pybind11 Documentation

Wenzel Jakob

Feb 14, 2024

CONTENTS

pybind11 is a lightweight header-only library that exposes C++ types in Python and vice versa, mainly to create Python bindings of existing C++ code. Its goals and syntax are similar to the excellent Boost.Python library by David Abrahams: to minimize boilerplate code in traditional extension modules by inferring type information using compile-time introspection.

The main issue with Boost.Python—and the reason for creating such a similar project—is Boost. Boost is an enormously large and complex suite of utility libraries that works with almost every C++ compiler in existence. This compatibility has its cost: arcane template tricks and workarounds are necessary to support the oldest and buggiest of compiler specimens. Now that C++11-compatible compilers are widely available, this heavy machinery has become an excessively large and unnecessary dependency.

Think of this library as a tiny self-contained version of Boost.Python with everything stripped away that isn't relevant for binding generation. Without comments, the core header files only require ~4K lines of code and depend on Python (3.6+, or PyPy) and the C++ standard library. This compact implementation was possible thanks to some of the new C++11 language features (specifically: tuples, lambda functions and variadic templates). Since its creation, this library has grown beyond Boost.Python in many ways, leading to dramatically simpler binding code in many common situations.

Tutorial and reference documentation is provided at pybind11.readthedocs.io. A PDF version of the manual is available here. And the source code is always available at github.com/pybind/pybind11.

Core features

pybind11 can map the following core C++ features to Python:

- Functions accepting and returning custom data structures per value, reference, or pointer
- Instance methods and static methods
- Overloaded functions
- Instance attributes and static attributes
- Arbitrary exception types
- Enumerations
- Callbacks
- Iterators and ranges
- Custom operators
- Single and multiple inheritance
- STL data structures
- Smart pointers with reference counting like `std::shared_ptr`
- Internal references with correct reference counting
- C++ classes with virtual (and pure virtual) methods can be extended in Python

Goodies

In addition to the core functionality, pybind11 provides some extra goodies:

- Python 3.6+, and PyPy3 7.3 are supported with an implementation-agnostic interface (pybind11 2.9 was the last version to support Python 2 and 3.5).
- It is possible to bind C++11 lambda functions with captured variables. The lambda capture data is stored inside the resulting Python function object.
- pybind11 uses C++11 move constructors and move assignment operators whenever possible to efficiently transfer custom data types.

- It's easy to expose the internal storage of custom data types through Pythons' buffer protocols. This is handy e.g.for fast conversion between C++ matrix classes like Eigen and NumPy without expensive copy operations.
- pybind11 can automatically vectorize functions so that they are transparently applied to all entries of one or more NumPy array arguments.
- Python's slice-based access and assignment operations can be supported with just a few lines of code.
- Everything is contained in just a few header files; there is no need to link against any additional libraries.
- Binaries are generally smaller by a factor of at least 2 compared to equivalent bindings generated by Boost.Python. A recent pybind11 conversion of PyRosetta, an enormous Boost.Python binding project, reported a binary size reduction of **5.4x** and compile time reduction by **5.8x**.
- Function signatures are precomputed at compile time (using `constexpr`), leading to smaller binaries.
- With little extra effort, C++ types can be pickled and unpickled similar to regular Python objects.

Supported compilers

1. Clang/LLVM 3.3 or newer (for Apple Xcode's clang, this is 5.0.0 or newer)
2. GCC 4.8 or newer
3. Microsoft Visual Studio 2017 or newer
4. Intel classic C++ compiler 18 or newer (ICC 20.2 tested in CI)
5. Cygwin/GCC (previously tested on 2.5.1)
6. NVCC (CUDA 11.0 tested in CI)
7. NVIDIA PGI (20.9 tested in CI)

About

This project was created by Wenzel Jakob. Significant features and/or improvements to the code were contributed by Jonas Adler, Lori A. Burns, Sylvain Corlay, Eric Cousineau, Aaron Gokaslan, Ralf Grosse-Kunstleve, Trent Houliston, Axel Huebl, @hulucc, Yannick Jadoul, Sergey Lyskov, Johan Mabille, Tomasz Miąsko, Dean Moldovan, Ben Pritchard, Jason Rhinelander, Boris Schäling, Pim Schellart, Henry Schreiner, Ivan Smirnov, Boris Staletic, and Patrick Stewart.

We thank Google for a generous financial contribution to the continuous integration infrastructure used by this project.

Contributing

See the contributing guide for information on building and contributing to pybind11.

License

pybind11 is provided under a BSD-style license that can be found in the LICENSE file. By using, distributing, or contributing to this project, you agree to the terms and conditions of this license.

CHAPTER

ONE

CHANGELOG

Starting with version 1.8.0, pybind11 releases use a semantic versioning policy.

Changes will be added here periodically from the "Suggested changelog entry" block in pull request descriptions.

1.1 IN DEVELOPMENT

Changes will be summarized here periodically.

1.2 Version 2.11.1 (July 17, 2023)

Changes:

- `PYBIND11_NO_ASSERT_GIL_HELD_INCREF_DECREF` is now provided as an option for disabling the default-on `PyGILState_Check()`'s in `pybind11::handle`'s `inc_ref()` & `dec_ref()`. #4753
- `PYBIND11_ASSERT_GIL_HELD_INCREF_DECREF` was disabled for PyPy in general (not just PyPy Windows). #4751

1.3 Version 2.11.0 (July 14, 2023)

New features:

- The newly added `pybind11::detail::is_move_constructible` trait can be specialized for cases in which `std::is_move_constructible` does not work as needed. This is very similar to the long-established `pybind11::detail::is_copy_constructible`. #4631
- Introduce `recursive_container_traits`. #4623
- `pybind11/type_caster_pyobject_ptr.h` was added to support automatic wrapping of APIs that make use of `PyObject *`. This header needs to included explicitly (i.e. it is not included implicitly with `pybind/pybind11.h`). #4601
- `format_descriptor<>` & `npy_format_descriptor<>` `PyObject *` specializations were added. The latter enables `py::array_t<PyObject *>` to/from-python conversions. #4674
- `buffer_info` gained an `item_type_is_equivalent_to<T>()` member function. #4674
- The `capsule` API gained a user-friendly constructor (`py::capsule(ptr, "name", dtor)`). #4720

Changes: