GIANNI VALENTI
Software Developer

C++ and Python integration with Boost Python

Develer webinar

05 Oct 2022

develer

INTRODUCTION

In this webinar I will show you how to create a Python module using the Boost Python library to integrate C++ code, with emphasis on memory management and conversion between types.

Boost Python is an open source library that is part of the Boost project and that «enables seamless interoperability between C++ and the Python programming language».

💡 More information can be found here.

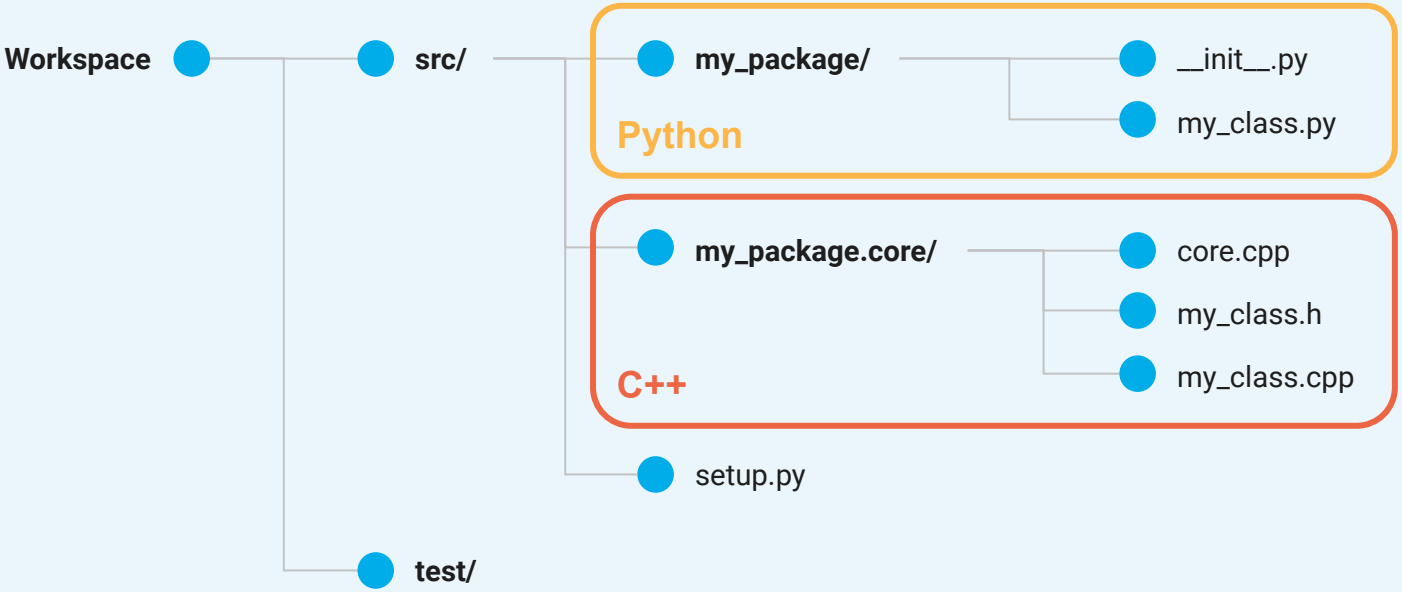
ALTERNATIVES TO BOOST PYTHON

- **Cython** - the source code gets translated into optimized C/C++ code and compiled as Python extension.
- **Ctypes** - a Python extension that allows you to call an arbitrary function in a shared library.
- ⭐ **pybind11** - «a tiny self-contained header-only version of Boost Python» distributed under the BSD license.

1. My First Class

How can I expose a C++ class to Python?

1. My First Class - Directory Structure



WHY A PYTHON WRAPPER?

- With a Python wrapper you can mix Python code and C++ code together.
- You can take advantage of C++ optimizations just for methods that need to be very fast.
- You can add type hints so that Python linters can better understand the code. PyCharm creates .pyi files from introspection, VSCode doesn't.

1. My First Class - C++ object definition

my_class.h

```
#pragma once

class my_class {
public:

    void say_hello() const;
};
```

my_class.cpp

C++

```
#include "my_class.h"

#include <iostream>

void my_class::say_hello() const
{
    std::cout << "Hello there! I'm a C++ method."
              << std::endl;
}
```

1. My First Class - core.cpp

C++

```cpp
#include <boost/python.hpp>

#include "my_class.h"

BOOST_PYTHON_MODULE (core)
{
    boost::python::class_<my_class> class_("my_class");
    class_.def("say_hello", &my_class::say_hello);
}
```

exposing classes

1. My First Class - my_class.py

Python

```python
from . import core

class my_class:
    def __init__(self):
        self._my_class = core.my_class()

    def say_hello(self) -> None:
        self._my_class.say_hello()
```

1. My First Class - __init__.py

Python

```python
from .my_class import my_class
```

1. My First Class - setup.py

Python

```python
import setuptools
import glob

setuptools.setup(
    name="my_package",
    version="1.0",
    packages=["my_package"],
    ext_modules=[
        setuptools.Extension(
            "my_package.core",
            sources=glob.glob("my_package.core/*.cpp"),
            libraries=["boost_python310"]
        )
    ]
)
```

building extension modules

1. My First Class - Let's try it out!

Shell

```
boost-python-workshop#  pip install src/
Processing ./src
  Preparing metadata (setup.py) ... done
Building wheels for collected packages: my-package
  Building wheel for my-package (setup.py) ... done
  Created wheel for my-package: filename=my_package-1.0-cp310-cp310-linux_x86_64.whl size=574149
sha256=8ea6c75f049fad6cd1f6aabb3f2c854330f2b2a3b6ead8b66d9949d3f12f69e5
  Stored in directory:
/tmp/pip-ephem-wheel-cache-sk2ps7pu/wheels/d9/ac/9e/f771bab4c4ce961f8bf065e31b7db637dccc0fde82f869bb
68
Successfully built my-package
Installing collected packages: my-package
Successfully installed my-package-1.0
boost-python-workshop#  python
Python 3.10.4 (main, Apr  2 2022, 09:04:19) [GCC 11.2.0] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> from my_package import my_class
>>> a = my_class()
>>> a.say_hello()
Hello there! I'm a C++ method.
>>>
```

C++ OBJECT LIFE CYCLE

- The C++ object is constructed when the **Python init** method is called.
- When is the C++ destructor called by the garbage collector?
- To implement RAII patterns you should implement Python context manager methods (enter and exit) and call the appropriate C++ methods.

2. Adding Properties

How can I expose a property to Python?

2. Adding Properties - my_class.h

C++

```
#pragma once

class my_class {
public:
    // [...]

    const char *get_name() const;
    void set_name(const char *name);

    // [...]
};
```

2. Adding Properties - core.cpp

C++

```cpp
#include <boost/python.hpp>

#include "my_class.h"

BOOST_PYTHON_MODULE (core)
{
    boost::python::class_<my_class> class_("my_class");
    class_.add_property("name", &my_class::get_name, &my_class::set_name);
}
```

exposing classes

2. Adding Properties - my_class.py

Python

```python
from . import core

class my_class:
    def __init__(self):
        self._my_class = core.my_class()

    @property
    def name(self) -> str:
        return self._my_class.name

    @name.setter
    def name(self, value) -> None:
        self._my_class.name = value
```

2. Adding Properties - Let's try it out!

Shell

```
boost-python-workshop#python
Python 3.10.4 (main, Apr  2 2022, 09:04:19) [GCC 11.2.0] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> from my_package import my_class
>>> a = my_class()
>>> type(a.name)
<class 'str'>
>>> a.name
''
>>> a.name = "John Coltrane"
>>> a.name
'John Coltrane'
>>>
```

a C++ const char pointer is automatically converted to a Python string

the conversion works both ways

3. Convert C++ Pointer to MemoryView

How can I expose a read-only pointer from C++ to Python?

3. Convert C++ Pointer to MemoryView - my_view.h

C++

```cpp
#pragma once

#include <stddef.h>
#include <utility>

class my_view : std::pair<const char *, size_t> {
public:
    my_view(const char *ptr, size_t size)
        : std::pair<const char *, size_t>(ptr, size)
    {
    }

    const char *ptr() const { return first; }

    size_t size() const { return second; }
};
```

3. Convert C++ Pointer to MemoryView - core.cpp

C++

```
// [...]

struct my_view_to_memoryview
{
    static PyObject *convert(const my_view &view)
    {
        return PyMemoryView_FromMemory(const_cast<char *>(view.ptr()),
                                       view.size(),
                                       PyBUF_READ);
    }
};

// [...]
```

to_python_converter, memory view, ownership rules

3. Convert C++ Pointer to MemoryView - core.cpp (2)

C++

```cpp
#include <boost/python.hpp>

#include "my_class.h"

// [...]

BOOST_PYTHON_MODULE (core)
{
    // [...]

    boost::python::to_python_converter <my_view, my_view_to_memoryview >();
}
```

to_python_converter

3. Convert C++ Pointer to MemoryView - C++ object definition

my_class.h

```cpp
#pragma once

#include <string>
#include "my_view.h"

class my_class {
public:
    // [...]

    my_view get_name_ptr() const;

private:
    std::string m_name {};
};
```

my_class.cpp

C++

```cpp
#include "my_class.h"

// [...]

my_view my_class::get_name_ptr() const
{
    return { m_name.c_str(), m_name.size() };
}
```

3. Convert C++ Pointer to MemoryView - core.cpp (3)

C++

```cpp
#include <boost/python.hpp>

#include "my_class.h"

// [...]

BOOST_PYTHON_MODULE (core)
{
    boost::python::class_<my_class> class_("my_class");
    class_.add_property("name", &my_class::get_name, &my_class::set_name);
    class_.add_property("name_ptr", &my_class::get_name_ptr);

    boost::python::to_python_converter<my_char_ptr, my_char_ptr_to_memoryview>();
}
```

to_python_converter

3. Convert C++ Pointer to MemoryView - my_class.py

Python

```python
from . import core

class my_class:
    def __init__(self):
        self._my_class = core.my_class()

    # [...]

    @property
    def name_ptr(self) -> memoryview:
        return self._my_class.name_ptr
```

3. Convert C++ Pointer to MemoryView - Let's try it out!

Shell

```
boost-python-workshop#  python
Python 3.10.4 (main, Apr  2 2022, 09:04:19) [GCC 11.2.0] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> from my_package import my_class
>>> a = my_class()
>>> a.name = "John Coltrane"
>>> type(a.name_ptr)
<class 'memoryview'>
>>> a.name_ptr
<memory at 0x7f32916f8580>
>>> bytes(a.name_ptr)
b'John Coltrane'
>>>
```

4. Convert C++ Byte Array to and from Bytes

How can I expose a C++ array of bytes to Python?
How can I read a Python bytes object?

4. Convert C++ Byte Array to and from Bytes - my_byte_array.h

C++

```cpp
#pragma once

#include <cstdint>
#include <vector>

class my_byte_array : public std::vector<uint8_t> {
public:
    using std::vector<uint8_t>::vector;
};
```

4. Convert C++ Byte Array to and from Bytes - core.cpp

C++

```cpp
// [...]

struct my_byte_array_to_bytes
{
    static PyObject *convert(const my_byte_array &array)
    {
        return PyBytes_FromStringAndSize(reinterpret_cast<const char *>(array.data()),
                                         array.size());
    }
};

// [...]
```

📘 to_python_converter, bytes objects, ownership rules

4. Convert C++ Byte Array to and from Bytes - core.cpp (2)

C++

```cpp
// [...]

struct my_byte_array_from_bytes
{
    static void *convertible(PyObject *py_obj)
    {
        Py_buffer py_buffer  = {};
        if (PyObject_GetBuffer(py_obj, &py_buffer, PyBUF_FORMAT)  != 0)
            return nullptr;

        PyBuffer_Release(&py_buffer);
        return py_obj;
    }

    // [...]
```

convert custom types, buffer protocol

4. Convert C++ Byte Array to and from Bytes - core.cpp (3)

C++

```
    // [...]

    static void construct(PyObject* py_obj,
                          boost::python::converter::rvalue_from_python_stage1_data * data)
    {
        using rvalue_from_python_storage  =
                     boost::python::converter::rvalue_from_python_storage <my_byte_array>;
        auto rvalue = reinterpret_cast< rvalue_from_python_storage  *>(data);

        Py_buffer py_buffer  = {};
        PyObject_GetBuffer(py_obj, &py_buffer, PyBUF_FORMAT) ;
        auto cpp_obj  = new (rvalue->storage.bytes)my_byte_array(py_buffer.len);
        PyBuffer_ToContiguous(cpp_obj->data(), &py_buffer, py_buffer.len, 'C');
        PyBuffer_Release(&py_buffer);

        data->convertible  = storage;
    }
}; // struct my_byte_array_from_bytes
```

convert custom types, buffer protocol, boost python errors

4. Convert C++ Byte Array to and from Bytes - core.cpp (4)

C++

```cpp
#include <boost/python.hpp>

#include "my_class.h"

// [...]

BOOST_PYTHON_MODULE (core)
{
    // [...]

    boost::python::to_python_converter <my_byte_array, my_byte_array_to_bytes >();
    boost::python::converter::registry::push_back(&my_byte_array_from_bytes ::convertible,
                                                  &my_byte_array_from_bytes ::construct,
                                                  boost::python::type_id<my_byte_array >());
}
```

convert custom types, buffer protocol, boost python errors

4. Convert C++ Byte Array to and from Bytes - C++ object definition

my_class.h

```cpp
#pragma once

#include <string>
#include "my_byte_array.h"

class my_class {
public:
    // [...]

    my_byte_array get_name_as_bytes() const;
    void set_name_as_bytes(const my_byte_array &name);

private:
    std::string m_name {};
};
```

my_class.cpp

C++

```cpp
#include "my_class.h"

// [...]

my_byte_array my_class::get_name_as_bytes() const
{
    const auto begin =
        reinterpret_cast<const uint8_t *>(m_name.c_str());
    return my_byte_array(begin, begin + m_name.length());
}

void my_class::set_name_as_bytes(const my_byte_array
&name)
{
    const auto begin =
        reinterpret_cast<const char *>(name.data());
    m_name = std::string(begin, name.size());
}
```

4. Convert C++ Byte Array to and from Bytes - core.cpp (5)

C++

```cpp
#include <boost/python.hpp>

#include "my_class.h"
#include "my_byte_array.h"

// [...]

BOOST_PYTHON_MODULE(core)
{
    boost::python::class_<my_class> class_("my_class");
    class_.add_property("name", &my_class::get_name, &my_class::set_name);
    class_.add_property("name_as_bytes", &my_class::get_name_as_bytes,
&my_class::set_name_as_bytes);

    boost::python::to_python_converter<my_byte_array, my_byte_array_to_bytes>();
    boost::python::converter::registry::push_back(&my_byte_array_from_bytes::convertible,
                                                  &my_byte_array_from_bytes::construct,
                                                  boost::python::type_id<my_byte_array>());
}
```

convert custom types, buffer protocol, boost python errors

4. Convert C++ Byte Array to and from Bytes - my_class.py

Python

```python
class my_class:
    def __init__(self):
        from . import core
        self._my_class = core.my_class()

    # [...]

    @property
    def name_as_bytes(self) -> bytes:
        return self._my_class.name_as_bytes

    @name_bytes.setter
    def name_as_bytes(self, value) -> None:
        self._my_class.name_as_bytes = value
```

4. Convert C++ Byte Array to and from Bytes - Let's try it out!

Shell

```
boost-python-workshop#  python
Python 3.10.4 (main, Apr  2 2022, 09:04:19) [GCC 11.2.0] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> from my_package import my_class
>>> a = my_class()
>>> a.name_as_bytes = b'John Coltrane'
>>> a.name
'John Coltrane'
>>> a.name_as_bytes
b'John Coltrane'
>>> a.name = "Miles Davis"
>>> a.name_bytes
b'Miles Davis'
>>>
```

5. Translating Exceptions

How can I throw a C++ exception letting Python handle the error?

5. Translating Exceptions - C++ object definition

my_exception.h

```cpp
#pragma once

#include <exception>
#include <string>

class my_exception : public std::exception
{
public:

    explicit my_exception(const std::string &what);

    // [...]

    static void translate(const my_exception &e);
};
```

my_exception.cpp

C++

```cpp
#include "my_exception.h"

#include <boost/python/exception_translator.hpp>

// [...]

void my_exception::translate(const my_exception &e)
{
    // Use the Python 'C' API to set up an exception
object.
    PyErr_SetString(PyExc_RuntimeError, e.what());
}
```

exception translator, python exceptions

5. Translating Exceptions - core.cpp

C++

```cpp
#include <boost/python.hpp>
#include <boost/python/exception_translator.hpp>

// [...]

#include "my_exception.h"

// [...]

BOOST_PYTHON_MODULE (core)
{
    // [...]

    boost::python::register_exception_translator <my_exception>(&my_exception::translate);
}
```

exception translator

5. Translating Exceptions - C++ object definition

my_class.h

```cpp
#pragma once

// [...]

class my_class {
public:

    // [...]

    void sabotage();

    // [...]
};
```

my_class.cpp

C++

```cpp
#include "my_class.h"
#include "my_exception.h"

// [...]

void my_class::sabotage()
{
    throw my_exception("This is my error string");
}
```

5. Translating Exceptions - core.cpp (2)

C++

```cpp
#include <boost/python.hpp>

#include "my_class.h"

// [...]

BOOST_PYTHON_MODULE (core)
{
    boost::python::class_<my_class> class_("my_class");
    class_.def("sabotage", &my_class::sabotage);

    // [...]
}
```

exposing classes

5. Translating Exceptions - my_class.py

Python

```python
from . import core

class my_class:
    def __init__(self):
        self._my_class = core.my_class()

    # [...]

    def sabotage(self) -> None:
        self._my_class.sabotage()
```

5. Translating Exceptions - Let's try it out!

Shell

```
boost-python-workshop#  python
Python 3.10.4 (main, Apr  2 2022, 09:04:19) [GCC 11.2.0] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> from my_package import my_class
>>> a = my_class()
>>> a.sabotage()
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
  File "/usr/local/lib/python3.10/dist-packages/my_package/my_class.py", line 39, in sabotage
    self._my_class.sabotage()
RuntimeError: This is my error string
>>>
```

Thank You!

Gianni Valenti

gvalenti@develer.com

Want to stay up-to-date on Develer events?
Follow us on our social channels:

develer

www.develer.com