PyRosetta-4

What’s new and why should you care!

% cd main/source/src/python/PyRosetta && ./build.py -j8

Sergey Lyskov, GrayLab@JHU
PyRosetta-3

- GCCXML → XML
- Boost.Python
- Resulting code: C++98
PyRosetta-3

- GCCXML → XML
- Boost.Python
- Resulting code: C++98
- Can only be built with GCC and GNU libstdc++
- Mac’s build is limited to use old version of GCC-4.1
- No support for C++11
- SLOOOOOOW
PyRosetta-4
PyRosetta-4

• GCCXML
PyRosetta-4

- GCCXML
PyRosetta-4

- GCCXML

- Binder: tool for creating Python bindings for arbitrary C++ code.
PyRosetta-4

- GCCXML

- Binder: tool for creating Python bindings for arbitrary C++ code.
  - based on Clang libTooling
PyRosetta-4

- GCCXML

- Binder: tool for creating Python bindings for arbitrary C++ code.
  - based on Clang libTooling
  - implemented in C++
PyRossetta-4

• GCCXML

• Binder: tool for creating Python bindings for arbitrary C++ code.
  • based on Clang libTooling
  • implemented in C++
  • supports parsing of C++11, C++14, C++17,…
PyRosetta-4
PyRosetta-4

- Boost.Python → PyBind11
PyRosetta-4

- Boost.Python → PyBind11
PyRosetta-4

- Boost.Python $\rightarrow$ PyBind11
- Our-own-custom-build-system $\rightarrow$ CMake
Architecture

• PyRosetta-3: ‘rosetta’ included both rosetta and PyRosetta code

• PyRosetta-4: two separate entities:
  • rosetta.so (bindings for Rosetta C++ code) and
  • pyrosetta

```python
from __future__ import print_function

import rosetta
import pyrosetta

pyrosetta.init()
print( pyrosetta.version() )
```
What’s new in PyRosetta-4?
What’s new in PyRosetta-4?

• Automatic bindings generation for C++ templates
What’s new in PyRosetta-4?

• Automatic bindings generation for C++ templates
• Classes with virtual function: no run-time overhead for overload classes!
What’s new in PyRosetta-4?

- Automatic bindings generation for C++ templates
- Classes with virtual function: no run-time overhead for overload classes!
- Function default arguments now bound properly (via C++11 lambda). For example when binding void foo(int a=1, int b=2); we will generate:

```cpp
void py_foo() { foo(); }
void py_foo(int a) { foo(a); }
void py_foo(int a, int b) { foo(a, b); }
```
What’s new in PyRosetta-4?
What’s new in PyRosetta-4?

- Binding for function accepting pointer to primitive types: int *, double *, bool *, … etc
What’s new in PyRosetta-4?

- Binding for function accepting pointer to primitive types: int *, double *, bool *, … etc
- Support for bindings of C++11 code and C++14!
What’s new in PyRosetta-4?

- Binding for function accepting pointer to primitive types: int *, double *, bool *, ... etc
- Support for bindings of C++11 code and C++14!
- ‘python setup.py install’ → SetPyRosettaEnvironment.sh
What’s new in PyRosetta-4?

- Binding for function accepting pointer to primitive types: int *, double *, bool *, … etc

- Support for bindings of C++11 code and C++14!

- ‘python setup.py install’ → SetPyRosettaEnvironment.sh

- New source location:
  main/source/src/python/PyRosetta
Build types
Build types

- Old PyRosetta-3: namespace/monolith builds
Build types

- Old PyRosetta-3: namespace/monolith builds
- New PyRosetta-4: **ONLY MONOLITH BUILD**
Build types

- Old PyRosetta-3: namespace/monolith builds
- New PyRosetta-4: ONLY MONOLITH BUILD
- Debug/Release/MinSizeRel/RelWithDebInfo
What code is bound?
What code is bound?

- All ‘bindable’ regular code (no exceptions!) And this includes:
What code is bound?

• All ‘bindable’ regular code (no exceptions!) And this includes:

• ObjexxFCL
What code is bound?

• All ‘bindable’ regular code (no exceptions!) And this includes:
  • ObjexxFCL
  • Option System
What code is bound?

- All ‘bindable’ regular code (no exceptions!) And this includes:
  - ObjexxFCL
  - Option System

- At this moment only skipped code is ether:
What code is bound?

• All ‘bindable’ regular code (no exceptions!) And this includes:
  
  • ObjexxFCL
  
  • Option System

• At this moment only skipped code is ether:
  
  • Template code that could not be instantiated
What code is bound?

- All ‘bindable’ regular code (no exceptions!) And this includes:
  - ObjexxFCL
  - Option System

- At this moment only skipped code is ether:
  - Template code that could not be instantiated
  - Code that could not be adequately represented in Python
Python versions compatibility

• Thank you for participating in Python version survey!

Which versions of Python should we support for PyRosetta builds?
(13 responses)

- Python-2 only. I do not care about Python-3 at all and do not plan to use it! Maybe Python-4 later...
- Both Python-2 and Python-3 versions should be supported even if that raise complexity.
- Python-3 only! Let's embrace the future!
Python versions compatibility

• Both generation of bindings for Python-2 and Python-3 now supported

• PyRosetta python code, demos, tests are compatible with both Python-2 and Python-3
What is not yet implemented?

• No Python ‘doc’ strings yet
• print <object>
• PyRosetta ‘apps’ is not yet ported
• PyRosetta GUI tests is not yet ported
• PyRosetta test C001_Carbohydrates_Demo01 fail when ‘installed’
• Python PyMOL Mover implementation is not ported and deprecated (please use C++ version instead)
Limitations
Limitations

• Right now needs to be built with Clang
Limitations

• Right now needs to be built with Clang

• std::pair and std::tuple is read-only
Limitations

• Right now needs to be built with Clang

• `std::pair` and `std::tuple` is *read-only*

• no support for multiple inheritance (and probably will never be due to PyBind11 limitations)
Build Speed
## Build Speed

<table>
<thead>
<tr>
<th>Generation phase</th>
<th>Build phase</th>
<th>Total (Rosetta excluded)</th>
</tr>
</thead>
</table>


## Build Speed

<table>
<thead>
<tr>
<th></th>
<th>Generation phase</th>
<th>Build phase</th>
<th>Total (Rosetta excluded)</th>
</tr>
</thead>
<tbody>
<tr>
<td>PyRosetta-3</td>
<td>5.9 cpu·h</td>
<td>27.2 cpu·h</td>
<td>33.1 cpu·h</td>
</tr>
</tbody>
</table>
## Build Speed

<table>
<thead>
<tr>
<th></th>
<th>Generation phase</th>
<th>Build phase</th>
<th>Total (Rosetta excluded)</th>
</tr>
</thead>
<tbody>
<tr>
<td>PyRosetta-3</td>
<td>5.9 cpu·h</td>
<td>27.2 cpu·h</td>
<td>33.1 cpu·h</td>
</tr>
<tr>
<td>PyRosetta-4</td>
<td>0.06 cpu·h, 3.5 cpu·min</td>
<td>5.6 cpu·h</td>
<td>5.66 cpu·h (17%!)</td>
</tr>
</tbody>
</table>
Size and Memory consumption
## Size and Memory consumption

<table>
<thead>
<tr>
<th>PyRosetta-3</th>
<th>1,368 Mb allocating ~4Gb on scoring!!!</th>
<th>100%</th>
</tr>
</thead>
<tbody>
<tr>
<td>boost.python, release</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
## Size and Memory consumption

<table>
<thead>
<tr>
<th></th>
<th>Size</th>
<th>Utilization</th>
</tr>
</thead>
<tbody>
<tr>
<td>PyRosetta-3 boost.python, release</td>
<td>1,368 Mb allocating ~4Gb on scoring!!!</td>
<td>100%</td>
</tr>
<tr>
<td>PyRosetta-4 PyBind11, release, Linux</td>
<td>346 Mb</td>
<td>25%</td>
</tr>
</tbody>
</table>
## Size and Memory consumption

<table>
<thead>
<tr>
<th>PyRosetta-3</th>
<th>1,368 Mb allocating ~4Gb on scoring!!!</th>
<th>100%</th>
</tr>
</thead>
<tbody>
<tr>
<td>PyRosetta-4</td>
<td>346 Mb</td>
<td>25%</td>
</tr>
<tr>
<td>PyBind11, release, Linux</td>
<td></td>
<td></td>
</tr>
<tr>
<td>PyRosetta-4</td>
<td>244 Mb</td>
<td>18%</td>
</tr>
<tr>
<td>PyBind11, MinSizeRelease, Linux</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Overall PyRosetta-4:
Overall PyRosetta-4:

• Binds more code (all templates!)
Overall PyRosetta-4:

- Binds more code (all templates!)
- Binds all default function arguments
Overall PyRosetta-4:

- Binds more code (all templates!)
- Binds all default function arguments
- Builds x5 faster
Overall PyRosetta-4:

- Binds more code (all templates!)
- Binds all default function arguments
- Builds x5 faster
- Consumes x5 less memory
Overall PyRosetta-4:

- Binds more code (all templates!)
- Binds all default function arguments
- Builds x5 faster
- Consumes x5 less memory
- Standard Python package
Binder architecture
Binder architecture

- No mention of ‘Rosetta’ or Rosetta specific classes in the code!
Binder architecture

• No mention of ‘Rossetta’ or Rosetta specific classes in the code!

• Config file to specify bindings options, special binders, ... etc: pluggable architecture
Binder architecture

• No mention of ‘Rosetta’ or Rosetta specific classes in the code!

• Config file to specify bindings options, special binders, … etc: **pluggable architecture**

• I am going to release Binder as separate package under **FreeBSD license** this fall
PyRosetta-4, when?
PyRosetta-4, when?  - right now!
PyRosetta-4, when? - right now!
**PyRosetta-4, when? - right now!**

### PyRosetta Tests
- linux.PyRosetta.build
- linux.PyRosetta.unit
- linux.PyRosetta4.python-3.build
- linux.PyRosetta4.python-3.unit
- mac.PyRosetta.build
- mac.PyRosetta.unit
- mac.PyRosetta4.python-2.build
- mac.PyRosetta4.python-2.unit
- windows.PyRosetta.build

### Release
- linux.release.PyRosetta.monolith
- linux.release.PyRosetta.monolith_debug
- linux.release.PyRosetta.namespace
- linux.release.PyRosetta.namespace_debug
- linux.release.PyRosetta4.python2.Debug
- linux.release.PyRosetta4.python2.Release
- linux.release.binary
- mac.release.PyRosetta.monolith
- mac.release.PyRosetta.monolith_debug
- mac.release.PyRosetta.namespace
- mac.release.PyRosetta.namespace_debug
- mac.release.PyRosetta4.python2.Debug
- mac.release.PyRosetta4.python2.Release
- mac.release.binary
- release.source
- ubuntu.release.PyRosetta.monolith
- ubuntu.release.PyRosetta.monolith_debug
- ubuntu.release.binary
How to build?

Install Clang, CMake and Ninja and then:

```bash
% cd main/source/src/python/PyRosetta

% ./build.py -j8

% python3 build.py -j8
```
PyRosetta-4 generated source is available at Benchmark test page:
PyRosseta-4 generated source is available at Benchmark test page:

Test: linux.clang.python3.PyRosetta4.unit
Branch: master 「revision: #58790」
Test files: 「file-system-view」 「file-list-view」
Daemon: Hojo-1 Run time: 0:09:19
State: passed
PyRosetta-4 generated source is available at Benchmark test page:

Test: linux.clang.python3.PyRosetta4.unit
Branch: master 「revision: 658790」
Test files: 「file-system-view」「file-list-view」
Daemon: Hojo-1 Run time: 0:09:19
State: passed

[...] [source] .0.output.log .0.results.json build-log.txt output.json
PyRosetta-4 generated source is available at Benchmark test page:

Test: linux.clang.python3.PyRosetta4.unit
Branch: master 「revision: 58790」
Test files: 「file-system-view」 「file-list-view」
Daemon: Hojo-1 Run time: 0:09:19
State: passed

```
[...]  
[source]   
0.output.log  
0.results.json  
build-log.txt  
output.json
```
PyRosetta-4 generated source is available at Benchmark test page:

Test: linux.clang.python3.PyRosetta4.unit
Branch: master [revision: 158790]
Test files: file-system-view file-list-view
Daemon: Hojo-1  Run time: 0:09:15
State: passed

[...] [source]
0.output.log
0.results.json
build-log.txt
output.json
[...] [carbohydrates]
copydofs
datacache
[full_model_info]
[metrics]
[motif]
[ncbb]
[reference_pose]
rna
[signals]
symmetry
MiniPose.cpp
PDBPoseMap.cpp
Pose.cpp
annotated_sequence.cpp
selection.cpp
util.cpp
util_1.cpp
util_2.cpp
util_tmpl.cpp
xyzStripeHashPose.cpp
xyzStripeHashPose_fwd.cpp
Copyright Rosetta Commons Member Institutions.

Created in HRL by Sergey Lyskov and PyRosetta Team.

Copyright Rosetta Commons Member Institutions.

Copyright Rosetta Commons Member Institutions.

Copyright Rosetta Commons Member Institutions.
Thank you!
Testing Server and various related code issues
Testing Server Capacity
Testing Server Capacity

• Ubuntu dedicated testing servers?
Testing Server Capacity

• Ubuntu dedicated testing servers?

• Expansion, should buy more Testing Servers right now? Which platform: Linux, Mac, …?
General Code issues
General Code issues

- C++11 transition
General Code issues

• C++11 transition

• my_class::to_string() instead of operator<<
General Code issues

• C++11 transition

• my_class::to_string() instead of operator<<

• ban multiple inheritance (and require community for new classes that are going to use multiple inheritance)
General Code issues

- C++11 transition
- my_class::to_string() instead of operator<<
- ban multiple inheritance (and require community for new classes that's going to use multiple inheritance)
- Windows (Py)Rosetta build
General Code issues

- DO NOT USE ‘using namespace ...’ in headers outside of class or function definition!
- How about if we add a test to detect this?
- Would it be ok if we mark the file as ‘failed’ if it was modified?
What features is missing from our Testing platform?
Interesting Challenges

• How to made generated code more ‘rebuild-friendly’?

• Map all implementations of std::<thing> to ‘standard’ namespaces/names