Tour of PSI & Design Philosophy

Dr. Justin M. Turney

Friday, January 24, 2014
**Psi4**: An open-source *ab initio* electronic structure program

Psi4 is a new approach to modern quantum chemistry.

The program is written entirely in C++ and relies on a new infrastructure that has been designed to permit high-efficiency computations of both standard and emerging electronic structure methods.
Psi4: An open-source *ab initio* electronic structure program

Psi4 offers flexible user input built on the Python scripting language.

This enables...

- both new and experienced users to make full use of the program’s capabilities
- even to implement new functionality with moderate effort.
**Psi4: open-source**

To maximize its impact and usefulness, Psi4 is available through an open-source license to everyone.

To the best of my knowledge people are working to get Psi4 included in the main repositories of Fedora and Debian (and by extension Ubuntu).
Design Philosophy

The design philosophy of \texttt{Psi4} is to be as user-friendly as possible without sacrificing performance or extensibility and to choose the best method for the problem.
Design Philosophy

C++ is used for performance critical parts:

- Integrals and transformation
Design Philosophy

C++ is used for performance critical parts:

- Integrals and transformation
- SCF (HF and DFT)
C++ is used for performance critical parts:

- Integrals and transformation
- SCF (HF and DFT)
- Coupled cluster (CC)
Design Philosophy

C++ is used for performance critical parts:

- Integrals and transformation
- SCF (HF and DFT)
- Coupled cluster (CC)
- Configuration Interaction (CI)
Design Philosophy

C++ is used for performance critical parts:

- Integrals and transformation
- SCF (HF and DFT)
- Coupled cluster (CC)
- Configuration Interaction (CI)
- Symmetry Adapted Perturbation Theory (SAPT)
Design Philosophy

C++ is used for performance critical parts:

- Integrals and transformation
- SCF (HF and DFT)
- Coupled cluster (CC)
- Configuration Interaction (CI)
- Symmetry Adapted Perturbation Theory (SAPT)
- and others
Design Philosophy

Python is used for everything else:

• Input file
Design Philosophy

Python is used for everything else:

- Input file
- Procedures for performing a calculation
Design Philosophy

Python is used for everything else:

- Input file
- Procedures for performing a calculation
- Mixing of DFT functionals
Design Philosophy

Python is used for everything else:

- Input file
- Procedures for performing a calculation
- Mixing of DFT functionals
- $G_n$ methods
Design Philosophy

Python is used for everything else:

• Input file
• Procedures for performing a calculation
• Mixing of DFT functionals
• $G_n$ methods
• Focal point
Design Philosophy

Python is used for everything else:
- Input file
- Procedures for performing a calculation
- Mixing of DFT functionals
- $G_n$ methods
- Focal point
- PubChem interface
**Psi4**: *open-source*

You can obtain **Psi4** from GitHub:
Psi4: open-source

To clone Psi4 onto your box:

```bash
$ git clone git@github.com:psi4/psi4public.git
```
What do you get when you download \texttt{Psi4}?

Let’s go through the directory structure:

\begin{verbatim}
psi4public/
  ├ bin/
  ├ boost/
  ├ doc/
  ├ include/
  ├ lib/
  ├ plugins/
  ├ samples/
  ├ src/
  └ tests/
\end{verbatim}
What do you get when you download \texttt{Psi4}?

Let's go through the directory structure:

```
psi4public/
  └── bin/
  └── boost/
  └── doc/
  └── include/
  └── lib/
  └── plugins/
  └── samples/
  └── src/
  └── tests/
```

\texttt{bin} contains scripts needed for \texttt{configure}. This directory is hardly ever touched.

Chances are you don't need to bother with this directory ...ever.
What do you get when you download **Psi4**?

Let's go through the directory structure:

```bash
tree psi4public/
  bin/
  boost/
  doc/
  include/
  lib/
  plugins/
  samples/
  src/
  tests/
```

- **psi4public/** contains **Psi4**.
- **bin/** contains **Psi4**.
- **boost/** contains **Psi4**'s provided Boost tarball.
- **doc/** contains documentation.
- **include/** contains header files.
- **lib/** contains library files.
- **plugins/** contains plugins.
- **samples/** contains sample code.
- **src/** contains source code.
- **tests/** contains test code.

If your system has a compatible Boost that is use, if not, **Psi4** will compile one for you.
What do you get when you download Psi4?

Let’s go through the directory structure:

```
psi4public/
  └── bin/
  └── boost/
  └── doc/
  └── include/
  └── lib/
  └── plugins/
  └── samples/
  └── src/
  └── tests/
```

doc contains Psi4’s documentation or the scripts and files for generating the documentation.
What do you get when you download \texttt{Psi4}?

Let’s go through the directory structure:

```plaintext
psi4public/
  bin/
  boost/
  doc/
  include/
  lib/
  plugins/
  samples/
  src/
  tests/
```

\texttt{include} contains global header files used by both libraries and modules.

For example: \texttt{masses.h}, \texttt{physconst.h}, \texttt{psi4-dec.h}, and \texttt{psifiles.h}
What do you get when you download \texttt{Psi4}?

Let's go through the directory structure:

\begin{verbatim}
psi4public/
  __ bin/
  __ boost/
  __ doc/
  __ include/
  __ lib/
  __ plugins/
  __ samples/
  __ src/
  __ tests/

lib/
  __ basis/
  __ databases/
  __ grids/
  __ plugin/
  __ psi4/
  __ python/
  __ quadratures/
  __ scripts/
\end{verbatim}
What do you get when you download **Psi4**?

Let’s go through the directory structure:

```
psi4public/
  __bin/
  __boost/
  __doc/
  __include/
  __lib/
  __plugins/
  __samples/
  __src/
  __tests/
```

**plugins** contains sample plugin files. These include aointegrals, backtrans, mointegrals, mp2, and sointegrals.
What do you get when you download \texttt{Psi4}? 

Let's go through the directory structure:

```
psi4public/
  __ bin/
  __ boost/
  __ doc/
  __ include/
  __ lib/
  __ plugins/
  __ samples/
  __ src/
  __ tests/
```

\texttt{samples} contains example input files. Each example input file lives in it's own subdirectory of \texttt{samples}.
What do you get when you download \texttt{Psi4}?

Let’s go through the directory structure:

```
psi4public/
  ___ bin/
  ___ boost/
  ___ doc/
  ___ include/
  ___ lib/
  ___ plugins/
  ___ samples/
  ___ src/
  ___ tests/
```

\texttt{src} contains the C++ source code for the libraries and modules of \texttt{Psi4}. These will be discussed in detail shortly.
What do you get when you download **Psi4**?

Let’s go through the directory structure:

```plaintext
psi4public/
  __bin/
  __boost/
  __doc/
  __include/
  __lib/
  __plugins/
  __samples/
  __src/
  __tests/
```

tests contains the end test cases for **Psi4**. The samples are derived from the tests.

If you’re looking for inputs to start from don’t use these as they contain extra code specific for testing.
What libraries are provided by Psi4?

Let’s go through the libraries provided by Psi4:

```
lib/
  __lib3index/
  __libchkpt/
  __libciomr/
  __libderiv/
  __libdiis/
  __libdisp/
  __libdpd/
  __liberd/
  __libfock/
  __libfunctional/
  __libint/
  __libiwl/
```

`lib3index` provides classes for DF/RI integrals, Cholesky decomposition, and other types of decompositions.
What libraries are provided by \texttt{Psi4}?

Let's go through the libraries provided by \texttt{Psi4}:

\begin{itemize}
  \item \texttt{lib/}
  \item \hspace{1cm} \texttt{lib3index/}
  \item \hspace{1cm} \texttt{libchkpt/}
  \item \hspace{1cm} \texttt{libciomr/}
  \item \hspace{1cm} \texttt{libderiv/}
  \item \hspace{1cm} \texttt{libdiis/}
  \item \hspace{1cm} \texttt{libdisp/}
  \item \hspace{1cm} \texttt{libdpd/}
  \item \hspace{1cm} \texttt{liberd/}
  \item \hspace{1cm} \texttt{libfock/}
  \item \hspace{1cm} \texttt{libfunctional/}
  \item \hspace{1cm} \texttt{libint/}
  \item \hspace{1cm} \texttt{libiw1/}
\end{itemize}

\texttt{libchkpt} provides access to the old file32 checkpoint file.

This is here mostly for legacy codes. Newer codes are discouraged from using it.
What libraries are provided by Psi4?

Let's go through the libraries provided by Psi4:

```
lib/
  __lib3index/
  __libchkpt/
  __libciomr/
  __libderiv/
  __libdiis/
  __libdisp/
  __libdpd/
  __liberd/
  __libfock/
  __libfunctional/
  __libint/
  __libiwl/
```

libciomr provides a collection of functions that deal with files, matrices, printing, some math (matrix multiplication and diagonalization).

Newer codes tend to not use this library much.
What libraries are provided by Psi4?

Let’s go through the libraries provided by Psi4:

```
lib/
  __lib3index/
  __libchkpt/
  __libciomr/
  __libderiv/
  __libdiis/
  __libdisp/
  __libdpd/
  __liberd/
  __libfock/
  __libfunctional/
  __libint/
  __libiwl/
```

libderiv computes 1\textsuperscript{st} and 2\textsuperscript{nd} derivatives of two-electron integrals using the Obara-Saika recursion scheme.

You should not directly use this library. Instead, use libmints to abstract and simply the interface.
What libraries are provided by Psi4?

Let’s go through the libraries provided by Psi4:

```
lib/
  __lib3index/
  __libchkpt/
  __libciomr/
  __libderiv/
  __libdiis/
  __libdisp/
  __libdpd/
  __liberd/
  __libfock/
  __libfunctional/
  __libint/
  __libiwl/
```

`libdiis` handles DIIS extrapolations.

Compatible with `libdpd` buffers and `libmints` matrices.
What libraries are provided by \texttt{Psi4}?

Let’s go through the libraries provided by \texttt{Psi4}:

```
lib/
  - lib3index/  
  - libchkpt/   
  - libciomr/   
  - libderiv/   
  - libdiis/    
  - libdisp/    
  - libdpd/     
  - liberd/     
  - libfock/    
  - libfunctional/  
  - libint/     
  - libiwl/     
```

\texttt{libdisp} handles computing dispersion corrections; e.g. Grimme’s -D2 and -D3.
What libraries are provided by Psi4?

Let’s go through the libraries provided by Psi4:

```
lib/
  __lib3index/
  __libchkpt/
  __libciomr/
  __libderiv/
  __libdiis/
  __libdisp/
  __libdpd/
  __liberd/
  __libfock/
  __libfunctional/
  __libint/
  __libiwil/
```

**libdpd** Daniel’s Direct Product Decomposition library for tensors commonly found coupled cluster theory.

Andy Simmonett has been working to add DF/RI capabilities to libdpd.
What libraries are provided by \texttt{Psi4}?

Let’s go through the libraries provided by \texttt{Psi4}:

\begin{verbatim}
lib/
  __lib3index/
  __libchkpt/
  __liciomr/
  __libderiv/
  __libdiis/
  __libdisp/
  __libdpd/
  __liberd/
  __libfock/
  __libfunctional/
  __libint/
  __libiwl/
\end{verbatim}

- \texttt{liberd} an integrals package from ACESIII. Currently, only compiled with using CMake.

This is the only Fortran code found in \texttt{Psi4}. 
What libraries are provided by \texttt{Psi4}?

Let's go through the libraries provided by \texttt{Psi4}:

```
lib/
  lib3index/
  libchkpt/
  libciomr/
  libderiv/
  libdiis/
  libdisp/
  libdpd/
  liberd/
  libfock/
  libfunctional/
  libint/
  libiwl/
```

\texttt{libfock} handles all things pertaining to a Fock matrix.
What libraries are provided by \texttt{Psi4}?

Let’s go through the libraries provided by \texttt{Psi4}:

\begin{itemize}
  \item \texttt{lib/}
  \item \hspace{1cm} \texttt{lib3index/}
  \item \hspace{1cm} \texttt{libchkpt/}
  \item \hspace{1cm} \texttt{libciomr/}
  \item \hspace{1cm} \texttt{libderiv/}
  \item \hspace{1cm} \texttt{libdiis/}
  \item \hspace{1cm} \texttt{libdisp/}
  \item \hspace{1cm} \texttt{libdpd/}
  \item \hspace{1cm} \texttt{liberd/}
  \item \hspace{1cm} \texttt{libfock/}
  \item \hspace{1cm} \texttt{libfunctional/}
  \item \hspace{1cm} \texttt{libint/}
  \item \hspace{1cm} \texttt{libiw/}
\end{itemize}

\texttt{libfunctional} handles computing of DFT functionals and their quadratures. The low level code for DFT lives here.
What libraries are provided by \texttt{Psi4}?

Let’s go through the libraries provided by \texttt{Psi4}:

\begin{itemize}
  \item \texttt{lib/}
  \item \texttt{lib3index/}
  \item \texttt{libchkpt/}
  \item \texttt{libciomr/}
  \item \texttt{libderiv/}
  \item \texttt{libdiis/}
  \item \texttt{libdisp/}
  \item \texttt{libdpd/}
  \item \texttt{liberd/}
  \item \texttt{libfock/}
  \item \texttt{libfunctional/}
  \item \texttt{libint/}
  \item \texttt{libiwl/}
\end{itemize}

\texttt{libint} the original integrals library of \texttt{Psi}. You should not directly use this library use \texttt{libmints} instead.
What libraries are provided by \texttt{Psi4}?
Let’s go through the libraries provided by \texttt{Psi4}:

\begin{verbatim}
lib/
  |__lib3index/
  |__libchkpt/
  |__libciomr/
  |__libderiv/
  |__libdiis/
  |__libdisp/
  |__libdptd/
  |__liberd/
  |__libfock/
  |__libfunctional/
  |__libint/
  |__libiw1/
\end{verbatim}

\texttt{libiw1} the Integrals With Labels library. Handles reading AO integrals from disk.
What libraries are provided by Psi4?

lib/  
  _libmints/  
  _liboptions/  
  _libplugin/  
  _libpsio/  
  _libqt/  
  _libsapt_solver/  
  _libscf_solver/  
  _libthce/  
  _libtrans/  

libmints handles computing one- and two-electron integrals in a user-friendly fashion.
What libraries are provided by Psi4?

lib/
  ___libmints/
  ___liboptions/
  ___libplugin/
  ___libpsio/
  ___libqt/
  ___libsapt_solver/
  ___libscf_solver/
  ___libthce/
  ___libtrans/

liboptions manages all user options from the input file.
What libraries are provided by Psi4?

- lib/
  - _libmints/
  - _liboptions/
  - _libplugin/
  - _libpsio/
  - _libqt/
  - _libsapt_solver/
  - _libscf_solver/
  - _libthce/
  - _libtrans/

libplugin manages C++ side of the plugin system in Psi4. Most programmers need not worry about this.
What libraries are provided by Psi4?

lib/
  __libmints/
  __liboptions/
  __libplugin/
  __libpsio/
  __libqt/
  __libsapt_solver/
  __libscf_solver/
  __libthce/
  __libtrans/

libpsio PSI’s I/O manager. All binary files operations go through this library.
What libraries are provided by Psi4?

- `lib/
  - __libmints/`
  - __liboptions/`
  - __libplugin/`
  - __libpsio/`
  - __libqt/`
  - __libsapt_solver/`
  - __libscf_solver/`
  - __libthce/`
  - __libtrans/`

libqt “qt“ short for the Quantum Trio is a collection of useful functions used throughout Psi4/
What libraries are provided by Psi4?

lib/
  __libmints/
  __liboptions/
  __libplugin/
  __libpsio/
  __libqt/
  __libsapt Solver/
  __libscf Solver/
  __libthce/
  __libtrans/

libsapt Solver core solver
for the Symmetry Adapted Perturbation Theory module.
What libraries are provided by Psi4?

lib/
  __libmints/
  __liboptions/
  __libplugin/
  __libpsio/
  __libqt/
  __libsapt_solver/
  __libscf_solver/
  __libthce/
  __libtrans/

libscf_solver solver for HF theory and DFT.
What libraries are provided by Psi4?

```
lib/
  __libmints/
  __liboptions/
  __libplugin/
  __lipsio/
  __libqt/
  __libsapt_solver/
  __libscf_solver/
  __libthce/
  __libtrans/
```

libthce the Tensor Hyper-Contraction Engine. Rob will discuss this later.
What libraries are provided by Psi4?

lib/
  __libmints/
  __liboptions/
  __libplugin/
  __libpsio/
  __libqt/
  __libsapt_solver/
  __libscf_solver/
  __libthce/
  __libtrans/

libtrans the integral transformation library. Very flexible. Can transformation and back-transform to and from any file.
_compiling a large software package like \texttt{Psi4} can take 30 minutes or more depending on configuration and computer.

So, we came up with a plugin system that allows developers to work with just their code (not worrying about \texttt{Psi4}) but still having all of \texttt{Psi4}'s capabilities available to them.
Developer Friendly - Plugin System

**NOTE:** Plugins are only available if you configured your version of Psi4 with the `--with-plugins` command line configure option.

```
$ ../configure --with-plugins
```

While developing we recommend the following minimum configure flags:

```
$ ../configure --without-opt --with-debug --with-plugins
```
Creating a new plugin

**NOTE:** This assumes the psi4 executable is in your active PATH environment variable.

Tell Psi4 to create your new plugin:

1. `$ psi4 --new-plugin hellosi4`

The syntax for this call is: `--new-plugin name {+templatename}`
Creating a new plugin

The current list of plugin templates:

• aointegrals
• mointegrals
• scf
• sointegrals
• wavefunction
Creating a new plugin

1. $ psi4 --new-plugin hellopsi4

hellopsi4/
  __Makefile
  __init__.py
  doc.rst
  hellopsi4.cc
  input.dat
  pymodule.py

Makefile contains the make rules for compiling this plugin against the Psi4 used to create it.
Creating a new plugin

1. $ psi4 --new-plugin hellopsi4

hellopsi4/
  ├── Makefile
  │    ├── __init__.py
  │    └── doc.rst
  │    └── hellopsi4.cc
  ├── input.dat
  └── pymodule.py

__init__.py includes Python commands to ensure your plugin is loaded when Psi4 runs.
Creating a new plugin

$ psi4 --new-plugin hellopsi4

doc.rst contains a documentation template for your plugin in restructured text format.

hellopsi4/
  __init__.py
  doc.rst
  hellopsi4.cc
  input.dat
  pymodule.py
Creating a new plugin

$ psi4 --new-plugin hellopsi4

hellopsi4/
  Makefile
  __init__.py
  doc.rst
  hellopsi4.cc
  input.dat
  pymodule.py

hellopsi4.cc is your new basic plugin source file.
Creating a new plugin

$ psi4 --new-plugin hellopsi4

```
hellopsi4/
  Makefile
  __init__.py
  doc.rst
  hellopsi4.cc
  input.dat
  pymodule.py
```

input.dat basic input file with enough in it to run your plugin.
Creating a new plugin

$ psi4 --new-plugin hellopsi4

<table>
<thead>
<tr>
<th>hellopsi4/</th>
<th>pymodule.py template Python wrapper for your plugin.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Makefile</td>
<td></td>
</tr>
<tr>
<td><strong>init</strong>.py</td>
<td></td>
</tr>
<tr>
<td>doc.rst</td>
<td></td>
</tr>
<tr>
<td>hellopsi4.cc</td>
<td>Executes SCF prior to calling your code.</td>
</tr>
<tr>
<td>input.dat</td>
<td></td>
</tr>
<tr>
<td>pymodule.py</td>
<td>This serves as an example when you're incorporating your plugin fully into Psi4 proper.</td>
</tr>
</tbody>
</table>
sys.path.insert(0, '../')
import hellopsi4

molecule {
    O
    H 1 R
    H 1 R 2 A
    R = .9
    A = 104.5
}

set {
    basis sto-3g
}

set hellopsi4 {
    print 1
}

energy('hellopsi4')

hellopsi4.exampleFN()
What’s in hellopsi4.cc?

Let’s go through the code:

```cpp
#include <libplugin/plugin.h>
#include <psi4-dec.h>
#include <libparallel/parallel.h>
#include <liboptions/liboptions.h>
#include <libmints/mints.h>
#include <libpsio/psio.hpp>

INIT_PLUGIN

using namespace boost;

namespace psi
{
  namespace hellopsi4
  {
```
What’s in hellopsi4.cc?

Let’s go through the code:

```cpp
extern "C"
int read_options(std::string name, Options& options) {
  if (name == "HELLOPSI4" || options.read_globals()) {
    /*− The amount of information printed to the output file −*/
    options.add_int("PRINT", 1);
  }
  return true;
}
```
What’s in hellopsi4.cc?

Let’s go through the code:

```c
extern "C"
PsiReturnType hellopsi4(Options& options)
{
    int print = options.get_int("PRINT");

    /* Your code goes here */
    return Success;
}

// End namespaces
```
What can we assume in this plugin?

At this point in the code we can safely assume

• Options from the input file have been validated and current options (specific to our plugin and the globals) are available to us.

• The molecule and the current geometry are available to us.

• That a standard HF calculation was performed.
Debugging your plugin

Common programs used in debugging include:

- **gdb** → The GNU Debugger
- **lldb** → Low Level Virtual Machine Debugger. Standard on Mac OS X Mavericks
- **idb** → Intel Debugger. Available if you have Intel compilers.

Diagnosing memory issues

- **valgrind** → valgrind.org
Using valgrind

You will find it very useful to use a Python suppression file when using valgrind with Psi4.

In the file $HOME/.valgrindrc put the following; replacing $topsrcdir with the location of your Psi4 code.

1  --suppressions=$topsrcdir/lib/valgrind-python.supp
Summary

We have taken a whirlwind tour of Psi4 focused on giving you a broad overview of the code and working with it. In the following talks more detail will be given into specific libraries and the capabilities of Psi4.