cpptools

Thorsten Beier

Jul 25, 2022

BASICS

cpptools is a modern C++ library

CHAPTER

ONE

LICENSING

This software is licensed under the MIT license license. See the LICENSE.txt file for details.

1.1 Features

- CMake build system
- C++ Unit Tests with Doctest
- Benchmark code with google benchmark
- **Continous Integration:**
  - azure-pipelines
  - circle-ci
  - travis-ci
- bumpversion for version handling
- Documentation with sphinx breathe and on readthedocs
- Conda Recipe Included
- Python bindings are created via pybind11

1.2 Basic Usage

```
cd cpptools
conda env create -f cpptools-dev-requirements.yml
source activate cpptools-dev-requirements
mkdir build
cd build
cmake ..
make -j2
make cpp-test
make python-test
make install
cd examples
./hello_world
cd ..
```

(continues on next page)

(continued from previous page)

```
cd benchmark
./benchmark_cpptools
```

On a windows machine this looks like:

```
cd cpptools
conda env create -f cpptools-dev-requirements.yml
call activate cpptools-dev-requirements
mkdir build
cd build
cmake .. -G"Visual Studio 15 2017 Win64" -DCMAKE_BUILD_TYPE=Release  ^
      -DDEPENDENCY_SEARCH_PREFIX="%CONDA_PREFIX%\Library" -DCMAKE_PREFIX_PATH="%CONDA_
↪PREFIX%\Library"
call activate cpptools-dev-requirements
cmake --build . --target ALL_BUILD
cmake --build . --target python-test
cmake --build . --target cpp-test
cmake --build . --target install
```

1.3 Folder Structure

The generated project has the following folder structure

```
cpptools
   ├──azure-pipelines.yml                                  # Ci script
   │
   ├──benchmark                                            # C++ benchmark code
   │    └── ...
   │
   ├──binder                                               # dockerfile for mybinder.org
   │    └── Dockerfile
   │
   ├──cmake                                                # Cmake script/modules
   │    └── ...
   │
   ├──CMakeLists.txt                                       # Main cmake list
   │
   ├──CONTRIBUTING.rst                                     # Introduction how to constribute
   │
   ├──cpptoolsConfig.cmake.in # Script to make find_package(...)
                                                           # work for this package
   │
   ├──cpptools.pc.in          # Packaging info
   │
   ├──cpptools-dev-requirements.yml # List of development conda dependencies
   │
   ├──docker                                                # dockerfile for dockerhub
   │    └── Dockerfile
   ├──docs                                                 # Sources for sphinx␣
↪documentation
```

(continues on next page)

(continued from previous page)

```
   │   └── ...
   │
   ├──examples                                          # C++ examples
   │   └── ...
   │
   ├──include                                           # C++ include directory for this␣
↪folder
   │   └── ...
   │
   ├──LICENCE.txt                                       # License file
   │
   ├──python                                            # Python binding source code
   │   └── ...
   │
   ├──README.rst                                        # Readme shown on github
   │
   ├──readthedocs.yml                                   # Entry point for automated
   │                                                    # documentation build on␣
↪readthedocs.org
   │
   ├──recipe                                            # Folder for conda recipes
   │   └── ...
   │
   └──test                                              # Folder containing C++ unit␣
↪tests
       └── ...
```

1.4 Unit Tests

We use doctest to create a benchmark for the C++ code.

The test subfolder contains all the code related to the C++ unit tests. In `main.cpp` implements the benchmarks runner, The unit tets are implemented in `test_*.cpp`. The test older looks like.

```
cpptools
├── ...
├── test
│   ├── CMakeLists.txt
│   ├── main.cpp
│   ├── test_cpptools_config.cpp
├── ...
```

1.4.1 Build System

There is a meta target called `test_cpptools` which bundles the build process of unit tests. Assuming you cmake-build directory is called `bld` the following will build all examples.

```
$ cd bld
$ make test_cpptools
```

To run the actual test you can use the target `cpp_tests` .. code-block:: shell

$ cd bld $ make cpp_tests

1.4.2 Adding New Tests

To add new tests just add a new cpp file to the test folder and update the `CMakeLists.txt`. Assuming we named the new cpp file `test_my_new_feture.cpp`, the relevant part in the `CMakeLists.txt` shall look like this:

```
# all tests
set(${PROJECT_NAME}_TESTS
    test_cpptools_config.cpp
    test_my_new_feture.cpp
)
```

After changing the `CMakeLists.txt` cmake needs to be rerun to configure the build again. After that `make examples` will build all examples including the freshly added examples.

```
$ cd bld
$ cmake .
$ make examples
```

1.5 Benchmark

We use gbench to create a benchmark for the C++ code.

The benchmark subfolder contains all the code related to the benchmarks. In main.cpp the actual benchmarks are implemented.

```
cpptools
├── ...
├── benchmark
│   ├── main.cpp
├── ...
```

1.6 Python Module

1.6.1 Folder Structure

We use pybind11 to create the python bindings. The python subfolder contains all the code related to the python bindings. The `module/cpptools` subfolder contains all the `*.py` files of the module. The src folder contains the `*.cpp` files used to export the C++ functionality to python via pybind11. The `test` folder contains all python tests.

```
cpptools
├── ...
├── python
│   ├── module
│   │   └── cpptools
│   │       ├── __init__.py
│   │       └── ...
│   ├── src
│   │   ├── CMakeLists.txt
│   │   ├── main.cpp
│   │   ├── def_build_config.cpp
│   │   ├── ...
│   └── test
│       ├── test_build_configuration.py
│       └── ...
└── ...
```

1.6.2 Build System

To build the python package use the `python-module` target.

```
make python-module
```

This will build the `*.cpp` files in the `src` folder and copy the folder `module/cpptools` folder to build location of the python module, namely `${CMAKE_BINARY_DIR}/python/module/` where `${CMAKE_BINARY_DIR}` is the build directory.

1.6.3 Usage

After a successfully building and installing the python module can be imported like the following:

```
import cpptools

config = cpptools.BuildConfiguration
print(config.VERSION_MAJOR)
```

1.6.4 Run Python Tests

To run the python test suite use the *python-test* target:

```
make python-test
```

1.6.5 Adding New Python Functionality

We use pybind11 to export functionality from C++ to Python. pybind11 can create modules from C++ without the use of any `*.py` files. Nevertheless we prefer to have a regular Python package with a proper `__init__.py`. From the `__init__.py` we import all the C++ / pybind11 exported functionality from the build submodule named `_cpptools`. This allows us to add new functionality in different ways:

- new functionality from c++ via pybind11
- new puren python functionality

Add New Python Functionality from C++

To export functionality from C++ to python via pybind11 it is good practice to split functionality in multiple `def_*.cpp` files. This allow for readable code, and parallel builds. To add news functionality we create a new file, for example `def_new_stuff.cpp`.

```
#include "pybind11/pybind11.h"
#include "pybind11/numpy.h"

#include <iostream>
#include <numeric>

#define FORCE_IMPORT_ARRAY
#include "xtensor-python/pyarray.hpp"
#include "xtensor-python/pytensor.hpp"

// our headers
#include "cpptools/cpptools.hpp"

namespace py = pybind11;


namespace cpptools {

    void def_new_stuff(py::module & m)
    {
        py::def('new_stuff',[](xt::pytensor<1,double> values){
            return values * 42.0;
        });
    }

}
```

Next we need to declare and call the `def_new_stuff` from `main.cpp`. To declare the function modify the following block in `main.cpp`

```
namespace cpptools {

    // ....
    // ....
    // ....

    // implementation in def_myclass.cpp
    void def_class(py::module & m);

    // implementation in def_myclass.cpp
    void def_build_config(py::module & m);

    // implementation in def.cpp
    void def_build_config(py::module & m);

    // implementation in def.cpp
    void def_build_config(py::module & m);

    // implementation in def_new_stuff.cpp
    void def_new_stuff(py::module & m);             // <- our new functionality

}
```

After declaring the function `def_new_stuff`, we can call `def_new_stuff`. We modify the `PYBIND11_MODULE` in code:*main.cpp*:

```
// Python Module and Docstrings
PYBIND11_MODULE(_cpptools , module)
{
    xt::import_numpy();

    module.doc() = R"pbdoc(
        _cpptools  python bindings

        .. currentmodule:: _cpptools

        .. autosummary::
           :toctree: _generate

           BuildConfiguration
           MyClass
           new_stuff
    )pbdoc";

    cpptools::def_build_config(module);
    cpptools::def_class(module);
    cpptools::def_new_stuff(module);  // <- our new functionality

    // make version string
    std::stringstream ss;
    ss<<CPPTOOLS_VERSION_MAJOR<<"."
      <<CPPTOOLS_VERSION_MINOR<<"."
      <<CPPTOOLS_VERSION_PATCH;
```

(continues on next page)

(continued from previous page)

```
    module.attr("__version__") = ss.str();
}
```

We need to add this file to the `CMakeLists.txt` file at `{cookiecutter.github_project_name}}/python/src/CMakeLists.txt` The file needs to be passed as an argument to the `pybind11_add_module` function.

```
# add the python library
pybind11_add_module(${PY_MOD_LIB_NAME}
    main.cpp
    def_build_config.cpp
    def_myclass.cpp
    def_new_stuff.cpp  # <- our new functionality
)
```

Now we are ready to build the freshly added functionality.

```
make python-test
```

After a successful build we can use the new functionality from python.

```
import numpy as np
import cpptools

cpptools.new_stuff(numpy.arange(5), dtype='float64')
```

Add New Pure Python Functionality

To add new pure Python functionality, just add the desired function / classes to a new `*.py` file and put this file to the `module/cpptools` subfolder. After adding the new file, cmake needs to be rerun since we copy the content `module/cpptools` during the build process.

1.6.6 Adding New Python Tests

We use pytest as python test framework. To add new tests, just add new `test_*.py` files to the test subfolder. To run the actual test use the `python-test` target

```
make python-test
```

1.7 Examples

1.7.1 Folder Structure

The examples subfolder contains C++ examples which shall show the usage of the C++ library.

```
cpptools
├── ...
├── examples
│   ├── CMakeLists.txt
```

(continues on next page)

(continued from previous page)

```
│   ├── hello_world.cpp
├── ...
```

1.7.2 Build System

There is a meta target called `examples` which bundles the build process of all cpp files in the folder examples in one target. Assuming you cmake-build directory is called `bld` the following will build all examples.

```
$ cd bld
$ make examples
```

1.7.3 Adding New Examples

To add new examples just add a new cpp file to the example folder and update the `CMakeLists.txt`. Assuming we named the new cpp file `my_new_example.cpp`, the relevant part in the `CMakeLists.txt` shall look like this:

```
# all examples
set(CPP_EXAMPLE_FILES
   hello_world.cpp
   my_new_example.cpp
)
```

After changing the `CMakeLists.txt` cmake needs to be rerun to configure the build again. After that `make examples` will build all examples including the freshly added examples.

```
$ cd bld
$ cmake .
$ make examples
```

1.8 Conda Recipe

The recipe subfolder contains all the code related to the conda recipe

```
project
├── ...
├── recipe
│   ├── bld.bat
│   ├── build.sh
│   ├── meta.tml
├── ...
```

1.9 cpptools API

1.9.1 Class Hierarchy

1.9.2 File Hierarchy

1.9.3 Full API

Namespaces

Namespace cpptools

Contents

Classes

Classes and Structs

Class MyClass

- Defined in file__home_docs_checkouts_readthedocs.org_user_builds_cpptools_checkouts_latest_include_cpptools_cpptools.hpp

Class Documentation

class **MyClass**

Public Functions

inline **MyClass**(const uint64_t size)

inline void **hello_world**()

Defines

Define CPPTOOLS_CPPTOOLS_CONFIG_HPP

- Defined in file__home_docs_checkouts_readthedocs.org_user_builds_cpptools_checkouts_latest_include_cpptools_cpptools_conf

Define Documentation

```
CPPTOOLS_CPPTOOLS_CONFIG_HPP
```

Define CPPTOOLS_CPPTOOLS_HPP

- Defined in file__home_docs_checkouts_readthedocs.org_user_builds_cpptools_checkouts_latest_include_cpptools_cpptools.hpp

Define Documentation

```
CPPTOOLS_CPPTOOLS_HPP
```

Define CPPTOOLS_CPPTOOLS_VERSION_MAJOR_HPP

- Defined in file__home_docs_checkouts_readthedocs.org_user_builds_cpptools_checkouts_latest_include_cpptools_cpptools_vers

Define Documentation

```
CPPTOOLS_CPPTOOLS_VERSION_MAJOR_HPP
```

Define CPPTOOLS_CPPTOOLS_VERSION_MINOR_HPP

- Defined in file__home_docs_checkouts_readthedocs.org_user_builds_cpptools_checkouts_latest_include_cpptools_cpptools_vers

Define Documentation

```
CPPTOOLS_CPPTOOLS_VERSION_MINOR_HPP
```

Define CPPTOOLS_CPPTOOLS_VERSION_PATCH_HPP

- Defined in file__home_docs_checkouts_readthedocs.org_user_builds_cpptools_checkouts_latest_include_cpptools_cpptools_vers

Define Documentation

```
CPPTOOLS_CPPTOOLS_VERSION_PATCH_HPP
```

Define CPPTOOLS_VERSION_MAJOR

- Defined in file__home_docs_checkouts_readthedocs.org_user_builds_cpptools_checkouts_latest_include_cpptools_cpptools_vers

Define Documentation

```
CPPTOOLS_VERSION_MAJOR
```

Define CPPTOOLS_VERSION_MINOR

- Defined in file__home_docs_checkouts_readthedocs.org_user_builds_cpptools_checkouts_latest_include_cpptools_cpptools_vers

Define Documentation

```
CPPTOOLS_VERSION_MINOR
```

Define CPPTOOLS_VERSION_PATCH

- Defined in file__home_docs_checkouts_readthedocs.org_user_builds_cpptools_checkouts_latest_include_cpptools_cpptools_vers

Define Documentation

```
CPPTOOLS_VERSION_PATCH
```

1.10 cpptools

1.10.1 cpptools package

Submodules

cpptools._cpptools module

_cpptools python bindings

class cpptools._cpptools.**BuildConfiguration**
Bases: pybind11_builtins.pybind11_object

This class show the compile/build configuration Of cpptools

DEBUG = True

VERSION_MAJOR = 0

VERSION_MINOR = 1

VERSION_PATCH = 0

class cpptools._cpptools.**MyClass**
Bases: pybind11_builtins.pybind11_object

hello_world(*self:* cpptools._cpptools.MyClass) → None

Module contents

class cpptools.**BuildConfiguration**
Bases: pybind11_builtins.pybind11_object

This class show the compile/build configuration Of cpptools

DEBUG = True

VERSION_MAJOR = 0

VERSION_MINOR = 1

VERSION_PATCH = 0

class cpptools.**MyClass**
Bases: pybind11_builtins.pybind11_object

hello_world(*self:* cpptools._cpptools.MyClass) → None

cpptools.**pure_python**()
hello

PYTHON MODULE INDEX

C

INDEX