HDF5pp Documentation

Tom de Geus

Apr 25, 2019

Contents

Note: This library is free to use under the MIT license. Any additions are very much appreciated, in terms of suggested functionality, code, documentation, testimonials, word of mouth advertisement, ... Bug reports or feature requests can be filed on GitHub. As always, the code comes with no guarantee. None of the developers can be held responsible for possible mistakes.

CHAPTER 1

Contents

1.1 HDF5pp in C++

1.1.1 Teaser

This header-only module provides a very simple interface to store data to a HDF5-file. Using this library writing an HDF5 file becomes as simple as:

```
#include <iostream>
#include <vector>
#include <HDF5pp.h>

int main()
{
  std::vector<double> data = ...;

  H5p::File file = H5p::File("example.hdf5","w");

  file.write("/path/to/data", data);

  return 0;
}
```

Note: Compilation of this example for example with

```
h5c++ -std=c++14 `pkg-config --cflags HDF5pp` example.cpp
```

But probably most easily using CMake. See compile for more details.

Note: In addition to the header-only library this module provides some *command-line tools*.

Note: It might be interesting to look at the header-only library [cpppath](https://github.com/tdegeus/cpppath) to handle the paths of the HDF5 datasets.

1.1.2 General example

The general structure of a program is

```
#include <iostream>
#include <vector>
#include <HDF5pp.h>

int main()
{
  std::vector<double> data = ...;

  H5p::File file = H5p::File("example.hdf5", "w");

  file.write("/path/to/data", data);

  std::vector<double> read = file.read<std::vector<double>>("/path/to/data");

  return 0;
}
```

Note: Although this library is header only, the HDF5 library should be linked. Therefore using either `h5c++` or CMake can be used, see compile.

1.1.3 Function overview

All functions are members of the File class:

```
H5p::File("/path/to/file", "mode");
```

The constructor takes two arguments: the file name and the read/write mode. For the latter there are three possibilities:

- `"w"`: write a new file or overwrite existing file.
- `"r"`: read from existing file.
- `"r+"` or `"a"`: read from and write to an existing file.

In addition it takes one option, the flush settings. The default `true` ensures the file to be flushed after each write operation, allowing external reading while the file is open.

Main functions:

- `void File::write("/path/to/data",...)`

  Write data (scalar, array, matrix, ...). Can be overloaded with many different types, see *Overloaded types*.

- `void File::overwrite("/path/to/data",...)`

  Overwrite data (scalar, array, matrix, ...) of an existing datasets. Can be overloaded with many different types, see *Overloaded types*. Note that the type and shape must match the existing dataset.

- `Type File::read<Type>("/path/to/data")`

  Read data (scalar, array, matrix, ...). Can be templated with many different types, see *Overloaded types*.

- `std::vector<size_t> shape("/path/to/data")`

  Return the shape of the data array.

- `size_t shape("/path/to/data", i)`

  Return the shape of the data array along axis `i`.

- `size_t size("/path/to/data")`

  Return the number of elements in the data array.

Support functions:

- `void File::unlink("/path/to/data")`

  Unlink a path. The dataset is removed when there are no more links to it. Warning: depending on the version of the HDF5 library, the space may not be freed from the file. In that case use `$ h5repack file1 file2` to create a new file without the unused data.

- `bool File::exists("/path/to/data")`

  Check if a path exists.

- `void File::createGroup("/path/to/group")`

  Create a group. Usually there is no need to call this function because the `write` function automatically creates all parent groups.

- `void File::flush()`

  Flush all buffers associated with a file to disk. Usually there is no need to call this function because the `write` function automatically flushes the file (this can be suppressed using the option of the File constructor).

1.1.4 Overloaded types

Note: If your type of choice is not present please submit an issue on GitHub, or file a pull request.

Basic types (size_t, double, ...)

The examples below feature a `double`, which may be replaced with:

- `int`
- `size_t`
- `float`
- `double`
- `std::string`

Writing and or reading is done as follows:

```cpp
#include <iostream>
#include <vector>
#include <HDF5pp.h>

int main()
{
  double data = 10.;

  H5p::File file = H5p::File("example.hdf5", "w");

  file.write("/path/to/data", data);

  double read_data = file.read<double>("/path/to/data");

  return 0;
}
```

[source: example.cpp, compile: CMakeLists.txt]

Basic types, part of an expandable array (size_t, double, ...)

In this case the scalar will be part of an array that automatically expands to contain new entries. The behavior is thus like allocating an array of arbitrary shape and then filling it item-by-item. The actual size is determined by the highest index specified. All entries in the array that have not been explicitly specified are assigned a default fill value. Note:

- One can read one value from, but also read the array as any array (i.e. using `file.read<std::vector<...>>(...)`).
- One can convince oneself about the size of the array using the standard tools (`file.size(...)` and `file.shape(...)`).
- At the first call the array some properties of the array are defined. At this time can choose the fill value (`fill_val`) and the size of the blocks in which the array is stored in the file (`chunk_size`). If one knows the ultimate size one can store in one chunk (most efficient). Otherwise one should choose a value which is high enough not to get a very scattered file, but low enough not to allocate a lot of space that is not used.

The examples below feature a `double`, which may be replaced with:

- `int`
- `size_t`
- `float`
- `double`

Writing and or reading is done as follows:

```cpp
#include <iostream>
#include <vector>
#include <HDF5pp.h>

int main()
{
  H5p::File file = H5p::File("example.hdf5", "w");

  double data = 10.;
  size_t idx  = 0;
```

(continues on next page)

(continued from previous page)

```
  file.write("/path/to/data", data, idx);

  data = 20.;
  idx  = 1;

  // "/path/to/data" is automatically expanded to contain the new entry
  file.write("/path/to/data", data, idx);

  // read one entry
  idx = 0;
  double read_entry = file.read<double>("/path/to/data", idx);

  // read entire array
  std::vector<double> read_data = file.read<std::vector<double>>("/path/to/data");

  return 0;
}
```

[source: example.cpp, compile: CMakeLists.txt]

std::vector

Writing a vector (and optionally its 'dimensions') is done as follows:

```
#include <iostream>
#include <vector>
#include <HDF5pp.h>

int main()
{
  H5p::File file = H5p::File("example.hdf5", "w");

  std::vector<double> data  = { 0., 1., 2., 3., 4., 5. };
  std::vector<size_t> shape = { 3 , 2 };

  file.write("/path/to/data", data, shape);

  std::vector<double> read_data  = file.read<std::vector<double>>("/path/to/data");
  std::vector<size_t> read_shape = file.shape("/path/to/data");

  return 0;
}
```

[source: example.cpp, compile: CMakeLists.txt]

Note: In the HDF5 archive the data is stored as a matrix. However, because `std::vector` is just an array the shape has to be extracted separately. For the richer classes below this is not necessary.

Reading with Python does allow direct interpretation of the matrix

```
import h5py
import numpy as np

f = h5py.File('example.hdf5','r')
```

(continues on next page)

(continued from previous page)

```
print(f['/data'][...])
```

[source: example.py]

cppmat - multidimensional arrays

To enable this feature:

- Include cppmat before HDF5pp:

```
#include <cppmat/cppmat.h>
#include <HDF5pp.h>
```

- Define `HDF5PP_CPPMAT` somewhere before including HDF5pp:

```
#define HDF5PP_CPPMAT
#include <HDF5pp.h>
#include <cppmat/cppmat.h>
```

Writing and reading matrices of arbitrary dimensions can be done as follows:

```
#include <iostream>
#include <cppmat/cppmat.h>
#include <HDF5pp.h>

int main()
{
  cppmat::array<double> data({2,3,4,5});

  // ... fill "data"

  H5p::File file = H5p::File("example.hdf5", "w");

  file.write("/path/to/data", data);

  cppmat::array<double> read_data = file.read<cppmat::array<double>>("/path/to/data");

  return 0;
}
```

[source: example.cpp, compile: CMakeLists.txt]

Note: Several other cppmat-classes can be read directly using for example:

```
cppmat::matrix<double> read_data = file.read<cppmat::array<double>>("/path/to/data");
```

Whether or not your class can be read depends cppmat (if there exists some automatic conversion, which is the case for most classes but not for all).

Eigen - linear algebra library

To enable this feature:

- Include Eigen before HDF5pp:

```
#include <Eigen/Eigen>
#include <HDF5pp.h>
```

- Define `HDF5PP_EIGEN` somewhere before including HDF5pp:

```
#define HDF5PP_EIGEN
#include <HDF5pp.h>
#include <Eigen/Eigen>
```

Writing and reading matrices or arrays can be done as follows:

```
#include <iostream>
#include <Eigen/Eigen>
#include <HDF5pp.h>

// alias row-major Eigen matrix
typedef Eigen::Matrix<double, Eigen::Dynamic, Eigen::Dynamic, Eigen::RowMajor> MatD;

int main()
{
  MatD data(2,2);

  // ... fill "data"

  H5p::File file = H5p::File("example.hdf5", "w");

  file.write("/path/to/data", data);

  MatD read_data = file.read<MatD>("/path/to/data");

  return 0;
}
```

[source: example.cpp, compile: CMakeLists.txt]

1.1.5 Compiling

Introduction

This module is header only. So one just has to `#include <HDF5pp/HDF5pp.h>` (or only one of the submodules) somewhere in the source code, and to tell the compiler where the header-files are. For the latter, several ways are described below.

One should still link with the HDF5 libraries. This is briefly described in *Linking with the HDF5 libraries*.

Note: Before proceeding, some words about optimization. Of course one should use optimization when compiling the release of the code (`-O2` or `-O3`). But it is also a good idea to switch of the assertions in the code (mostly checks on size) that facilitate easy debugging, but do cost time. Therefore, include the flag `-DNDEBUG`. Note that this is all C++ standard. I.e. it should be no surprise, and it is always a good idea to do.

Manual compiler flags

GNU / Clang

Add the following compiler's arguments:

```
-I${PATH_TO_HDF5PP}/src -std=c++14
```

Note: (Not recommended)

If you want to avoid separately including the header files using a compiler flag, `git submodule` is a nice way to go:

1. Include the submodule using `git submodule add https://github.com/tdegeus/HDF5pp.git`.
2. Include using `#include "HDF5pp/src/HDF5pp/HDF5pp.h"`.

   If you decide to manually copy the header file, you might need to modify this relative path to your liking.

Or see *(Semi-)Automatic compiler flags*. You can also combine the `git submodule` with any of the below compiling strategies.

(Semi-)Automatic compiler flags

Install

To enable (semi-)automatic build, one should 'install' HDF5pp somewhere.

Install systemwide (depends on your privileges)

1. Proceed to a (temporary) build directory. For example

   ```
   $ cd /path/to/temp/build
   ```

2. 'Install' HDF5pp:

   ```
   $ cmake /path/to/HDF5pp
   $ make install
   ```

Note: One usually does not need any compiler arguments after following this protocol.

Install in custom location (user)

1. Proceed to a (temporary) build directory. For example

   ```
   $ cd /path/to/temp/build
   ```

2. 'Install' HDF5pp, to install it in a custom location

   ```
   $ mkdir /custom/install/path
   $ cmake /path/to/HDF5pp -DCMAKE_INSTALL_PREFIX:PATH=/custom/install/path
   $ make install
   ```

3. Add the following path to your `~/.bashrc` (or `~/.zshrc`):

```
export PKG_CONFIG_PATH=/custom/install/path/share/pkgconfig:$PKG_CONFIG_PATH
export CPLUS_INCLUDE_PATH=$HOME/custom/install/path/include:$CPLUS_INCLUDE_PATH
```

Note: One usually does not need any compiler arguments after following this protocol.

Note: **(Not recommended)**

If you do not wish to use `CMake` for the installation, or you want to do something custom. You can, of course. Follow these steps:

1. Copy the file `src/HDF5pp.pc.in` to `HDF5pp.pc` to some location that can be found by `pkg_config` (for example by adding `export PKG_CONFIG_PATH=/path/to/HDF5pp.pc:$PKG_CONFIG_PATH` to the `.bashrc`).
2. Modify the line `prefix=@CMAKE_INSTALL_PREFIX@` to `prefix=/path/to/HDF5pp`.
3. Modify the line `Cflags: -I${prefix}/@HDF5_INCLUDE_DIR@` to `Cflags: -I${prefix}/src`.
4. Modify the line `Version: @HDF5PP_VERSION_NUMBER@` to reflect the correct release version.

Compiler arguments from 'pkg-config'

Should the compiler for some reason not be able to find the headers, instead of `-I...` one can now use

```
`pkg-config --cflags HDF5pp` -std=c++14
```

as compiler argument.

Compiler arguments from 'cmake'

Add the following to your `CMakeLists.txt`:

```
set(CMAKE_CXX_STANDARD 14)

find_package(PkgConfig)

pkg_check_modules(HDF5PP REQUIRED HDF5pp)
include_directories(${HDF5PP_INCLUDE_DIRS})
```

Note: Except the C++ standard it should usually not be necessary to load HDF5pp explicitly, as it is installed in a location where the compiler can find it.

Linking with the HDF5 libraries

Using the h5c++ executable

The `h5c++` executable provides a wrapper around your compiler, with all flags set correctly to use HDF5. To compile the following suffices:

```
h5c++ `pkg-config --cflags HDF5pp` -std=c++14 example.cpp
```

Using cmake

The following basic structure of `CMakeLists.txt` can be used:

```
cmake_minimum_required(VERSION 2.8.12)

# define a project name
project(example)

# define empty list of libraries to link
set(PROJECT_LIBS "")

# enforce the C++ standard
set(CMAKE_CXX_STANDARD 14)
set(CMAKE_CXX_STANDARD_REQUIRED ON)

# set optimization level and switch of assertions (set to your liking)
set(CMAKE_BUILD_TYPE Release)
add_definitions(-DNDEBUG)

# load pkg-config
find_package(PkgConfig)

# find HDF5
find_package(HDF5 COMPONENTS CXX REQUIRED)
include_directories(${HDF5_INCLUDE_DIRS})
set(PROJECT_LIBS ${HDF5_LIBS} ${HDF5_LIBRARIES})

# find HDF5pp
pkg_check_modules(HDF5PP REQUIRED HDF5pp)
include_directories(${HDF5PP_INCLUDE_DIRS})

# create executable
add_executable(${PROJECT_NAME} example.cpp)

# link libraries
target_link_libraries(${PROJECT_NAME} ${PROJECT_LIBS})
```

1.2 LowFive in C++

1.2.1 Teaser

```
#include <xtensor/xtensor.hpp> // before LowFive to enable functions
#include <LowFive.h>

```

(continues on next page)

(continued from previous page)

```cpp
int main()
{
  HighFive::File file("example.h5", HighFive::File::Overwrite);

  // write/read to string
  {
    std::string A = "test";

    LowFive::string::dump(file, "/path/to/string", A);

    std::string B = LowFive::string::cast(file, "/path/to/string");
  }

  // write/read to scalar
  {
    int A = 100;

    LowFive::scalar::dump(file, "/path/to/scalar", A);

    int B = LowFive::scalar::cast<int>(file, "/path/to/scalar");
  }

  // write/read to scalar to extendible dataset
  {
    int    A   = 100;
    size_t idx = 10;

    LowFive::scalar::dump(file, "/path/to/extendible", idx, A);

    int B = LowFive::scalar::cast<int>(file, "/path/to/extendible", idx);
  }

  // (over)write/read to xtensor
  {
    xt::xtensor<int,2> A = xt::ones<int>({10,5});

    LowFive::xtensor::dump(file, "/path/to/matrix", A);

    LowFive::xtensor::overwrite(file, "/path/to/matrix", A);

    xt::xtensor<double,2> B = LowFive::xtensor::cast<double,2>(file, "/path/to/matrix
↪");
  }

  // check
  {
    if ( LowFive::exist(file, "/path/to/matrix") )
    {
      size_t size = LowFive::size(file, "/path/to/matrix");

      std::vector<size_t> shape = LowFive::shape(file, "/path/to/matrix");
    }
  }

  return 0;
}
```

1.3 Command-line tools

Note: These tools use Python an depend on h5py as docopt. Get these tools for example using

```
pip3 install h5py doctopt
```

1.3.1 HDF5pp_check

[HDF5pp_check]

```
HDF5pp_check
  Try reading datasets. In case of reading failure the path is printed (otherwise␣
↪nothing is
  printed).

Usage:
  HDF5pp_check <source> [options]

Arguments:
  <source>        HDF5-file.

Options:
  -b, --basic     Only try getting a list of datasets, skip trying to read them.
  -h, --help      Show help.
      --version   Show version.
```

1.3.2 HDF5pp_list

[HDF5pp_list]

```
HDF5pp_list
  List datasets (or groups of datasets) in a HDF5-file.

Usage:
  HDF5pp_list [options] [--fold ARG]... <source>

Arguments:
  <source>    HDF5-file.

Options:
  -f, --fold=ARG        Fold paths.
  -d, --max-depth=ARG   Maximum depth to display.
  -h, --help            Show help.
      --version         Show version.
```

1.3.3 HDF5pp_repair

[HDF5pp_repair]

```
HDF5pp_repair
  Extract readable data from a HDF5-file and copy it to a new HDF5-file.

Usage:
  HDF5pp_repair [options] <source> <destination>

Arguments:
  <source>        Source HDF5-file, possibly containing corrupted data.
  <destination>   Destination HDF5-file.

Options:
  -f, --force     Force continuation, overwrite existing files.
  -h, --help      Show help.
      --version   Show version.
```

1.3.4 HDF5pp_merge

[HDF5pp_merge]

```
HDF5pp_merge
  Merge an entire HDF5-file into another HDF5-file: copy all datasets from <source>␣
↪to some root
  in <destination>. The root is based on the path of <source>, as it is specified:

  * without extension (default)
  * only directory name (--dirname)
  * as specified (--ext)
  * apply some regex substitution (--find ... --replace ...)

Usage:
  HDF5pp_merge [options] <source> <destination>

Arguments:
  <source>            Source HDF5-file.
  <destination>       Destination HDF5-file (appended).

Options:
      --ext           Include extension of <source> in root.
      --dirname       Use only directory name of <source> in root.
  -f, --find=ARG      Regex search  to apply to <source>.
  -r, --replace=ARG   Regex replace to apply to <source>.
  -p, --root=ARG      Manually set root.
       --rm           Remove <source> after successful copy.
  -d, --dry-run       Dry run.
      --verbose       Verbose operations.
  -h, --help          Show help.
      --version       Show version.
```

Tip: To merge a bunch of files one could use:

```
find . -iname '*.hdf5' -exec HDF5pp_merge {} output.hdf5 \;
```

In this case `HDF5pp_merge` is called for each HDF5-file that is found. Note that if `output.hdf5` already existed, it is skipped by `HDF5pp_merge`.

1.3.5 HDF5pp_select

[HDF5pp_select]

```
HDF5pp_select
  Select datasets (or groups of datasets) from a HDF5-file and store to a new HDF5-
↪file.

JSON:
  The input can be a JSON file that looks like:

    {
      "/new/path" : "/old/path",
      ...
    }

Usage:
  HDF5pp_select [options] [--path ARG]... <source> <destination>

Arguments:
  <source>          Source HDF5-file.
  <destination>     Destination HDF5-file (appended).

Options:
  -p, --path=ARG    Pair of paths: "/destination/path;/source/path".
  -j, --json=ARG    JSON file with contains the path change.
      --sep=ARG     Set path separator. [default: ;]
  -f, --force       Force continuation, continue also if this operation discards␣
↪fields.
      --verbose     Verbose operations.
  -h, --help        Show help.
      --version     Show version.
```

1.3.6 HDF5pp_find

[HDF5pp_find]

```
HDF5pp_find
  Function, loosely based on Linux' find, that searches datasets in a HDF5 file. It␣
↪allows to
  execute both file operations and basic dataset manipulations.

Usage:
  HDF5pp_find <source> [options]

Arguments:
  <source>    HDF5-file.

Options:
      --iname=ARG       Search for dataset, ignoring the case.
      --find=ARG        Find. [default: (.*)]
      --remove          Remove command.
      --not             Execute command only if there are no matches.
      --dry-run         Perform a dry-run.
      --verbose         Print file-path.
  -h, --help            Show help.
      --version         Show version.
```

Tip: To remove all files that do not contain a dataset called "completed":

```
find . -iname '*.hdf5' -exec HDF5pp_find {} --not --iname "completed" --remove \;
```

Tip: To rename the directory that contains a HDF5-file, if that file contains a dataset called "completed":

```
mv_regex --dirname "(id\=[0-9]{3})([A-Z])" "\1C" `HDF5pp_find --iname "completed"␣
↪id=000Q/data.hdf5`
```

(takes directories *id=000Q*, *id=001Q*, ... and renames them to *id=000C*, *id=001C*, ...).

1.4 Notes for developers

1.4.1 Make changes / additions

Be sure to run and verify all examples! All existing examples should pass, while new examples should be added to document and check any new functionality.

1.4.2 Create a new release

1. Update the version number specified in `HDF5pp.h` using `HDF5PP_WORLD_VERSION`, `HDF5PP_MAJOR_VERSION`, `HDF5PP_MINOR_VERSION`.
2. Upload the changes to GitHub and create a new release there (with the correct version number).