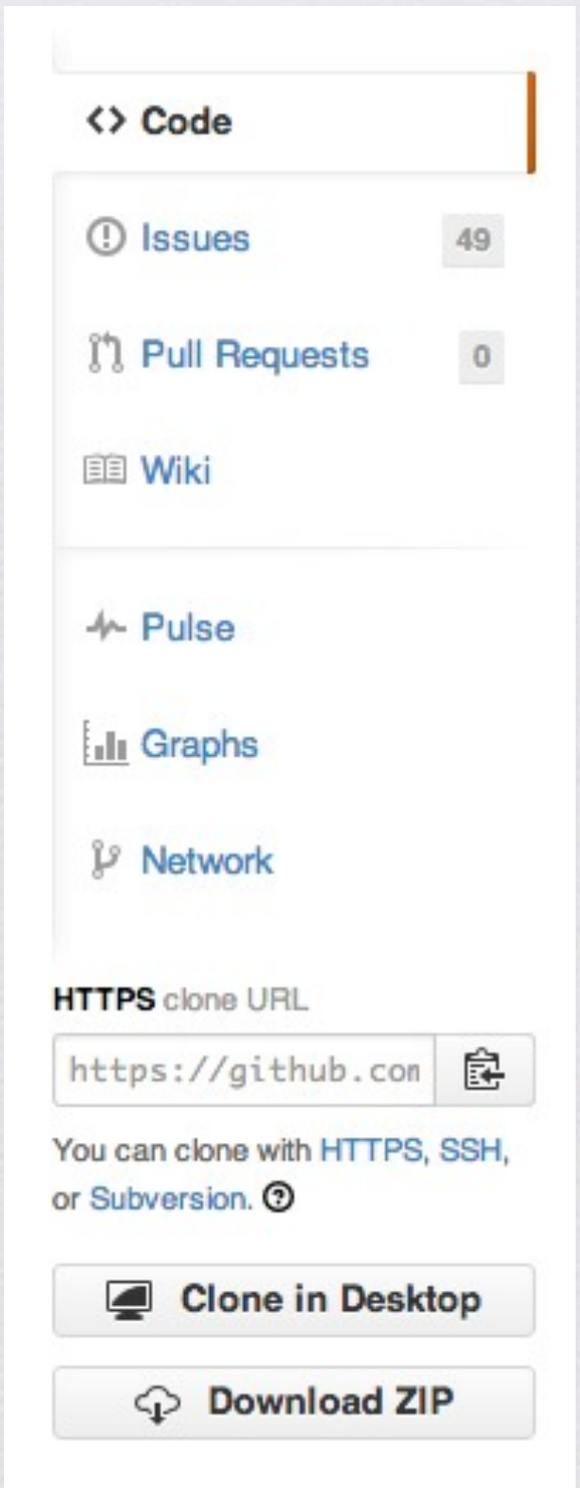
Lori A. Burns

CCMST, Georgia Institute of Technology

24 January 2014

GitHub

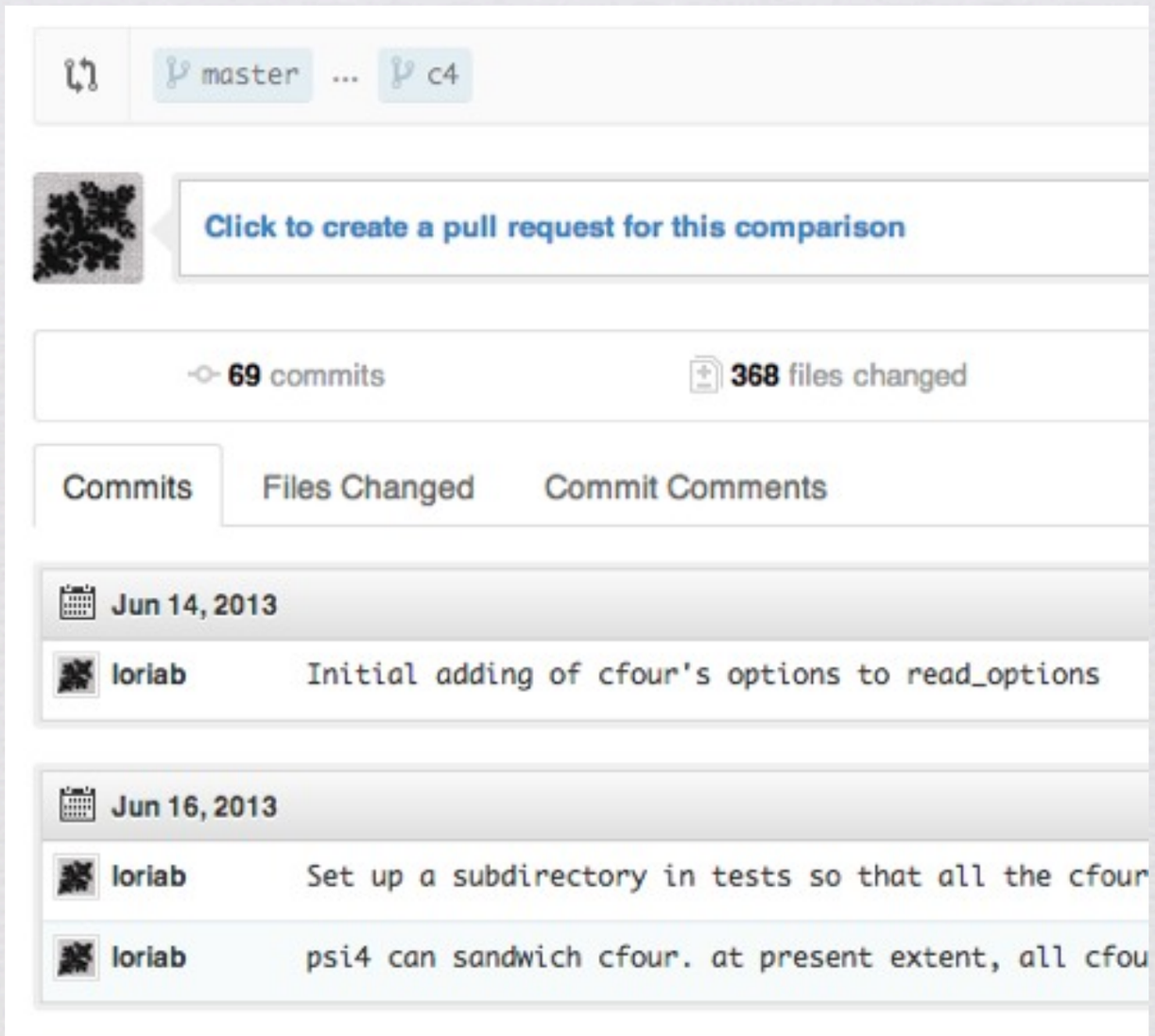
Managing psi4 & psi4public




```
# clone psi4public into target directory
>>> git clone https://github.com/psi4/psi4public.git psi4
>>> git remote -v
origin  https://github.com/psi4/psi4public.git (fetch)
origin  https://github.com/psi4/psi4public.git (push)

# add extra remotes
>>> git remote add pub https://github.com/psi4/psi4public.git
>>> git remote add priv https://github.com/psi4/psi4.git
>>> git remote -v
origin  https://github.com/psi4/psi4public.git (fetch)
origin  https://github.com/psi4/psi4public.git (push)
priv    https://github.com/psi4/psi4.git (fetch)
priv    https://github.com/psi4/psi4.git (push)
pub     https://github.com/psi4/psi4public.git (fetch)
pub     https://github.com/psi4/psi4public.git (push)

# make a commit, then double pull/double push to sync repos
>>> git commit
>>> git pull pub master
>>> git pull priv master
>>> git push pub master
>>> git push priv master
```

GitHub

Working with Branches
Tentative Workflow




```
# make a new branch
>>> git branch
* master
>>> git branch greatcode
>>> git branch
  greatcode
* master
>>> git checkout greatcode
Switched to branch 'greatcode'
>>> git status
# On branch greatcode
nothing to commit (working directory clean)

# introduce branch to GitHub
>>> git push pub greatcode

# make a commit and push branch
>>> git commit
>>> git pull pub greatcode
>>> git push pub greatcode

# on GitHub, submit pull request
```

GitHub

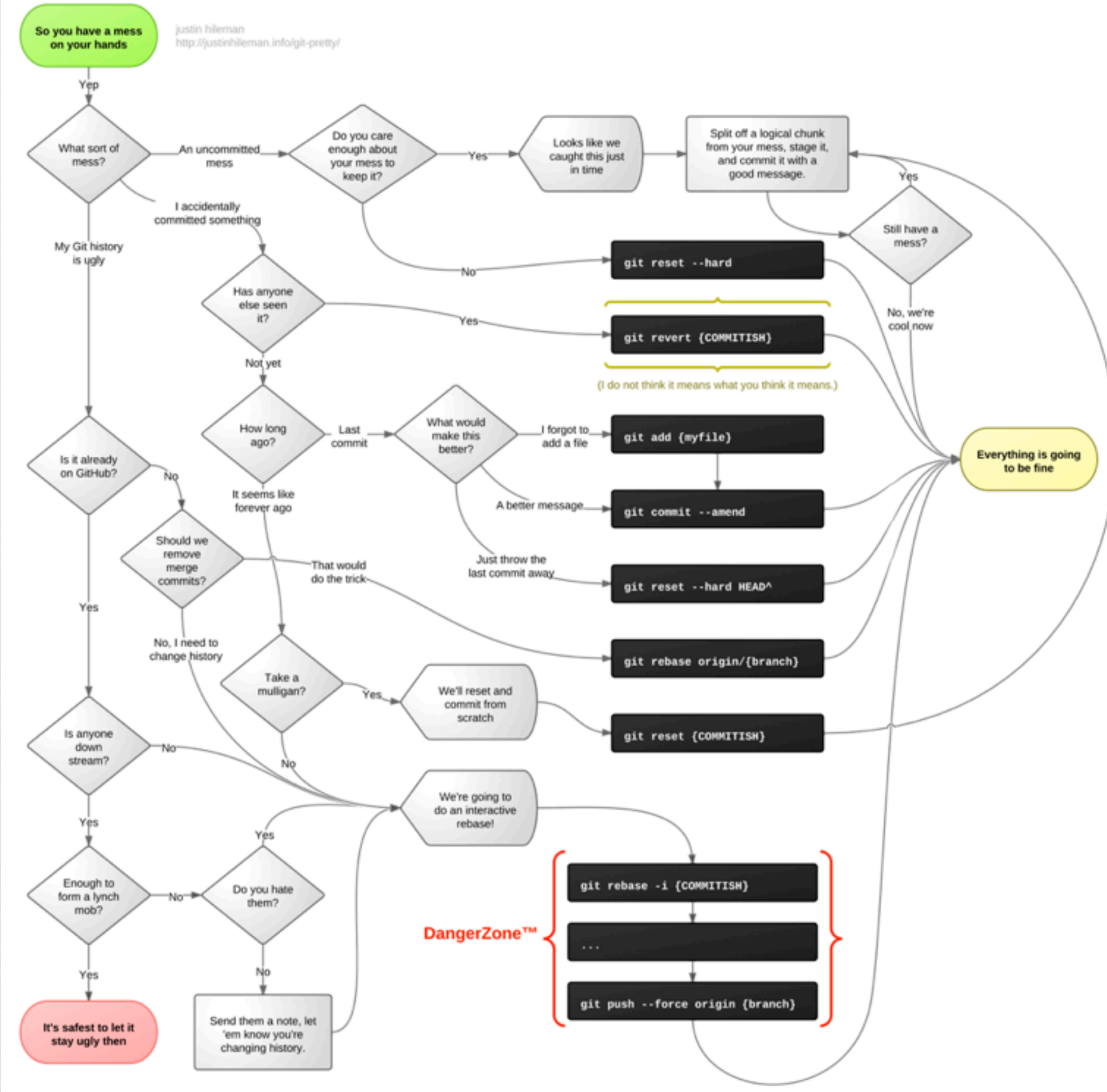
Troubleshooting

ONLY condoned uses of

`git push --force {remote} {branch}`

- your own branch
- a collaborative branch where everyone's checked in current changes and agrees it's time for a rebase

Note that neither covers the master branch



C++/Python Interface: Path of an Input

Statically, C++ & Python are Separate

- Looking at the codebase, C-side and Py-side are separated ...

```
>>> ls lib/python/
aliases.py        driver.py      inpsight.py         p4const     psiexceptions.py  wrappers_cfour.py
frac.py           inputparser.py p4regex.py          pubchem.py  wrappers.py
dashparam.py      functional.py  interactive.py      p4util      qcdb
diatomic_fits.py  gaussian_n.py  interface_cfour.py  pep8.py     qmmm.py
diatomic.py       grendel        molutil.py          proc.py     source.template
```

```
>>> ls src/*
src/bin:
adc         cchbar      CMakeLists.txt  dftsapt             MakeRules.in   occ       scf       transqt2
attic       cclambda    dcft            findif              MakeVars.in    optking   scfgrad
ccdensity   ccresponse  deriv_wrapper   fnocc               mcscf          psi4      stable
ccenergy    ccsort      detci           make_cmake_files.py mints_wrapper  psimrcc   thermo
cceom       cctriples   dfmp2           Makefile.in         mrcc           sapt      transqt

src/lib:
attic                  libciomr  libfock       libmoinfo   libqt           libutil
CMakeLists.libint.txt  libderiv  libfunctional  liboptions  libsapt_solver  make_cmake_files.py
CMakeLists.txt         libdiis   libint        libparallel libscf_solver   Makefile.in
lib3index              libdisp   libiwl        libplugin   libthce         MakeRules.in
libchkpt               libdpd    libmints      libpsio     libtrans        MakeVars.in
```

- ... with a hint of interface definition at Boost Python

```
>>> ls src/bin/psi4/python.cc src/bin/psi4/export*
src/bin/psi4/export_benchmarks.cc   src/bin/psi4/export_functional.cc  src/bin/psi4/export_plugins.cc
src/bin/psi4/export_blas_lapack.cc  src/bin/psi4/export_mints.cc       src/bin/psi4/export_psio.cc
src/bin/psi4/export_chkpt.cc        src/bin/psi4/export_oeprop.cc      src/bin/psi4/python.cc
```

C++/Python Interface: Path of an Input

Boost links existing C++ fns/cls with names & Python types

- psi4 Boost Python module created in macro (python.cc), slurps up

```
>>> vi python.cc
   BOOST_PYTHON_MODULE(psi4)
   { ...
      def("set_nthread", &py_psi_set_n_threads, "Sets the number of threads in SMP parallel computations.");
      def("dfmp2", py_psi_dfmp2, "Runs the DF-MP2 code.");
      export_mints();
   ... }
```

- handy functions defined in python.cc

```
>>> vi python.cc
   void py_psi_set_n_threads(int nthread) {
      Process::environment.set_n_threads(nthread);
   }
```

- wrappers to call whole modules in python.cc

```
>>> vi python.cc
 double py_psi_dfmp2() {
     py_psi_prepare_options_for_module("DFMP2");
     if (dfmp2::dfmp2(Process::environment.options) == Success) {
         return Process::environment.globals["CURRENT ENERGY"];
     } else
         return 0.0; }
```

- whole classes exported other directories (export_*.cc)

```
>>> vi export_mints.cc

    class_<Matrix, SharedMatrix>("Matrix", "docstring").
              def("print_out", &Matrix::print_out, "docstring").
              def("rows", &Matrix::rowdim, "docstring").
               ...
              def("trace", &Matrix::trace, "docstring");
```

C++/Python Interface: Path of an Input

User calls C++, C++ calls Python

```
>>> cd src/bin/psi4
```

- Upon invoking the psi4 binary, the only `main {...}` function in psi4 (`psi4.cc`) runs
- calls `psi_start {...}` (`psi_start.cc`) to handle command line options (*e.g.*, `psi4 -d`) and infile/outfile default names
- initializes IO library, timers, options
- calls `Script::language->run(infile);` (`python.cc`) with the string of the input file.

```
>>> vi python.cc
```

  - to import the Boost Python module

```
s = strdup("import psi4");
PyRun_SimpleString(s);
```

  - to process Psithon into proper Python, equiv to `python = inputparser.process_input(psithon)`

```
// Process the input file
PyObject *input;
PY_TRY(input, PyImport_ImportModule("inputparser") );
PyObject *function;
PY_TRY(function, PyObject_GetAttrString(input, "process_input"));
PyObject *pargs;
PY_TRY(pargs, Py_BuildValue("(s)", file.str().c_str()) );
PyObject *ret;
PY_TRY( ret, PyEval_CallObject(function, pargs) );
char *val;
PyArg_Parse(ret, "s", &val);

inputfile = val;
```

  - to execute the resulting Python commands

```
str strStartScript(inputfile);
object objectScriptInit = exec( strStartScript, objectDict, objectDict );
```

Psithon to Python

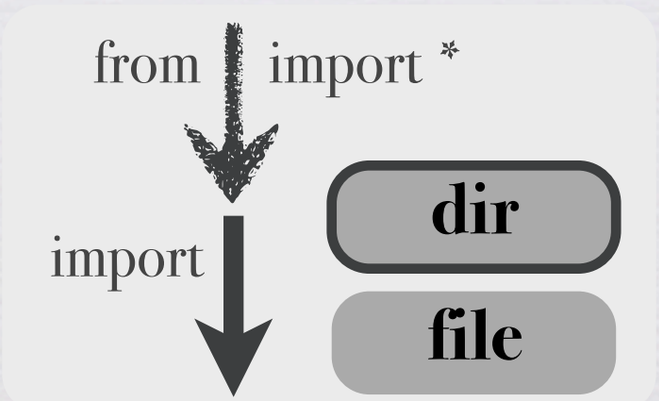
```
python =
inputparser.process_input(psithon)
```

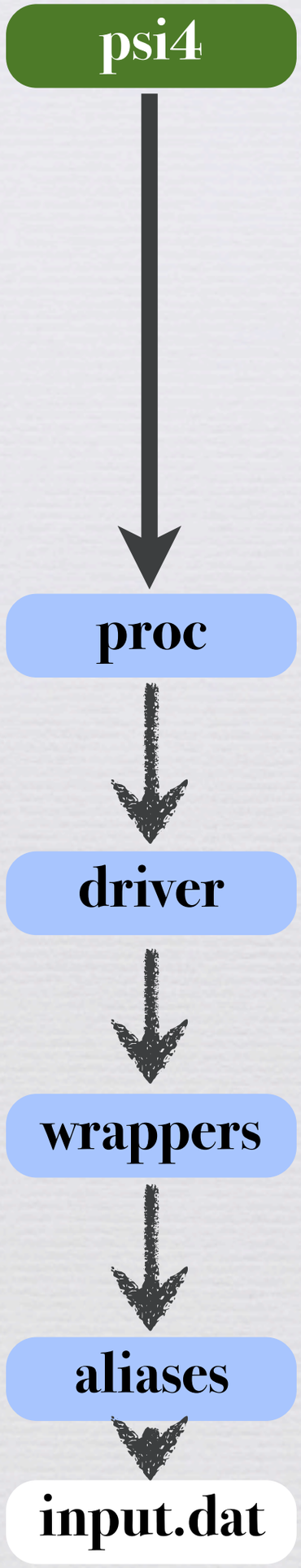
```
#! Example SAPT computation for ethene*ethine,
#! test case 16 from the S22 database

memory 250 mb

Eref = [ 85.189064,  -0.001672, -0.002235 ] #TEST

molecule dimer {
0 1
C   0.000000  -0.667578  -2.124659
C   0.000000   0.667578  -2.124659
H   0.923621  -1.232253  -2.126185
H  -0.923621  -1.232253  -2.126185
H  -0.923621   1.232253  -2.126185
H   0.923621   1.232253  -2.126185
--
0 1
C   0.000000   0.000000   2.900503
C   0.000000   0.000000   1.693240
H   0.000000   0.000000   0.627352
H   0.000000   0.000000   3.963929
units angstrom
}

set globals {
    BASIS jun-cc-pVDZ
    SCF_TYPE DF
    FREEZE_CORE True
}

energy('dfmp2')

Etot = get_variable("CURRENT ENERGY")              #TEST
Ecorl = get_variable("MP2 CORRELATION ENERGY")   #TEST

#psi4.print_variables() #TEST

compare_values(Eref[0], dimer.nuclear_repulsion_energy(), 9, "NRE") #TEST
compare_values(Eref[1], Etot, 6, "MP2 Tot") #TEST
compare_values(Eref[2], Ecorl, 6, "MP2 Corl")   #TEST
```

psi4 -v

(to get processed input in outfile)

```
from psi4 import *
from p4const import *
from p4util import *
from molutil import *
from aliases import *
psi4_io = psi4.IOManager.shared_object()

# <<<  Start .psi4rc  >>>
import os

job_scratch_pbs = os.environ.get('myscratch')
if job_scratch_pbs == None:
   psi4_io.set_default_path('/scratch/loriab/')
else:
   job_scratch_pbs += '/'
   psi4_io.set_default_path(job_scratch_pbs)

# <<<  End .psi4rc  >>>

geometry("""
0 1
H
H 1 0.74
""","blank_molecule_psi4_yo")
#! Example SAPT computation for ethene*ethine
psi4.set_memory(250000000)

Eref = [ 85.189064,  -0.001672, -0.002235 ]
dimer = geometry("""
0 1
C   0.000000  -0.667578  -2.124659
C   0.000000   0.667578  -2.124659
H   0.923621  -1.232253  -2.126185
H  -0.923621  -1.232253  -2.126185
H  -0.923621   1.232253  -2.126185
H   0.923621   1.232253  -2.126185
--
0 1
C   0.000000   0.000000   2.900503
C   0.000000   0.000000   1.693240
H   0.000000   0.000000   0.627352
H   0.000000   0.000000   3.963929
units angstrom
""","dimer")
psi4.IO.set_default_namespace("dimer")
psi4.set_global_option("BASIS", "jun-cc-pVDZ")
psi4.set_global_option("SCF_TYPE", "DF")
psi4.set_global_option("FREEZE_CORE", "True")
energy('dfmp2')
Etot = get_variable("CURRENT ENERGY")
Ecorl = get_variable("MP2 CORRELATION ENERGY")
compare_values(Eref[0], dimer.nuclear_repulsion_energy(), 9,
compare_values(Eref[1], Etot, 6, "MP2 Tot")
compare_values(Eref[2], Ecorl, 6, "MP2 Corl")
```



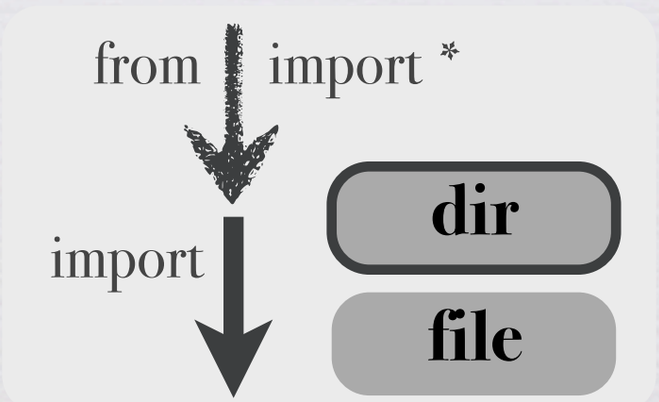


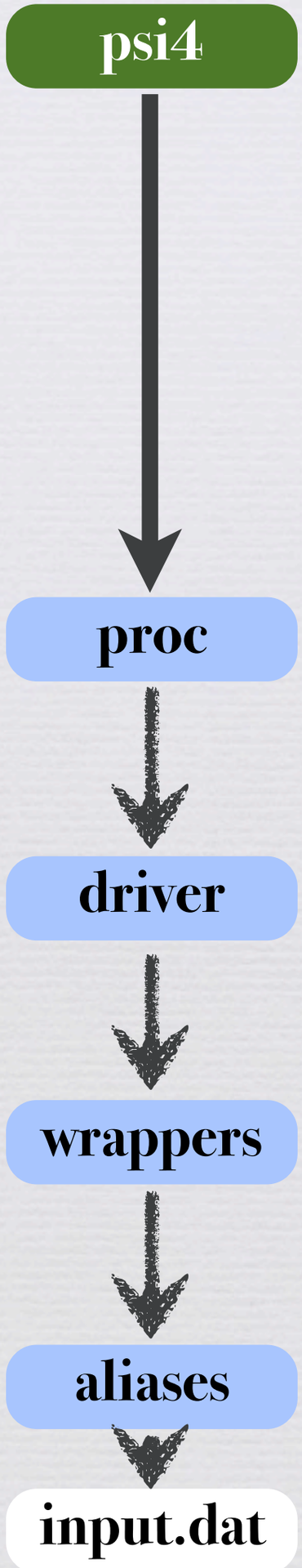
all C-side goodness

psi4

```
mints()
dfmp2()
print_out()
get_option()
Matrix.diagonalize()
Molecule.set_multiplicity()
DGEMM()
```

Map of the Python



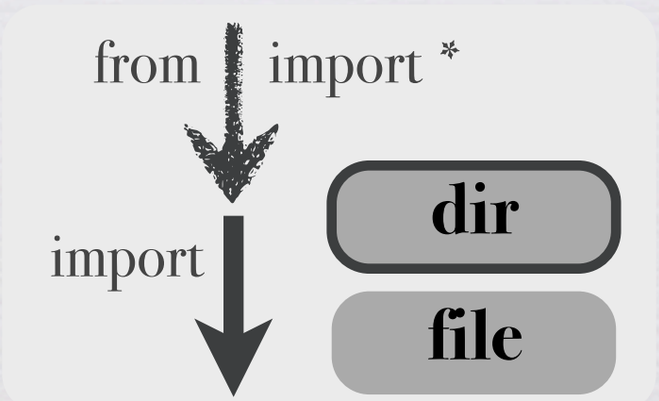


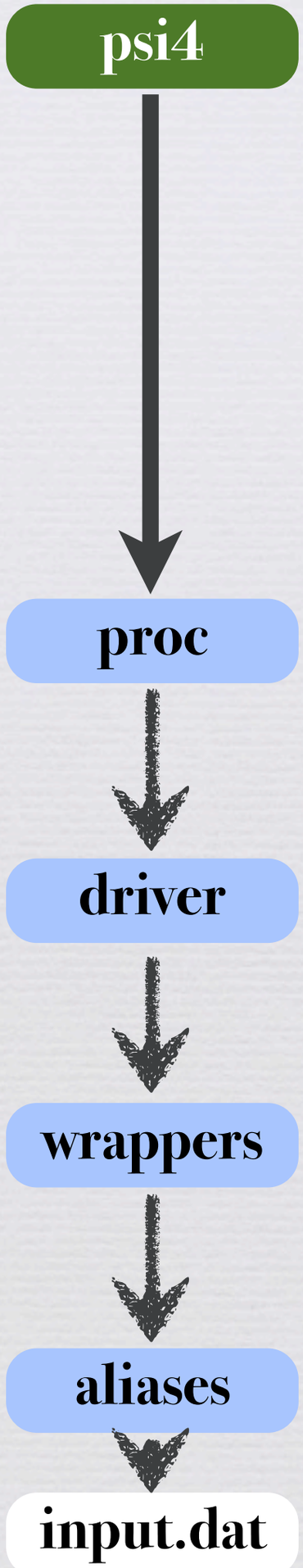
method-specific encoding of C++ module calls

proc

```
scf_helper()
run_dfmp2()
run_dfmp2_gradient()
run_dfmp2_property()
run_ccenergy()
```

Map of the Python



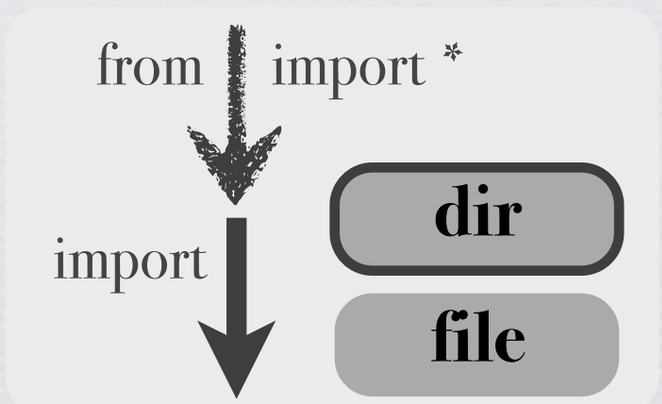


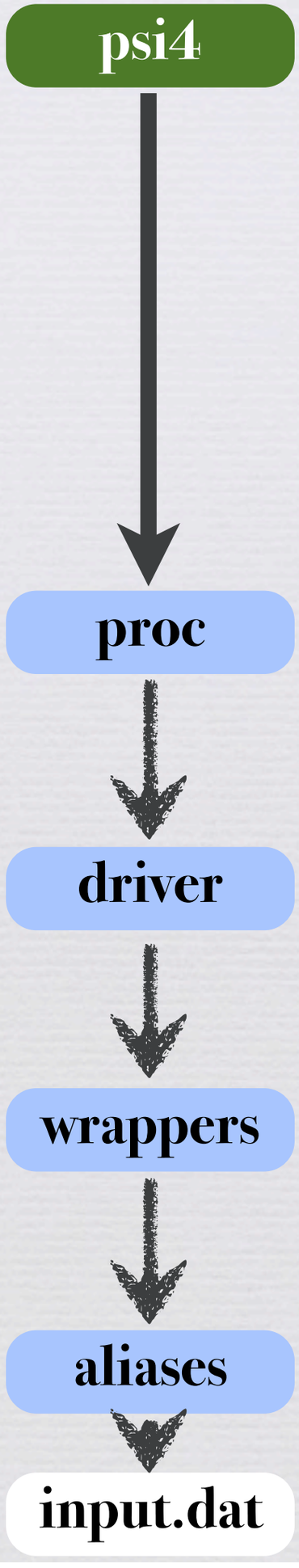
main QC functionality,
method-neutral

driver

```
procedures {}
energy()
gradient()
opt()
hessian()
freq()
property()
```

Map of the Python



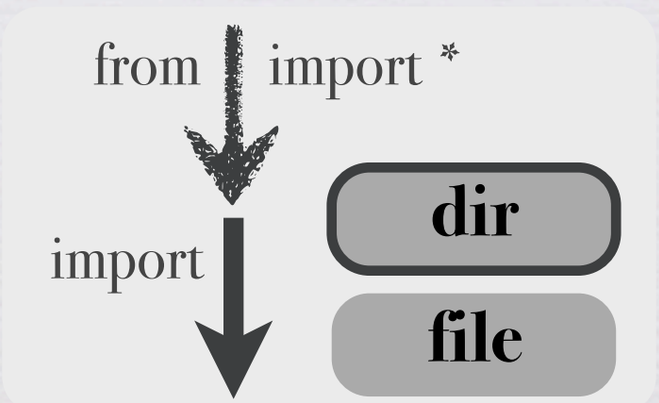


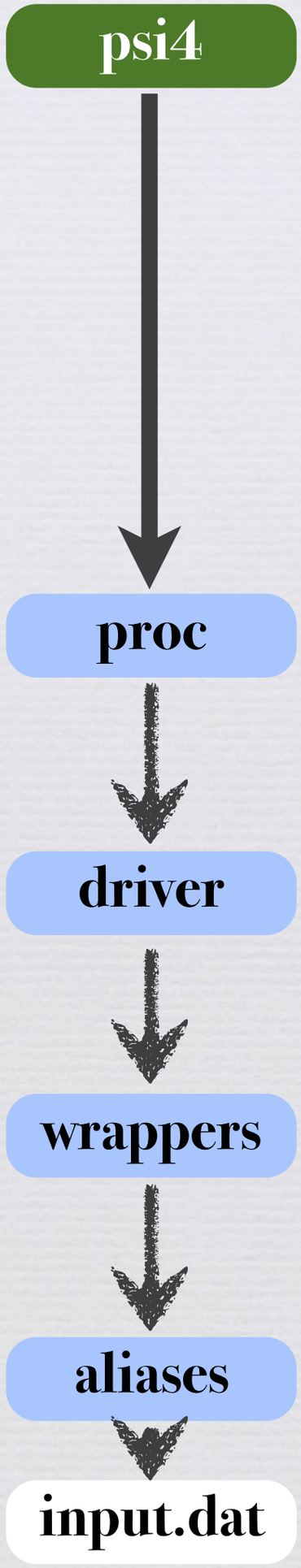
QC functionality involving multiple calls to energy, gradient, etc.

wrappers

```
n_body()
cp()
database()
complete_basis_set()
```

Map of the Python





sandbox for mixing QC methods, python functions

aliases

```
sherrill_gold_standard()
allen_focal_point()
```

C++/Python Interface: Path of an Input

Python calls C++

- Pass through the main driver files, calling Python functions and C++ functions

```
>>> vi lib/python/driver.py
procedures = {
      'energy': {
            'scf'            : run_scf,
            'dfmp2'          : run_dfmp2, }
      'gradient': {
            'scf'            : run_scf, } }
 def energy(name, **kwargs):
    r"""Function to compute the single-point electronic energy."""
    try:
        procedures['energy'][lowername](lowername, **kwargs)
    except KeyError:
        raise ValidationError('Energy method %s not available.' % (name))
    return psi4.get_variable('CURRENT ENERGY')
>>> vi lib/python/proc.py
def run_dfmp2(name, **kwargs):
    """Function encoding sequence of PSI module calls for density-fitted MP2"""
    psi4.scf()
    psi4.dfmp2()
```

- Returns control to `main {...}` (`src/bin/psi4/psi4.cc`), which proceeds to shut down timers, clean scratch, call for beers, etc.

C++/Python Interface Bidirectional

Mix Python & C++ Code in Classes

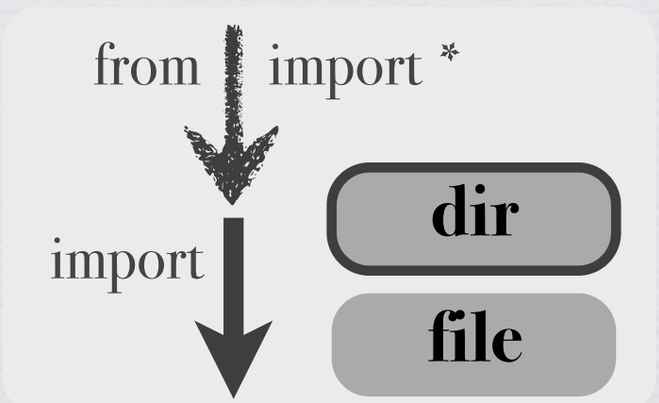
- Extend C++ classes Py-side
  - Here (`lib/python/interface_dftd3.py`), compute -D as member function of `psi4.Molecule`

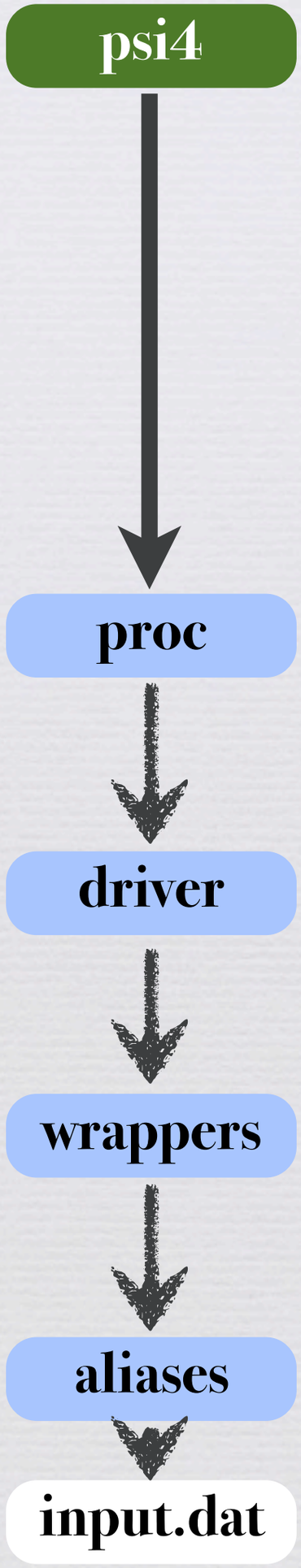
```
def run_dftd3(self, func=None, dashlvl=None, dashparam=None, dertype=None):
    """Function to call Grimme's dftd3 program (http://toc.uni-muenster.de/DFTD3/)
    """
    numAtoms = self.natom()
    geom = self.save_string_xyz()
    psi4.set_variable('DISPERSION CORRECTION ENERGY', dashd)
    return dashd, dashdderiv

psi4.Molecule.run_dftd3 = run_dftd3
```

- Call Python functions C-side
  - Here (`src/lib/libdisp/dispersion.cc`), run dftd3 prog., store -D result in Dispersion class

```
PyObject *molutil;
PY_TRY(molutil, PyImport_ImportModule("interface_dftd3") );
PyObject *grimme;
PY_TRY(grimme, PyObject_GetAttrString(molutil, "run_dftd3"));
PyObject *pargs;
PY_TRY(pargs, Py_BuildValue("(s s s {s:f,s:f} i)", NULL, NULL, "d2gr", "s6", s6_, "alpha6", d_, 0));
PyObject *ret;
PY_TRY(ret, PyEval_CallObject(grimme, pargs));
E = boost::python::extract<double>(ret);
```

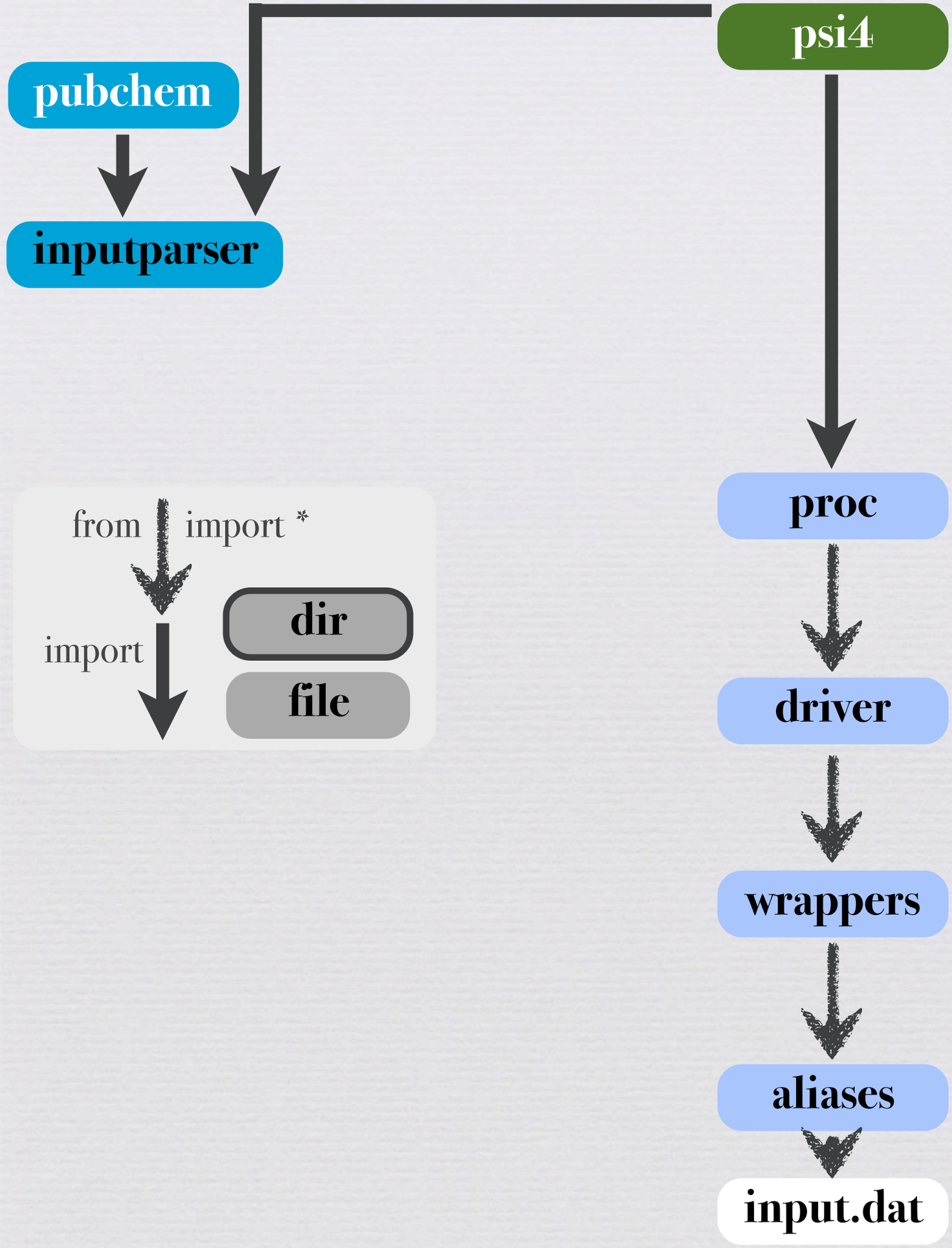
Map of the Python





Map of the Python

Idealized



psi4

pubchem

inputparser

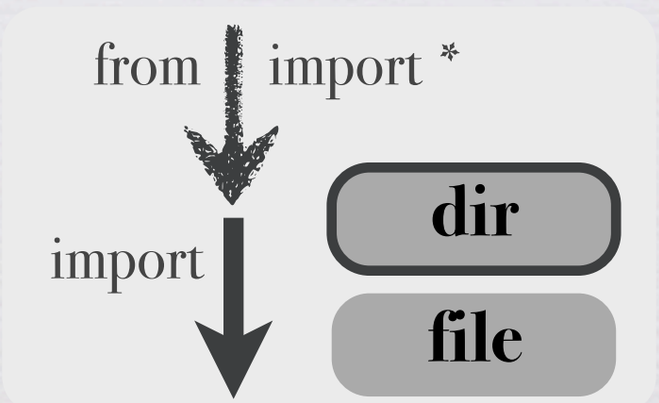
p4xcpt

p4const

p4regex

p4util

molutil



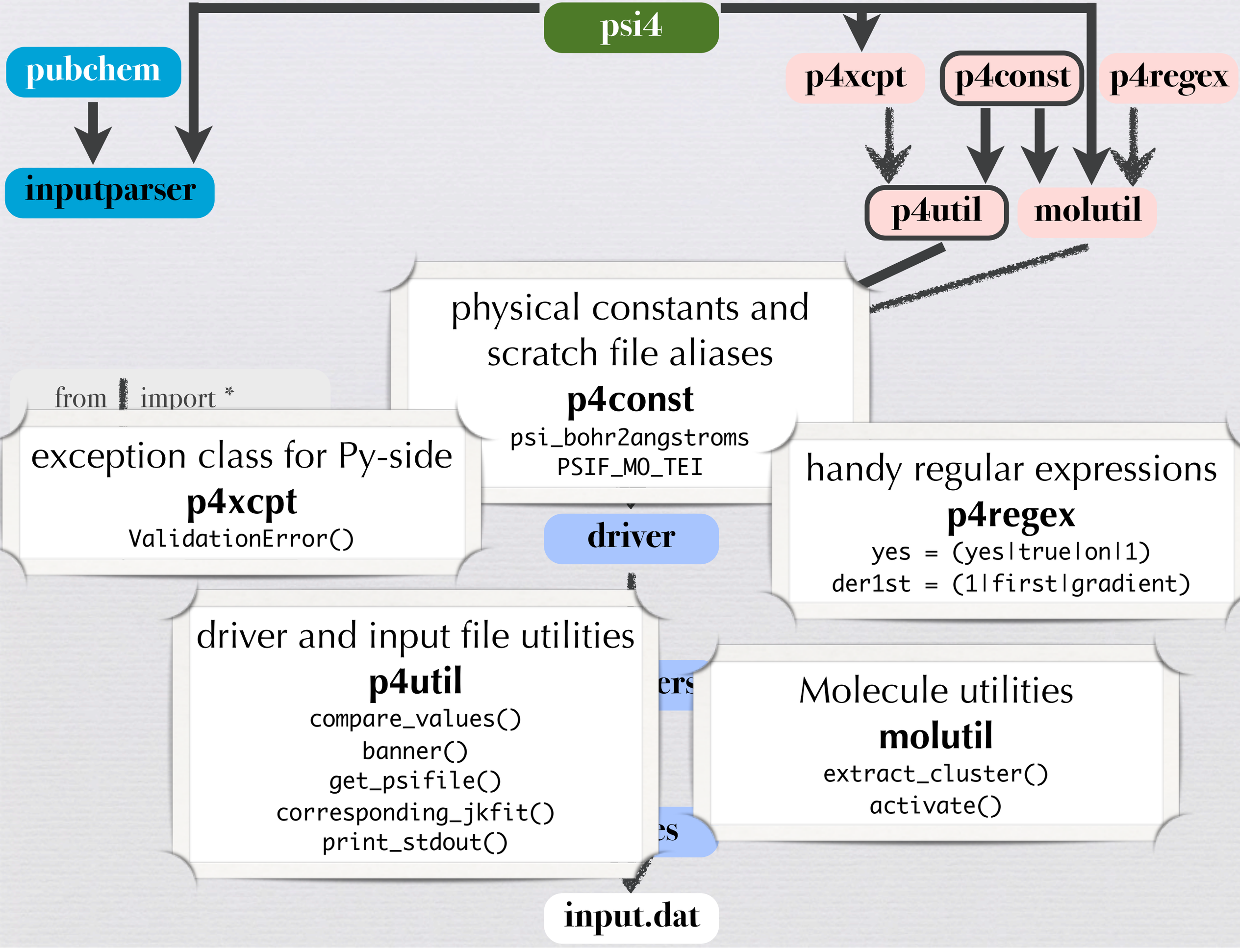
proc

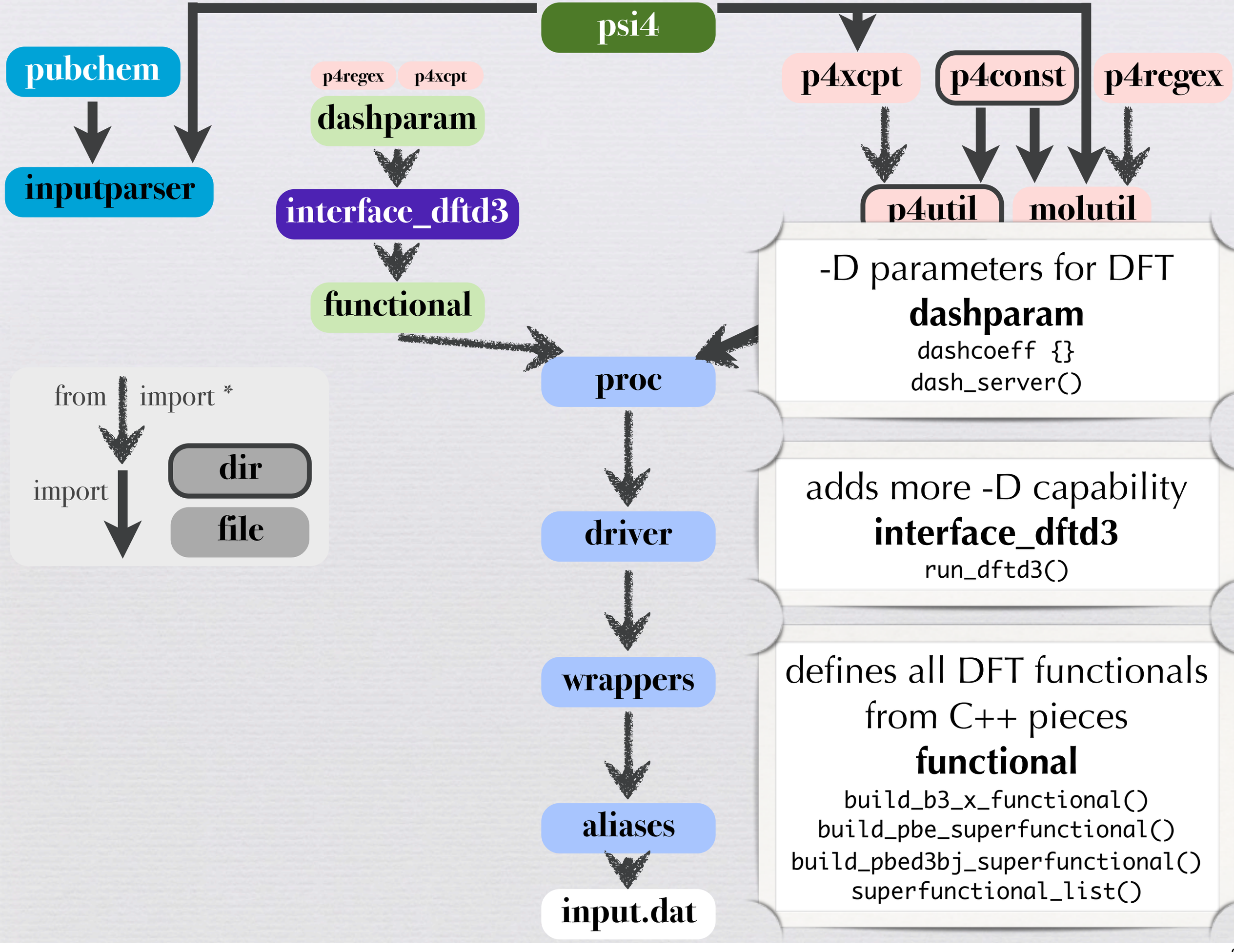
driver

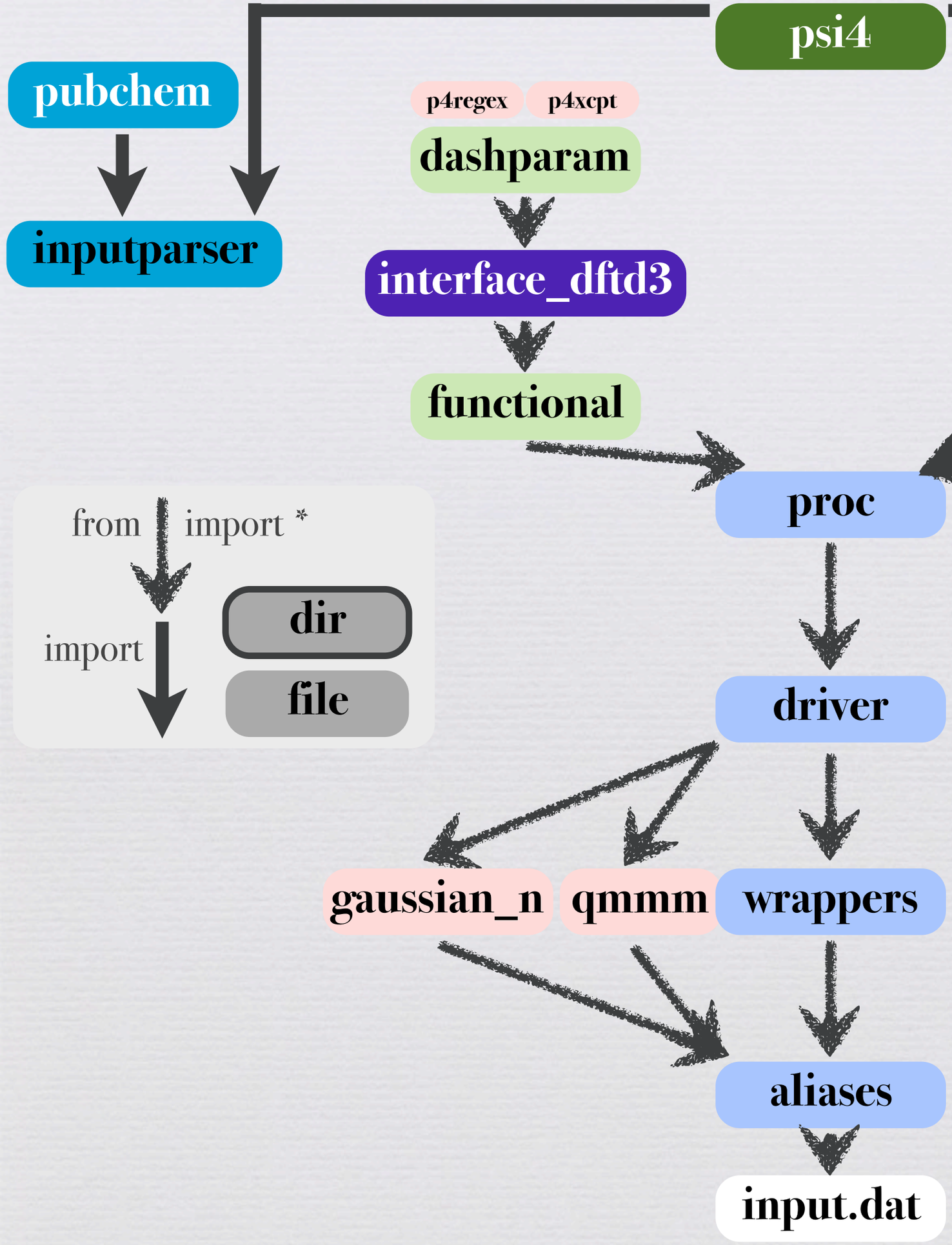
wrappers

aliases

input.dat


psi4
pubchem
inputparser
p4xcpt
p4const
p4regex
p4util
molutil
physical constants and scratch file aliases
p4const
psi_bohr2angstroms
PSIF_MO_TEI
handy regular expressions
p4regex
yes = (yes|true|on|1)
der1st = (1|first|gradient)
from import *
exception class for Py-side
p4xcpt
ValidationError()
driver
driver and input file utilities
p4util
compare_values()
banner()
get_psifile()
corresponding_jkfit()
print_stdout()
Molecule utilities
molutil
extract_cluster()
activate()
input.dat

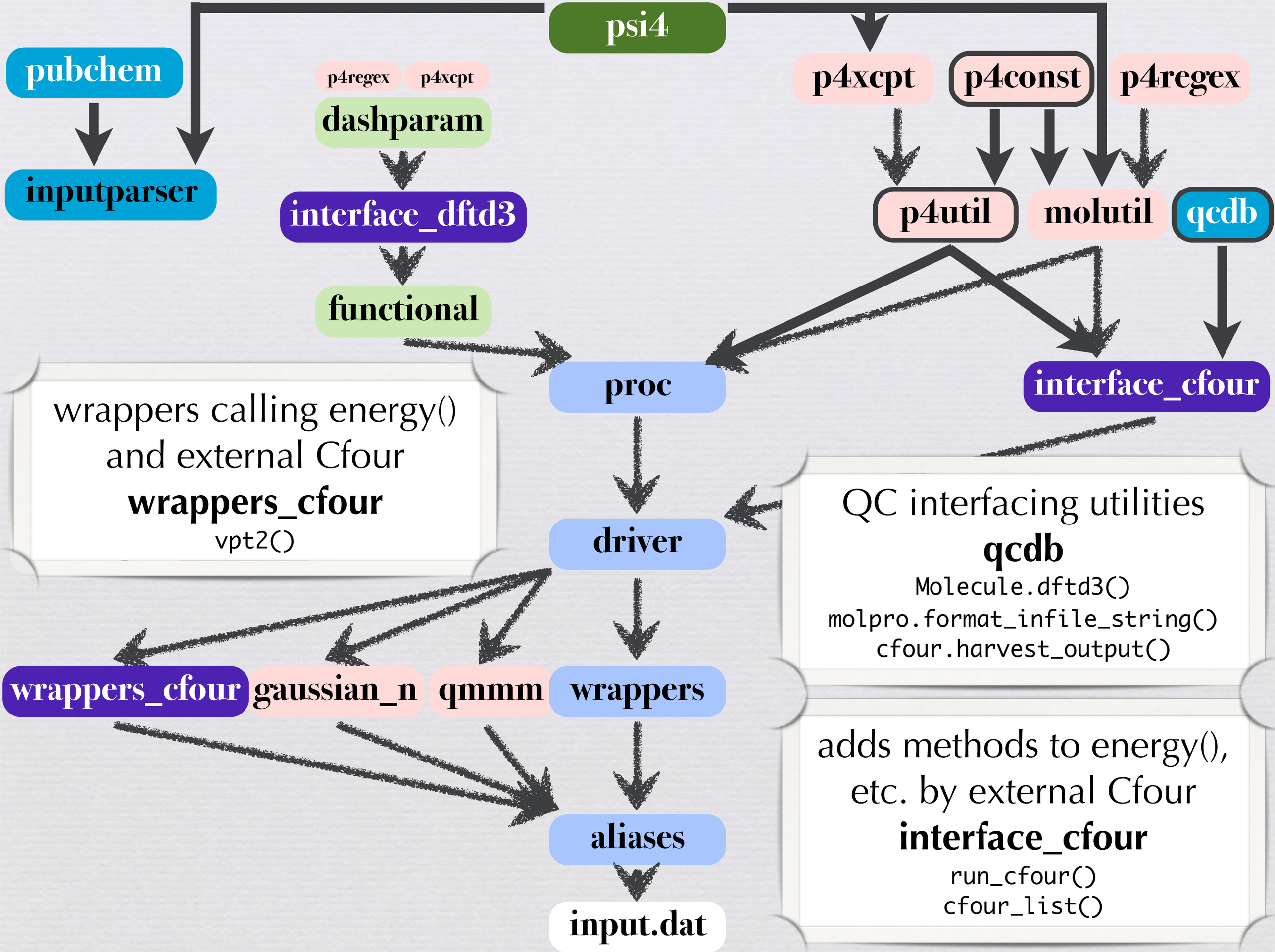

psi4
pubchem
inputparser
p4regex
p4xcpt
dashparam
interface_dftd3
functional
p4xcpt
p4const
p4regex
p4util
molutil
-D parameters for DFT
dashparam
dashcoeff {}
dash_server()
proc
from
import *
import
dir
file
driver
wrappers
aliases
input.dat
adds more -D capability
interface_dftd3
run_dftd3()
defines all DFT functionals
from C++ pieces
functional
build_b3_x_functional()
build_pbe_superfunctional()
build_pbed3bj_superfunctional()
superfunctional_list()



wrappers and classes calling energy(), etc.

gaussian_n / qmmm

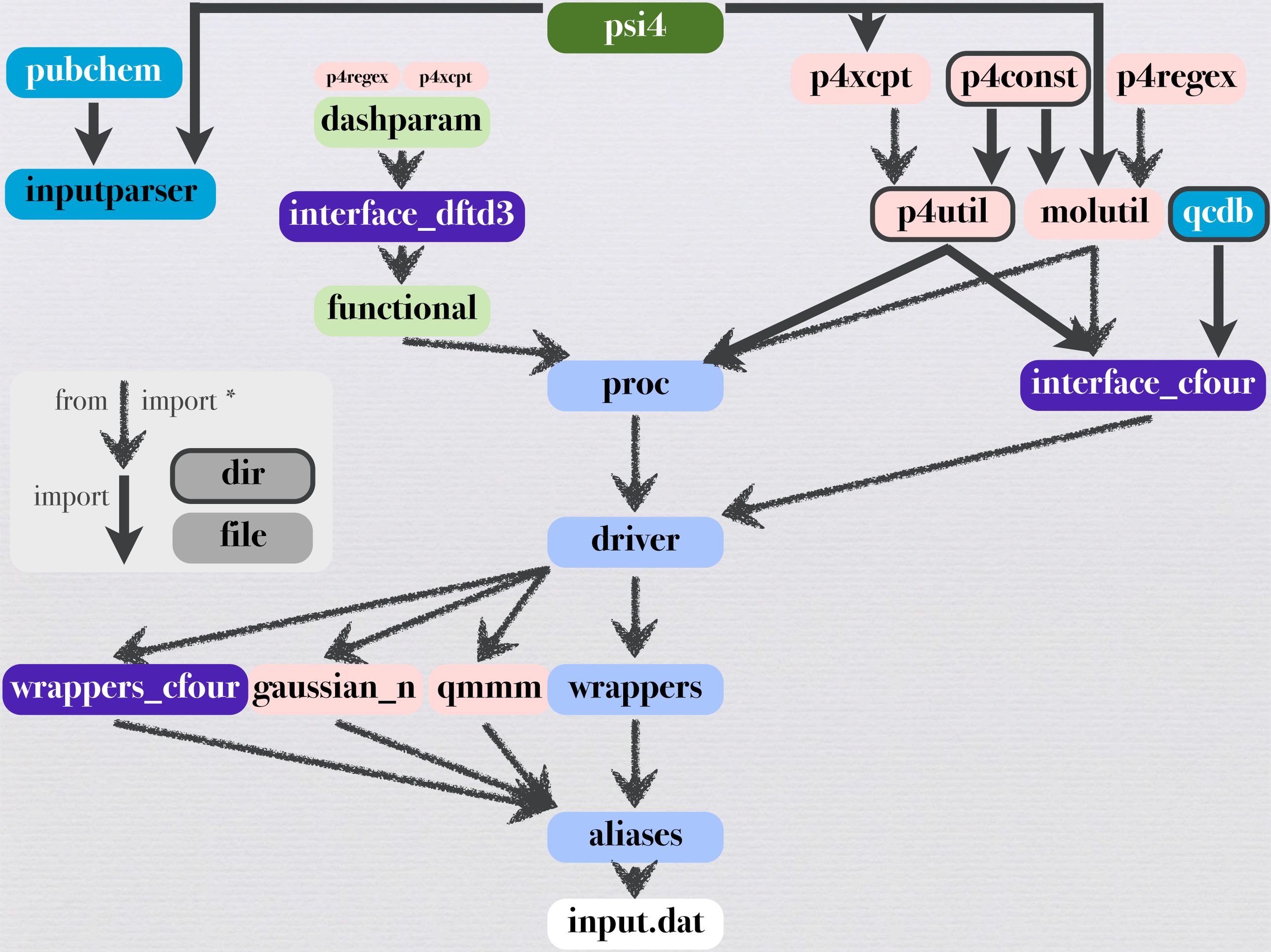
```
run_gaussian_2()
QMMM.addDiffuse()
```

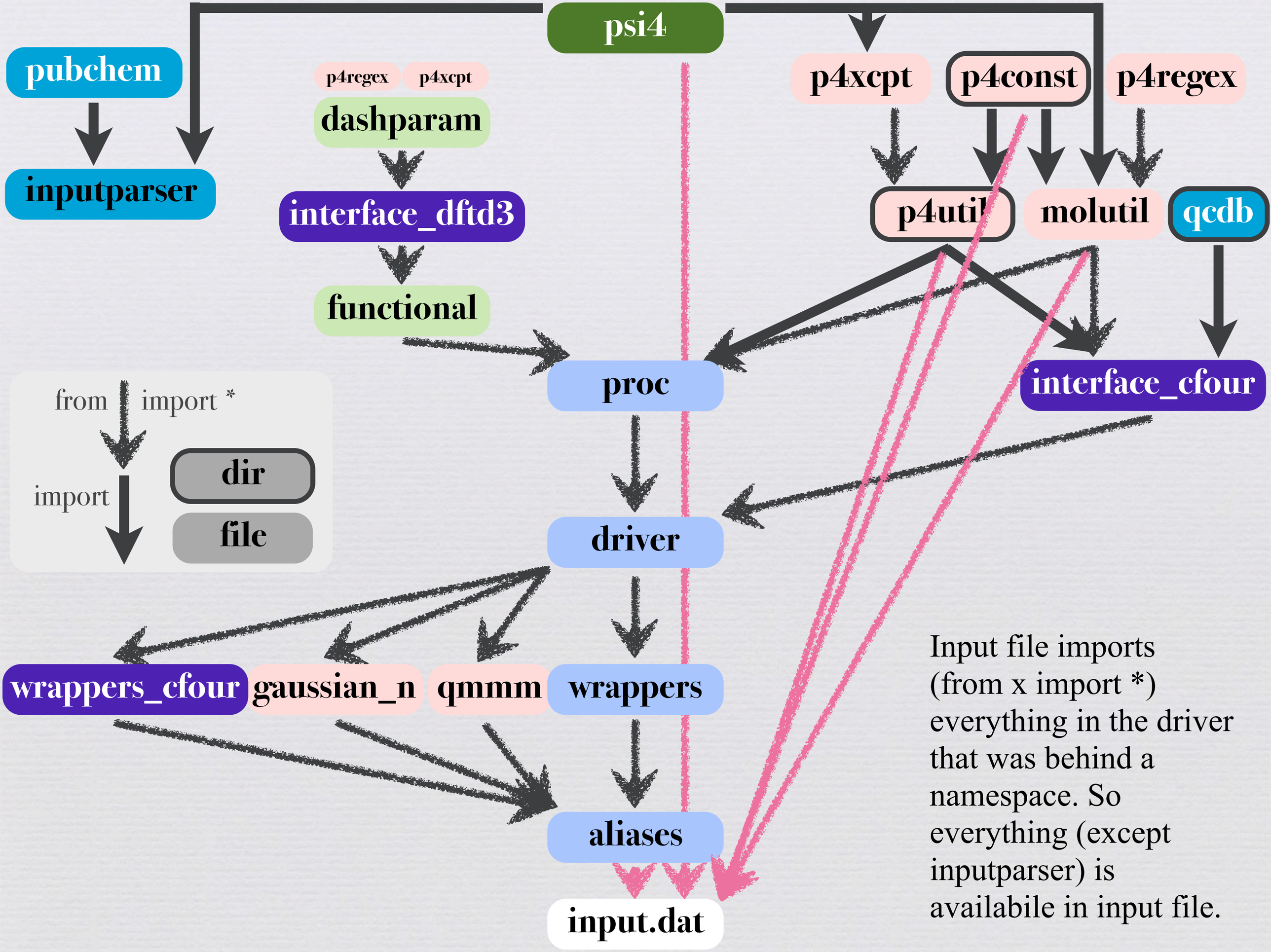

psi4
pubchem
inputparser
p4regex
p4xcpt
dashparam
interface_dftd3
functional
p4xcpt
p4const
p4regex
p4util
molutil
qcdb
proc
interface_cfour
wrappers calling energy() and external Cfour
wrappers_cfour
vpt2()
driver
QC interfacing utilities
qcdb
Molecule.dftd3()
molpro.format_infile_string()
cfour.harvest_output()
wrappers_cfour
gaussian_n
qmmm
wrappers
adds methods to energy(), etc. by external Cfour
interface_cfour
run_cfour()
cfour_list()
aliases
input.dat

Py-Side

Planning and Importing

- Free to make new python files or module directories, just put them in the right place: utility, procedure, wrapper, etc.
- Use the map to make minimal imports, and choose import vs. from x import * appropriately.
- From non-psi4 python, import whatever modules you want. use import mod or from mod import fun1, fun2, not from mod import *
- If you make the arrows go both ways, that's called a circular import. If you do that, what you're working on may work, but it will break other tasks.
- Additions should work in Python 2.6, 2.7, & 3.
- Method aliases are discouraged. If you do add them, search the whole driver for logic.


psi4
pubchem
inputparser
p4regex
p4xcpt
dashparam
interface_dftd3
functional
p4xcpt
p4const
p4regex
p4util
molutil
qcdb
proc
interface_cfour
from
import *
import
dir
file
driver
wrappers_cfour
gaussian_n
qmmm
wrappers
aliases
input.dat



Input file imports (from x import *) everything in the driver that was behind a namespace. So everything (except inputparser) is availabile in input file.

proc.py
Boilerplate

- Thy code shall not print to screen in its final version (except perhaps for the master calc in sow/reap jobs).
- Thy code shall raise ValidationError not sys.exit(). This adds error to traceback to screen and also to the output file, so jobs run in queue show why they're quitting.

```
if molecule.nfragments() != 2:
    raise ValidationError('SAPT requires active molecule to have 2 fragments, not %s.' % (nfrag))
```

- Thy code shall leave no files in the submission directory that alter other jobs or repeats of the same job (I'm thinking of you, `intco.dat`). That is, psi4 shall be idempotent.
- Thy code shall not demand case-sensitive input (mostly relevant Py-side as C-side insulated).

```
def function_that_couple_be_called_from_input_file(name, **kwargs):
    lowername = name.lower()
    kwargs = p4util.kwargs_lower(kwargs)
```

- Thy code shall make judicious selections of defaults. Convergences shall be on the loose side of production quality from a serious comp chem group. Once the user has declared his model chemistry, calculation type, etc., go ahead and make best practices adjustments to options based on those parameters. A simple input file should be sufficient to run a good calculation.

- Thy code shall return control to the user in the same state it took control from the user, though updated by results. That is, after an `optimize('method')` line, the active molecule may have an updated geometry and `CURRENT ENERGY` an updated value, but options should have the same values before and after, and the symmetry of the molecule shouldn't have changed.

```
optstash = p4util.OptionsState(                 # save the current values and has_changed status
    ['DF_BASIS_SCF'],                           # basis sets and global options (and puream) have one arg
    ['SCF', 'SCF_TYPE'])                        # most options have two args, module and option name
...                                             # freely change option value, call other functions, etc.
optstash.restore()                              # restore saved values before end of python function
```

- Thy code shall set fitting basis sets if needed and possible.

```
if (psi4.get_option('SCF', 'SCF_TYPE') == 'DF'):
    # if the df_basis_scf basis is not set, pick a sensible one.
    if psi4.get_global_option('DF_BASIS_SCF') == '':
        jkbasis = p4util.corresponding_jkfit(psi4.get_global_option('BASIS'))
        if jkbasis:
            psi4.set_global_option('DF_BASIS_SCF', jkbasis)
            psi4.print_out('\n  No DF_BASIS_SCF auxiliary basis selected, defaulting to %s\n\n' % (jkbasis))
        else:
            raise ValidationError('Keyword DF_BASIS_SCF is required.')
```

- Thy code shall trap forseeable invalid jobs. Whether a reference not implemented, missing keyword, wrong number of fragments, no virtual orbitals to excite into, only works with certain scf algorithm, etc., make psi4 exit gracefully. If you only do one of (a) document your code's limitations or (b) trap violations of you code's limitations, do the latter.

- Thy code shall be symmetry-tolerant, if not symmetry-exploiting.

```
molecule = psi4.get_active_molecule()
user_pg = molecule.schoenflies_symbol()
molecule.reset_point_group('c1')              # equiv to molecule {... symmetry c1} in input
molecule.fix_orientation(True)                # equiv to molecule {... no_reorient} in input
molecule.fix_com(True)                        # equiv to molecule {... no_com} in input
molecule.update_geometry()
if user_pg != 'c1':
    psi4.print_out('  SAPT does not make use of molecular symmetry, further calculations in C1 point group.\n')
...
molecule.reset_point_group(user_pg)
molecule.update_geometry()
```

- Thy code shall not re-invent SCF convergence tricks.
  - Use `scf_helper()` . Basis set casting, symmetry breaking, DF basis setting, it's got it all.
  - If user has converged a tricky SCF before calling `energy('method', bypass_scf=true)`, don't clobber it.
  - If your method expects integrals that may not be there, compute them.

```
# Bypass routine scf if user did something special to get it to converge
if not (('bypass_scf' in kwargs) and yes.match(str(kwargs['bypass_scf']))):
    scf_helper(name, **kwargs)
    # If the scf type is DF/CD, then the AO integrals were never written to disk
    if psi4.get_option('SCF', 'SCF_TYPE') == 'DF' or psi4.get_option('SCF', 'SCF_TYPE') == 'CD':
        mints = psi4.MintsHelper()
        mints.integrals()
psi4.transqt2()
psi4.detci()
```

Example Python Extension

MP2.5 in aliases.py

```
def run_mp2_5(name, **kwargs):

    energy('mp3', **kwargs)
    e_scf = psi4.get_variable('SCF TOTAL ENERGY')
    ce_mp2 = psi4.get_variable('MP2 CORRELATION ENERGY')
    ce_mp3 = psi4.get_variable('MP3 CORRELATION ENERGY')

    ce_mp25 = 0.5 * (ce_mp2 + ce_mp3)
    e_mp25 = e_scf + ce_mp25

    print """  MP2.5 total energy:                %16.8f\n""" % (e_mp25)
    print """  MP2.5 correlation energy:          %16.8f\n""" % (ce_mp25)

    return e_mp25
```

One final step is necessary. At the end of the `aliases.py` file, add the following line.

```
procedures['energy']['mp2.5'] = run_mp2_5
```

This permits the newly defined MP2.5 method to be called in the input file with the following command.

```
energy('mp2.5')
```

Example Python Extension

MP2.5 in aliases.py

```
def run_mp2_5(name, **kwargs):

    energy('mp3', **kwargs)
    e_scf = psi4.get_variable('SCF TOTAL ENERGY')
    ce_mp2 = psi4.get_variable('MP2 CORRELATION ENERGY')
    ce_mp3 = psi4.get_variable('MP3 CORRELATION ENERGY')

    ce_mp25 = 0.5 * (ce_mp2 + ce_mp3)
    e_mp25 = e_scf + ce_mp25

    print """  MP2.5 total energy:
    print """  MP2.5 correlation energy

    return e_mp25
```

One final step is necessary. At the en

```
procedures['energy']['mp2.5'] =
```

This permits the newly defined MP2.5

```
energy('mp2.5')
```

```
def run_mp2_5(name, **kwargs):
    lowername = name.lower()  # handy variable with name keyword in lowercase
    kwargs = kwargs_lower(kwargs)  # removes case sensitivity in keyword names

    # Run detci calculation and collect conventional quantities
    energy('mp3', **kwargs)
    e_scf = psi4.get_variable('SCF TOTAL ENERGY')
    ce_mp2 = psi4.get_variable('MP2 CORRELATION ENERGY')
    ce_mp3 = psi4.get_variable('MP3 CORRELATION ENERGY')
    e_mp2 = e_scf + ce_mp2  # reform mp2 and mp3 total energies for printing
    e_mp3 = e_scf + ce_mp3

    # Compute quantities particular to MP2.5
    ce_mp25 = 0.5 * (ce_mp2 + ce_mp3)
    e_mp25 = e_scf + ce_mp25
    psi4.set_variable('MP2.5 CORRELATION ENERGY', ce_mp25)  # add new method's important results
    psi4.set_variable('MP2.5 TOTAL ENERGY', e_mp25)         #     to PSI variable repository
    psi4.set_variable('CURRENT CORRELATION ENERGY', ce_mp25)
    psi4.set_variable('CURRENT ENERGY', e_mp25)  # geometry optimizer tracks this variable, permits
                                                 #     MP2.5 finite difference optimizations
    # build string of title banner and print results
    banners = ''
    banners += """psi4.print_out('\\n')\n"""
    banners += """banner(' MP2.5 ')\n"""
    banners += """psi4.print_out('\\n')\n\n"""
    exec banners

    tables  = ''
    tables += """  SCF total energy:                %16.8f\n""" % (e_scf)
    tables += """  MP2 total energy:                %16.8f\n""" % (e_mp2)
    tables += """  MP2.5 total energy:              %16.8f\n""" % (e_mp25)
    tables += """  MP3 total energy:                %16.8f\n\n""" % (e_mp3)
    tables += """  MP2 correlation energy:           %16.8f\n""" % (ce_mp2)
    tables += """  MP2.5 correlation energy:         %16.8f\n""" % (ce_mp25)
    tables += """  MP3 correlation energy:           %16.8f\n""" % (ce_mp3)
    psi4.print_out(tables)  # prints nice header and table of all involved quantities to output file

    return e_mp25
```

Keywords

Declaring

- Keywords control the workings of Psi4, including all C++ modules and most all Python functions (only exceptions are kwargs to Python functions).
- Naming: underscore-connected, all caps
- All keywords (outside of plugins) live in `src/bin/psi4/read_options.cc` . Therein, each module has a section, each section has one or more partitions by topic, each partition one or more options, and each option a docstring, a name, a type, and a default. Modules, sections, partitions, options, how many keywords does Psi4 support?
- Declaring C-side:

  integer: `options.add_int("TILE_SZ", 512);`

  boolean: `options.add_bool("MOLDEN_WRITE", false);`

  double: `options.add_double("S_TOLERANCE", 1E-7);`

  string:
  ```
  options.add_str("GUESS", "CORE", "CORE GWH SAD READ");  # default
  options.add_str("DF_BASIS_SCF", "");  # no default
  ```

  array: `options.add("RESTRICTED_DOCC", new ArrayType());  # int/double not declared here`
- Declaring Py-side: t'ain't done. Keywords live C-side only.

  `Process::environment.globals["MP2 CORRELATION ENERGY"] = emp2_os + emp2_ss;`

- In choosing a keyword name, look for analogous functionality in other modules and use same name. Follow best practices guidelines.

```
if (name == "SCF"|| options.read_globals()) {
    /*- MODULEDESCRIPTION Performs self consistent field (Hc
    Density Functional Theory) computations. -*/

    /*- SUBSECTION General Wavefunction Info -*/
    /*- Wavefunction type !expert -*/
    options.add_str("WFN", "SCF", "SCF");
    /*- Tolerance for Cholesky decomposition of the ERI tens
    options.add_double("CHOLESKY_TOLERANCE",1e-4);

    /*- SUBSECTION Convergence Control/Stabilization -*/
    /*- Maximum number of iterations.
    **Cfour Interface:** Keyword translates into |cfour__cfour_scf_maxcyc|. -*/
    options.add_int("MAXITER", 100);
    /*- Fail if we reach maxiter without converging? -*/
    options.add_bool("FAIL_ON_MAXITER",true);
     /*- Convergence criterion for SCF energy. See Table :re
     Convergence & Algorithm <table:conv_scf>` for default c
     criteria for different calculation types. -*/
     options.add_double("E_CONVERGENCE", 1e-6);
     options.add("FRAC_OCC", new ArrayType());
     /*- The occupations of the orbital indices specified above ($0.0\ge occ \ge 1.0$) -*/
```

Table Of Contents

- SCF
  - General Wavefunction Info
    - BASIS
    - CHOLESKY_TOLERANCE
    - DF_BASIS_SCF
    - DF_SCF_GUESS
    - GUESS
    - INDEPENDENT_J_TYPE
    - INTS_TOLERANCE
    - MOLDEN_WRITE
    - PRINT_BASIS
    - PRINT_MOS
    - REFERENCE
    - SAVE_JK
    - SCF_MEM_SAFETY_FACTOR
    - SCF_TYPE
    - S_ORTHOGONALIZATION
    - S_TOLERANCE
  - Convergence Control/Stabilization
    - BASIS_GUESS
    - DAMPING_CONVERGENCE
    - DAMPING_PERCENTAGE
    - DF_BASIS_GUESS
    - DIIS
    - DIIS_MAX_VECS
    - DIIS_MIN_VECS
    - DIIS_START
    - D_CONVERGENCE
    - E_CONVERGENCE
    - FAIL_ON_MAXITER
    - MAXITER
    - MOM_OCC
    - MOM_START
    - MOM_VIR
    - STABILITY_ANALYSIS
  - Fractional Occupation UHF/UKS
    - FRAC_DIIS
  - *Expert* General Wavefunction Info
    - WFN
  - *Expert* Convergence Control/Stabilization
    - FOLLOW_STEP_SCALE
  - *Expert* Parallel Runtime
    - DISTRIBUTED_MATRIX
    - PARALLEL
    - PROCESS_GRID
    - TILE_SZ
  - *Expert* Misc.
    - SAPT
  - *Expert* DFSCF Algorithm
    - DF_FITTING_CONDITION
    - DF_INTS_IO

Performs self consistent field (Hartree-Fock and Density Functio
so this code is called in most cases.

General Wavefunction Info

BASIS

Primary basis set

- **Type**: string
- **Possible Values**: *basis string*
- **Default**: No Default

CHOLESKY_TOLERANCE

Tolerance for Cholesky decomposition of the ERI tensor

- **Type**: *conv double*
- **Default**: 1e-4

DF_BASIS_SCF

Auxiliary basis set for SCF density fitting computations. De

- **Type**: string
- **Possible Values**: *basis string*
- **Default**: No Default

DF_SCF_GUESS

Use DF integrals tech to converge the SCF before switchir

- **Type**: *boolean*
- **Default**: true

- **Default**: true

FRAC_START

The iteration to start fractionally occupying orbitals (or 0 for

- **Type**: integer
- **Default**: 0

FRAC_VAL

The occupations of the orbital indices specified above (0.0

- **Type**: array
- **Default**: No Default

Keywords

Getting & Setting C-side

- Keywords declared as booleans can be input by the user as true/yes/on/1 or false/no/off/0. By the time you access it from the options class, value will be true or false.
- Keywords declared as doubles with CONV or TOL in their names can be input by the user as 6/1.0e-6/0.000001. By the time you access it from the options class, value will be a double.
- By the time your C++ module receives an Options object, it is specially prepared for the module, containing as keys only the keywords defined for the module and as values the local value if changed, otherwise the global value if changed, otherwise the local default.
- Accessing C-side:

  integer: `Parameters.cc_ex_lvl = options.get_int("CC_EX_LEVEL");`

  boolean: `diis_enabled_ = options_.get_bool("DIIS");`

  double: `Parameters.convergence = options.get_double("R_CONVERGENCE");`

  string: `std::string line1 = options.get_str("DIAG_METHOD");`

  array: can be multidimensional, need to cast `to_integer()` or `to_double()` as appropriate

```
int nirreps = options["DOCC"].size();
for(int h = 0; h < nirreps; ++h)
    doccpi_[h] = options_["DOCC"][h].to_integer();
```

- Setting C-side: this is sometimes done but strongly discouraged. Try to view the Options object as read-only C-side (of course you can overwrite defaults once they're read into local variables, just leave the Options object alone). This restriction is due to a subtle weakness of the Options class that changes C-side cannot be undone (can reset value but not has_changed)

Keywords

Getting & Setting Py-side

- Setting Py-side:
  - Basis sets and keywords at top (outside a module section) of `read_options.cc`:

```
psi4.set_global_option('BASIS', 'AUG-CC-PVDZ')
psi4.set_global_option('PUREAM', psi4.MintsHelper().basisset().has_puream())
```

  - Every other keyword, set with relevant module.

integer: `psi4.set_local_option('DETCI', 'EX_LEVEL', 4)`

boolean: `psi4.set_local_option('FNOCC', 'NAT_ORBS', True)`

double: `psi4.set_local_option('SAPT', 'E_CONVERGENCE', 10e-10)`

string: `psi4.set_local_option('SAPT', 'SAPT_LEVEL', 'SAPT2+3')`

array: `psi4.set_global_option("FROZEN_DOCC", [1, 0, 0, 0, 0, 0, 0, 0])`

- Accessing Py-side:
  - Basis sets and keywords at top (outside a module section) of `read_options.cc`:

```
user_ri_basis = psi4.get_global_option('DF_BASIS_MP2')
user_writer_file_label = psi4.get_global_option('WRITER_FILE_LABEL')
```

  - Every other keyword, query with relevant module. Returns int, float, str as appropriate.

```
df_ints_io = psi4.get_option('SCF', 'DF_INTS_IO')
```

Keywords

Curse of the has_changed()

- When keywords are independent, `read_options.cc` handles defaults nicely, sending the "right" value to the module C-side, whether that value be user-set or default. But sometimes defaults should change depending on the values of other keywords. Analogously Py-side, keywords are reset according to "best practices" for the modelchem and calc type.
- In both these cases, don't want to clobber the user's expressed preferences. Rather, these complex defaults should only be set `if !(option.has_changed())` . There's a mechanism!
- Note that `has_changed()` returns whether the option has been changed upcode of the present. Therefore, a true value indicates change by user or driver.
- C-side: tying a convergence into the "master" `E_CONVERGENCE` value

```
if (options_["MAX_MOGRAD_CONVERGENCE"].has_changed()) {
    mograd_max=options_.get_double("MAX_MOGRAD_CONVERGENCE");
} else {
    double temp2 = -log10(tol_grad) - 1.5;
    if (temp2 > 4.0)
         temp2 = 4.0;
    mograd_max = pow(10.0, -temp2); }
```

- Py-side: only tightening `E_CONVERGENCE` for optimizations if not explicitly set

```
 if not psi4.has_option_changed('SCF', 'E_CONVERGENCE'):
     if procedures['energy'][lowername] == run_scf or procedures['energy'][lowername] == run_dft:
         psi4.set_local_option('SCF', 'E_CONVERGENCE', 8)
     else:
         psi4.set_local_option('SCF', 'E_CONVERGENCE', 10)
```

PSI Variables

Getting & Setting

- PSI variables are a "dictionary" of scalar quantities useful to the user, like energies, dipoles
- Live in `Process::environment.globals`
- Accessing C-side:

```
long int nQ_scf = Process::environment.globals["NAUX (SCF)"];
```

- Setting C-side:

```
Process::environment.globals["MP2 CORRELATION ENERGY"] = emp2_os + emp2_ss;
```

- Accessing Py-side:

```
emp3 = psi4.get_variable("MP3 TOTAL ENERGY")
```

- Setting Py-side:

```
psi4.set_variable('CBS TOTAL ENERGY', finalenergy)
```

- Input file:

```
print_variables()  # print all psi variables to output file
```

```
  Variable Map:
  ----------------------------------------------------------------------
  "CURRENT ENERGY"                          =     -39.725973187696
  "CURRENT REFERENCE ENERGY"                =     -39.725973187696
  "S22 DATABASE MEAN ABSOLUTE DEVIATION"    =       2.105857433479
  "SCF DIPOLE X"                            =      -0.000028920495
  "SCF TOTAL ENERGY"                        =     -39.725973187696
```

```
clean_variables()  # unset all psi variables before next computation
```

PSI Variables

Naming

- Naming: all capital letters, spaces are allowed, follow trends (lowercase implies variable)

```
"MP2 TOTAL ENERGY"                    "CI ROOT n -> ROOT m DIPOLE Y"
"MP2 CORRELATION ENERGY"              "MP2 SAME-SPIN ENERGY"
```

- Notice hierarchies of psivars that equal the same thing. This is good practice.

```
"CURRENT ENERGY"                      "CURRENT DIPOLE Z"
"CI TOTAL ENERGY"                     "CC DIPOLE Z"
"CISDT TOTAL ENERGY"                  "CCSD DIPOLE Z"
```

- Auto-documentation utilities scan the source for psivars and add them to module page

- Appendices
  - Keywords
    - Keywords by Alpha
    - Keywords by Module
  - Basis Sets
    - Basis Sets by Family
    - Basis Sets by Element
    - Auxiliary Basis Sets
  - PSI Variables
    - PSI Variables by Alpha
    - PSI Variables by Module
  - Miscellaneous
    - Test Suite and Sample Inputs
    - DFT Functionals
    - PSIOH Intermediate Files
    - Bibliography

CCENERGY

Computes coupled cluster energies. Called as part of any coupled cluster comput

- BRUECKNER CONVERGED
- CC CORRELATION ENERGY
- CC TOTAL ENERGY
- CC2 CORRELATION ENERGY
- CC2 TOTAL ENERGY
- CC3 CORRELATION ENERGY
- CC3 TOTAL ENERGY
- CCSD CORRELATION ENERGY
- CCSD OPPOSITE-SPIN CORRELATION ENERGY
- CCSD SAME-SPIN CORRELATION ENERGY
- CCSD TOTAL ENERGY
- CURRENT CORRELATION ENERGY
- CURRENT ENERGY
- LCC2 (+LMP2) TOTAL ENERGY
- LCCSD (+LMP2) TOTAL ENERGY
- MP2 CORRELATION ENERGY
- MP2 OPPOSITE-SPIN CORRELATION ENERGY
- MP2 SAME-SPIN CORRELATION ENERGY
- MP2 TOTAL ENERGY
- SCF TOTAL ENERGY
- SCF TOTAL ENERGY (CHKPT)
- SCS-CCSD CORRELATION ENERGY
- SCS-CCSD OPPOSITE-SPIN CORRELATION ENERGY
- SCS-CCSD SAME-SPIN CORRELATION ENERGY
- SCS-CCSD TOTAL ENERGY
- SCS-MP2 CORRELATION ENERGY
- SCS-MP2 OPPOSITE-SPIN CORRELATION ENERGY
- SCS-MP2 SAME-SPIN CORRELATION ENERGY
- SCS-MP2 TOTAL ENERGY
- SCSN-MP2 CORRELATION ENERGY
- SCSN-MP2 OPPOSITE-SPIN CORRELATION ENERGY

PSI Variables

Setting & Documenting: Special Cases

- Sometimes you want to set psivar name from a string or variable

```
/*- strings so that variable-name psi variables get parsed in docs -*/
/*- Process::environment.globals["MPn TOTAL ENERGY"] -*/
  std::stringstream s;
  s << label << (2*k) << " TOTAL ENERGY";
  Process::environment.globals[s.str()] = Empn2;
  s.str(std::string());  # empty stringstream
```

- Psivars in python modules should be in the docstring

```
>>> vi lib/python/driver.py
    def energy(name, **kwargs):
        r"""Function to compute the single-point electronic energy.

        :returns: (*float*) Total electronic energy in Hartrees. SAPT returns interaction energy.

        :PSI variables:

        .. hlist::
           :columns: 1

           * :psivar:`CURRENT ENERGY <CURRENTENERGY>`
           * :psivar:`CURRENT REFERENCE ENERGY <CURRENTREFERENCEENERGY>`
           * :psivar:`CURRENT CORRELATION ENERGY <CURRENTCORRELATIONENERGY>`
```

PSI Variables

Documentation

- New psivars must be added to the glossary for them to be clickable. Alphabetize!

```
>>> vi doc/sphinxman/source/glossary_psivariables.rst
PSI Variables by Alpha
======================

.. psivar:: CI ROOT n -> ROOT m DIPOLE X
   CI ROOT n -> ROOT m DIPOLE Y
   CI ROOT n -> ROOT m DIPOLE Z

   The three components of the transition dipole [Debye] between roots *n*
   and *m* for the requested configuration interaction level of theory.

.. psivar:: DFT TOTAL ENERGY

The total electronic energy [H] for the requested DFT
:math:`E_{\text{DFT}}` in Eq. :eq:`DFTterms`.

.. math::
   :nowrap:
   :label: DFTterms

      \begin{align*}
         E_{\text{DFT}} & = E_{NN} + E_{1e^-} + E_{2e^-} + E_{xc} + E_{\text{-D}} + E_{\text{DH}} \\
                        & = E_{\text{FCTL}} + E_{\text{-D}} + E_{\text{DH}} \\
                        & = E_{\text{SCF}} + E_{\text{DH}}
      \end{align*}

Unless the method is a DFT double-hybrid, this quantity is equal to
:psivar:`SCF TOTAL ENERGY <SCFTOTALENERGY>`. If the method is neither a
double-hybrid, nor dispersion corrected, this quantity is equal to
:psivar:`DFT FUNCTIONAL TOTAL ENERGY <DFTFUNCTIONALTOTALENERGY>`.
```

DFT TOTAL ENERGY

The total electronic energy [H] for the requested DFT method, E_{DFT} in Eq. (1).

$$\begin{aligned} E_{\text{DFT}} &= E_{NN} + E_{1e^-} + E_{2e^-} + E_{xc} + E_{\text{-D}} + E_{\text{DH}} \\ &= E_{\text{FCTL}} + E_{\text{-D}} + E_{\text{DH}} \\ &= E_{\text{SCF}} + E_{\text{DH}} \end{aligned} \tag{1}$$

Unless the method is a DFT double-hybrid, this quantity is equal to **SCF TOTAL ENERGY**. If the method is neither a double-hybrid, nor dispersion corrected, this quantity is equal to **DFT FUNCTIONAL TOTAL ENERGY**.

PSI Variables

Programming Obligations

- Modules ***must*** set psivars for energy
  - If non-correlated method, set `"CURRENT ENERGY"`, `"CURRENT REFERENCE ENERGY"`, `"method TOTAL ENERGY"`
  - If correlated method, set `"CURRENT ENERGY"`, `"CURRENT CORRELATION ENERGY"` (must work w/ current reference energy), `"method TOTAL ENERGY"`, `"method CORRELATION ENERGY"`
  - Set any other interesting quantities
  - Perfectly ok to set in driver if that's where the requested method is known

```
if ( lowername == 'fno-mp3' ):
    psi4.set_variable("CURRENT ENERGY", psi4.get_variable("MP3 TOTAL ENERGY"))
    psi4.set_variable("CURRENT CORRELATION ENERGY", psi4.get_variable("MP3 CORRELATION ENERGY"))
elif ( lowername == 'fno-mp4'):
    psi4.set_variable("CURRENT ENERGY", psi4.get_variable("MP4 TOTAL ENERGY"))
    psi4.set_variable("CURRENT CORRELATION ENERGY", psi4.get_variable("MP4 CORRELATION ENERGY"))
```

- Psivars with the same name ***must*** contain analogous information. For example, SS/OS variables need to use the same definition for MP2, CCSD, etc. especially when multiple modules compute the "same" method (e.g., MP2 and DF-MP2).
- Psivars are usually in atomic units, but check with the glossary and make sure you're setting them consistent with best practices.

PSI Variables

Programming Obligations

- The return value of `energy()`, `opt()`, and `freq()` is the value of `"CURRENT ENERGY"`.
- If `"CURRENT ENERGY"` is not set for a method, finite difference optimizations or frequencies won't work.
- If `"CURRENT DIPOLE X/Y/Z"` is not set for a method, anharmonic frequencies won't work.
- If gradient not in `Process::environment.gradient()`, sow/reap optimizations won't work.
- Abuse: psivars are handy for storing quantities to talk between modules or C-side to Py-side, but try to use them only for debugging so they don't show up in users' output files

Documentation

Free-Form Documentation with Sphinx

- Psi4 uses Sphinx documentation system, same as (written for) Python docs.
- Installable on Linux, very easy to install on Mac (`easy_install sphinx`).
- To build, in `objdir/doc/sphinxman`, run `make`
- Sphinx uses restructured text language (reST), so text file with light mark-up for links, headers, LaTeX math. Plenty of online guidance.
- Yes, the first build, changes to `read_options.cc` or to the driver do take forever. Sorry.
- Model off existing files. Include lots of links & tables. Observe Users/Programmers Partition.
- Be aware that reST can be more sensitive to white space than you would expect of a modern language. Be generous with white space. Tables must be perfectly aligned.
- Docs are on nightly build on sirius. If you break the docs build or notice it broken, inform Lori, Jet, or Andy to fix and/or restart.
- If Sphinx and reST are too much of a burden, check in text files with all the stuff you want in the docs (incl. LaTeX math) but unformatted, let me know, and I'll format it up.
- Many topics you needn't worry about because they're slurped up by auto-documenting scripts. These include: basis sets, auxiliary basis set defaults, databases, docstrings in `lib/python`, options in `src/bin/psi4/read_options.cc`, psivars throughout, psi files, physical constants, density functionals, methods available to `energy()`, etc., test cases, and the Boost Python module psi4. So concentrate on the free-form documentation.

Documentation

Free-Form Documentation with Sphinx

```
+-------------------------+-----------------------------------------------
| method                  | :math:`\Delta_{ij}`
+=========================+===============================================
| sdci                    | :math:`E_c`
+-------------------------+-----------------------------------------------
| dci                     | :math:`E_c`
+-------------------------+-----------------------------------------------
| cepa(0)                 | 0
+-------------------------+-----------------------------------------------
| cepa(1)                 | :math:`\frac{1}{2}\sum_k(\epsilon_{ik}+\epsil
+-------------------------+-----------------------------------------------
| cepa(3)                 | :math:`-\epsilon_{ij}+\sum_k(\epsilon_{ik}+\e
+-------------------------+-----------------------------------------------
| acpf                    | :math:`\frac{2}{N} E_c`
+-------------------------+-----------------------------------------------
| aqcc                    | :math:`[1-\frac{(N-3)(N-2)}{N(N-1)}]E_c`
+-------------------------+-----------------------------------------------

The pair correlation energy, :math:`\epsilon_{ij}`, is simply a partial
sum of the correlation energy.  In a spin-free formalism, the pair energy
is given by

.. math::
   :label: pair_energy

   \epsilon_{ij} = \sum_{ab} v_{ij}^{ab} (2 t_{ij}^{ab} - t_{ij}^{ba})

Methods whose shifts (:math:`\Delta_{ij}` and :math:`\Delta_i`) do not
explicitly depend on orbitals :math:`i` or :math:`j` (CISD, CEPA(0), ACPF,
and AQCC) have solutions that render the energy stationary with respect
variations in the amplitudes.  This convenient property allows density
matrices and 1-electron properties to be evaluated without any additional
effort.  Note, however, that 1-electron properties are currently
unavailable when coupling these stationary CEPA-like methods with frozen
natural orbitals.

Density-fitted coupled cluster
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

Density fitting (DF) [or the resolution of the identity (RI)] and Cholesky
decomposition (CD) techniques are popular in quantum chemistry to avoid
the computation and storage of the 4-index electron repulsion integral
(ERI) tensor and even to reduce the computational scaling of some terms.
DF/CD-CCSD(T) computations are available in |Psifour|, with or without the
use of FNOs, through the FNOCC module.  The implementation and accuracy of
the DF/CD-CCSD(T) method are described in Ref. [DePrince:2013:inpress]_\.

The DF-CCSD(T) procedure uses two auxiliary basis sets.  The first set is
that used in the SCF procedure, defined by the |scf__df_basis_scf|
keyword.  If this keyword is not specified, an appropriate -JKFIT set is
```

method	Δ_{ij}	Δ_i
sdci	E_c	E_c
dci	E_c	NA
cepa(0)	0	0
cepa(1)	$\frac{1}{2}\sum_k(\epsilon_{ik}+\epsilon_{jk})$	$\sum_k \epsilon_{ik}$
cepa(3)	$-\epsilon_{ij}+\sum_k(\epsilon_{ik}+\epsilon_{jk})$	$-\epsilon_{ii}+2\sum_k \epsilon_{ik}$
acpf	$\frac{2}{N}E_c$	$\frac{2}{N}E_c$
aqcc	$[1-\frac{(N-3)(N-2)}{N(N-1)}]E_c$	$[1-\frac{(N-3)(N-2)}{N(N-1)}]E_c$

The pair correlation energy, ϵ_{ij}, is simply a partial sum of the correlation energy. In a spin

$$\epsilon_{ij} = \sum_{ab} v_{ij}^{ab}(2t_{ij}^{ab} - t_{ij}^{ba})$$

Methods whose shifts (Δ_{ij} and Δ_i) do not explicitly depend on orbitals i or j (CISD, CE
the energy stationary with respect variations in the amplitudes. This convenient property
be evaluated without any additional effort. Note, however, that 1-electron propertie
stationary CEPA-like methods with frozen natural orbitals.

Density-fitted coupled cluster

Density fitting (DF) [or the resolution of the identity (RI)] and Cholesky decomposition (C
avoid the computation and storage of the 4-index electron repulsion integral (ERI) tensc
some terms. DF/CD-CCSD(T) computations are available in PSI4, with or without the
implementation and accuracy of the DF/CD-CCSD(T) method are described in Ref. [DePri

The DF-CCSD(T) procedure uses two auxiliary basis sets. The first set is that used in t
keyword. If this keyword is not specified, an appropriate -JKFIT set is automatically selec
the Fock matrix used in the DF-CCSD(T) procedure. The second auxiliary set is used
procedure. The choice of auxiliary basis is controlled by the keyword *DF_BASIS_CC*. B
DFMP2) most appropriate for use with the primary basis. For example, if the primary basis
aug-cc-pVDZ-RI.

New Module Checklist

Introduce Module to Gnu Make's Notice

- Create new directory in `src/bin` or `src/lib`. Distinction between a library and a binary is weak nowadays. General rule is a bin can call libs, lib can't call other libs.
- Copy in `Makefile.in` from another module and edit as needed. Add name to `src/bin/Makefile.in` or `src/lib/Makefile.in` .
- Add to `PSILIBS` list in `src/bin/psi4/Makefile.in`
- Add module `Makefile.in` and any other `.in` files (including test case `Makefile.in`s) to `configure.ac` .
- If perchance your code demands higher versions of Boost, Python, etc., see that the autoconf requirements get updated and update `INSTALL.rst` .

New Module Checklist

Introduce Module to Boost Python

- Add in `src/bin/psi4/python.cc` :
- namespace line

```
namespace psi {...
    namespace detci      { PsiReturnType detci(Options &);    }
... }
```

- C++ function

```
double py_psi_detci() {
    py_psi_prepare_options_for_module("DETCI");
    if (detci::detci(Process::environment.options) == Success) {
        return Process::environment.globals["CURRENT ENERGY"];
    }
    else
        return 0.0;
}
```

- Boost Python module export command

```
def("detci", py_psi_detci, "Runs the determinant-based configuration interaction code.");
```

New Module Checklist

Keywords: Introduce Module's Knobs to User

- Move options section from plugin code over to `src/bin/psi4/read_options.cc`
- Look over best practices for naming keywords and see if any should be renamed for consistency with greater Psi4 code.
- See if you have boolean and level/value pairs of options that should be combined into one keyword and tested for on-ness by `has_changed()`.
- Make sure all options have `/*- special comment -*/`
- In special comment, reference other keywords with `|module__keyword_name|` and if value of default is complicated, reference back (`:ref:`Post-SCF Convergence <table:conv_corl>``) to table or pages in docs explaining it.
- Add expert flag to options that user really shouldn't need to touch, especially those hiding route control (e.g., `WFN`, `DERTYPE`).
- Divide options up into sections by topic.
- For any string options where keyword has same name as other modules (e.g., `REFERENCE`), perform checks C-side in module that a valid value was sent. (Slight bug in Options class concatenates valid choices from all modules.)

New Module Checklist

Instruct the Driver as to How to Run Module

- Write `lib/python/proc.py run_method()` routine, which at its simplest, lists the order in which to call C++ modules to get a sensible energy out. Can start from plugin `pymodule.py` function. Add boilerplate as needed.
- Signature of `run_method(name, **kwargs)` in `lib/python/proc.py` must be just that. Name is requested method, e.g., scs(n)-mp2. kwargs are a dict of keyword arguments which can be empty or full so just carry it around.
- If using fitting basis sets, copy boilerplate into `lib/python/proc.py` as appropriate so auxiliary basis sets have defaults.
- For correlated methods that need 4-index integrals, be sure you add boilerplate to compute them before launching your module in `proc.py` .

Adding a Module to Psi4

Introduce the Module to the Greater Driver

- Add methods to procedures table in `lib/python/driver.py` . Naturally, add to `energy {}` but also to `gradient {}`, `hessian {}`, and `property {}` subdictionaries if *analytic* routines available.
- If one could plausibly basis-set-extrapolate your new method, add it and any underlying energies computed along the way to `VARH` in `lib/python/wrappers.py` to expose to `cbs()`.

```
VARH['mp2.5'] = {       'scftot': 'SCF TOTAL ENERGY',
                      'mp2corl': 'MP2 CORRELATION ENERGY',
                   'mp2.5corl': 'MP2.5 CORRELATION ENERGY',
                      'mp3corl': 'MP3 CORRELATION ENERGY'}
```

- Consider overlap of your new methods with existing modules, and decide if yours is better for particular circumstances: symmetry, reference, efficiency, etc. Set up the route in `lib/python/driver.py` . For example (neglecting density-fitting), MP2 directs to Ugur's `occ` module, MP3 directs to `occ` unless ROHF, which goes to David's `detci`, MP4 directs to Eugene's `fnocc` if RHF otherwise `detci`, and >MP4 directs to `detci`. Make sure that both codes are accessible (with a convoluted name) but that a simple name always gets the best code.

New Module Checklist

PSI Variables: Expose Module's Results to User In-Memory

- Set return values. Set `CURRENT ENERGY` C-side. If several methods run through same module (e.g., SCS or MPn), make sure the requested method is set to current Py-side.
- Set `CURRENT DIPOLE` if available and add current gradient Matrix to env, too.

```
Process::environment.wavefunction()->set_gradient(grad);
Process::environment.set_gradient(grad);
```

- Add other PSI variables to code
- Be consistent with other modules and check units
- Include special comments if variables in name

```
/*- Process::environment.globals["CI ROOT n -> ROOT m DIPOLE Z"] -*/
```

- Edit `doc/sphinxman/source/glossary_psivariables.rst` to define new psivars, incl. arithmetic.

Adding a Module to Psi4

Free-Form (non auto-doc) Documentation

- In `doc/sphinxman/source`, create a `.rst` file for your module. `fnocc.rst` and `sapt.rst` are good choices to model upon.
- Add that file to list in `doc/sphinxman/source/methods.rst` .
- Add that file to `STATICDOC` list in `doc/sphinxman/Makefile.in` .
- Add references for citation when ppl use your module in `doc/sphinxman/source/introduction.rst` .
- In main documentation file:
  - Add an options list for the most common or discussed options.
  - Add some theory/equations. Latex math formatting works.
  - Discuss the limitations of your code wrt symmetry, reference, dertype, parallelism.
  - A section of recommendations like end of `doc/sphinxman/source/dfmp2.rst` is nice.
  - Discuss how and what properties are available at what additional expense.

Adding a Module to Psi4

Scattered Documentation

- For every line added to procedures table in `lib/python/driver.py` , add line to corresponding reST table below in `energy()`, `optimize()`, `frequency()`, and `property()` functions that includes link to doc page.

```
| dcft                        | density cumulant functional theory :ref:`[manual] <sec:dcft>`      |
```

- If any environment variables affect the running of your module, notate them at `doc/sphinxman/source/external.rst` .
- Update the capabilities table in `doc/sphinxman/source/introduction.rst` . This links directly off `psicode.org` .

Adding a Module to Psi4

Isolate Interfaces

- If new code depends on an external program (e.g., `mrcc, dftd3, cfour`)
- Add tests to their own interface subdirectory, following model for cfour (e.g., `tests/cfour`)
- Adapt `Makefile.in` so tests only run if external program present. ?
- In driver, place all functions that call program in separate python file, probably at same level as `proc.py` .

- Make sure program runs if external binary in environment variables `PATH` or `PSIPATH`.

```
    # Find environment by merging PSIPATH and PATH environment variables
    lenv = os.environ
    lenv['PATH'] = ':'.join([os.path.abspath(x) for x in os.environ.get('PSIPATH', '').split(':')]) + ':' +
lenv.get('PATH') + ':' + psi4.Process.environment["PSIDATADIR"] + '/basis' + ':' + psi4.psi_top_srcdir() + '/lib/basis'
```

- On `doc/sphinxman/source/interfacing.rst` , add a bullet point and a linking page.

New Module Checklist

PSI4 Includes: Introduce Data and Registries to Module

- Convert any physical constants to `pc_*` style (e.g., `pc_bohr2angstroms`) for suite consistency.
- Reference files in `include/` for any periodic table info (elements, Z, masses).
- Address psifiles.
- Check that any psifiles you use are registered in `include/psifiles.h`
- Fill in the documentation field thereat: `/*- AA UHF twopdm presort file -*/`
- Run at least the pre-Sphinx part of make docs so that psifiles gets translated for Py-side.

Adding a Module to Psi4

Test Cases: Introduce Module to Hazard and Deliverance

- Add test cases that cover the range of module capabilities.
- Each test case `input.dat` should have a descriptive comment line beginning `#!` . This will get slurped by the auto-doc into an appendix of available test cases.
- Make sure `input.dat`, `output.ref`, and `Makefile.in` are all present in test directory.
- Add all `.in` files to `configure.ac` .
- Add all test cases to new, existing, or multiple lists in `tests/Makefile.in` .
- Add a couple test cases to `quicktests` target in `tests/Makefile.in` .
- If analytic gradients are available, actually return and compare the gradient matrices (e.g., `mp2-grad2`).
- Check that tests have sensible convergence options versus decimal places checked. On other architectures, won't match to eight digits. For optimizations, do really tight opt, then check digits in proportion to `G_CONVERGENCE`.

Adding a Module to Psi4

Burnish Module Output

- Consider making a final pass through output formatting.
- Adopt `--> Heading <--` headings
- Purge `\t` characters since they don't display nicely in vi or when tab size varies.
- Don't use tilde (reserved for optimizations) or @ (unless grep-able).
- Add units where appropriate.
- Energies in Hartrees should have no fewer than 8 decimal positions and no more than 12
- Check that you're printing nothing to `std::out`.

Adding a Module to Psi4

Stress the Module & Introduce It to the World

- Valgrind your code.
- Review all the imported header files to see if all are necessary. Replace with forward declarations where appropriate.
- Make sure your method can do job / clean() / job / clean() in a single input file with different basis sets. Running the method through `cbs()` is another way to test this.
- Check that there are no `goto` statements. Future grad students will laugh and `git blame` you.
- Build, run complete test suite, commit, check for un-added files, push (or submit pull request).
- Delete any associated plugins in repository (not needed nowadays).