Software Release Guide (CWI CI Group)

Jan-Willem Buurlage, Allard Hendriksen

October 31, 2018

Contents

1 Coding standards

1.1 Python

1.1.1 PEP8

1.1.2 yapf

1.2 C/C++

1.2.1 `clang-format`

1.2.2 `clang-tidy`

1.2.3 Sensible compile flags

1. `-Wall`
2. `-Werror`
3. `-Wfatal`

4. ...

2 Distributing software

2.1 Python

- `distutil`
- How to define and distribute a conda package

2.2 C/C++

1. Modern CMake
   (a) C++ Weekly, Intro to CMake
   (b) CMakePrimer (LLVM)
   (c) CppCon 2017: Mathieu Ropert "Using Modern CMake Patterns to Enforce a Good Modular Design"
   (d) C++Now 2017: Daniel Pfeifer "Effective CMake"
   (e) Dependency management CMake/Git Example:

```
find_package(ZeroMQ QUIET)

if (ZeroMQ_FOUND)
    add_library(zmq INTERFACE)
    target_include_directories(zmq INTERFACE ${ZeroMQ_INCLUDE_DIR})
    target_link_libraries(zmq INTERFACE ${ZeroMQ_LIBRARY})
else()
    message("'zmq' not installed on the system, building from source...")

    execute_process(COMMAND git submodule update --init --remote -- ext/libzm
WORKING_DIRECTORY ${CMAKE_SOURCE_DIR})

    set(ZMQ_BUILD_TESTS OFF CACHE BOOL "disable tests" FORCE)
    set(WITH_PERF_TOOL OFF CACHE BOOL "disable perf-tools" FORCE)
    add_subdirectory(${CMAKE_SOURCE_DIR}/ext/libzmq)
    set(ZMQ_INCLUDE_DIR ${CMAKE_SOURCE_DIR}/ext/libzmq/include)

    # ZeroMQ names their target libzmq, which is inconsistent => create a gho
    add_library(zmq INTERFACE)
```

```
        target_link_libraries(zmq INTERFACE libzmq)
    endif()
```

   (f) `https://foonathan.net/blog/2018/10/17/cmake-warnings.html`

2. Dynamically linked dependencies Three places that a binary looks for shared dependencies

   (a) `LD_LIBRARY_PATH`
   (b) `rpath` encoded in binary
   (c) system default paths

   Danger of (1) is that it overrides the specific dependencies of all binaries run.

   For shared systems, or non-root users, (3) can be a problem.

   For 2 you proceed as follows:

   - set `LD_RUN_PATH` to something hardcoded
   - use `-R` in gcc

   To check the `RPATH` in a binary on Linux, use `readelf -d <binary>`.

   To list all dynamic dependencies, use `ldd <binary>`

   See also: `https://www.eyrie.org/~eagle/notes/rpath.html`.

3. Python bindings

   (a) `pybind11` Adding Python bindings to C++ code is straightforward with pybind11. A good setup is as follows. (All relative to the root folder of the C++ project, which I call `your_project` here)

      i. Add pybind11 as a git submodule

      ```
      git submodule add https://github.com/pybind/pybind11.git ext/pybind11
      ```

      ii. Set up the Python bindings Make a directory `python`, containing at least three files:

         A. `python/src/module.cpp` This contains the actual bindings, an example is like this:

         ```
         #include <pybind11/pybind11.h>
         namespace py = pybind11;
         ```

```
#include "your_project/your_project.hpp"

using namespace your_project;

PYBIND11_MODULE(py_your_project, m) {
    m.doc() = "bindings for your_project";

    py::class_<your_project::object>(m, "object");
}
```

B. `python/your_project/__init__.py` The entry point for the Python specific code of your project. Also reexports symbols from the generated bindings.

```
from py_your_project import *
```

C. `python/CMakeLists.txt` You can build the bindings using CMake.

```
set(BINDING_NAME "py_your_project")
set(BINDING_SOURCES "src/module.cpp")

set(CMAKE_LIBRARY_OUTPUT_DIRECTORY "${CMAKE_CURRENT_SOURCE_DIR}")

pybind11_add_module(${BINDING_NAME} ${BINDING_SOURCES})

target_link_libraries(${BINDING_NAME} PRIVATE your_project)
```

iii. Add it as a subdirectory In the main `CMakeLists.txt` of your project, add the Python folder:

```
...
add_subdirectory("ext/pybind11")
add_subdirectory("python")
```

Now, the python bindings will be built alongside your project.

3 Documentation

3.1 Python

3.1.1 Sphinx

1. Basic documentation generation

   - `http://www.sphinx-doc.org/en/master/`

```
pip install -U Sphinx
sphinx-apidoc -F -o docs
cd docs
make html
```

- Theme: `https://github.com/rtfd/sphinx_rtd_theme`

2. Publishing on gh-pages Two options:

   - `docs/` folder
   - `gh-pages` branch

   `https://help.github.com/articles/creating-project-pages-using-the-command-line/`

3.2 C/C++

- `http://www.sphinx-doc.org/en/master/`
- `mkdocs`
- `breathe`
- `doxygen`

4 Relevant links

- **Writing documentation**: `http://stevelosh.com/blog/2013/09/teach-dont-tell/`
- **Semantic versioning**: `http://semver.org/`
- **Writing good commit messages**: `http://chris.beams.io/posts/git-commit/`
- **Change log**: `http://keepachangelog.com/`
- **Branching model**: `http://nvie.com/posts/a-successful-git-branching-model/`
- UCL BUG coding standards (sent by Felix)

5 Editors

6 VIM

7 Emacs

8 Python

8.1 CONDA package

8.1.1 Publishing to cicwi

Willem Jan:

> Goed idee. Ik heb een cicwi organization aangemaakt, waarvan voorlopig Allard en ik owners zijn. Het gaat niet met een shared password, maar door anaconda-accounts rechten te geven binnen de cicwi organization door accounts aan de 'Owners' (admin) of 'Packagers' (read/write) group toe te voegen.
>
> Een package uploaden gaat dan met:

```
anaconda upload --user cicwi package.tar.bz2
```

Zie `https://docs.anaconda.com/anaconda-cloud/user-guide/tasks/work-with-organizations/` .

8.2 Documentatie met sphinx

On stackoverflow: What is the docstring format in Python?

8.3 Test my python code

Pytest is a popular python testing framework. It has some dependency injection thingies going on, but most importantly it contains code to compare numbers approximately.

`https://docs.pytest.org/en/latest/`

8.4 Use bumpversion

Changing the version of a python package is a pain. There are python versions in `setup.py`, `__init__.py`, and in `conda/meta.yaml`. This is all very confusing and annoying. Therefore, we have a program called bumpversion that does this for you.

9 C++

9.1 CMAKE

9.2 Python bindings for C++

10 Git

10.1 Good commit messages

10.2 Git branching model

10.3 Release on GitHub

11 General

11.0.1 Write a readme

This github repo contains a useful model of maturity levels for a project's README.md file. It defines both the current level of maturity of a README and gives pointers on how to improve.

11.0.2 Use module load

11.0.3 Use github pages with sphinx

11.0.4 Cookiecutter: project templates

Cookiecutter is a popular way to kickstart a python project. It fills in all the boilerplate.

Cookiecutter templates:

- conda
- rust in python cross platform publish

11.0.5 Travis CI

1. C++17
2. travis.yml / Makefile