

---

**xtensor-io**

**Apr 15, 2019**



---

# INSTALLATION

---

<b>1</b>	<b>Enabling xtensor-io in your C++ libraries</b>	<b>3</b>
<b>2</b>	<b>Licensing</b>	<b>5</b>
2.1	Installation . . . . .	5
2.2	Basic Usage . . . . .	6
2.3	HDF5 interface . . . . .	8
2.4	API Reference . . . . .	10
2.5	Releasing xtensor-io . . . . .	15



Input/Output routines for reading images, audio, and NumPy `npz` files for the `xtensor` C++ multi-dimensional array library.

`xtensor-io` offers bindings to popular open source libraries for reading and writing

- Images with `OpenImageIO` (many supported formats, among others: `jpg`, `png`, `gif`, `tiff`, `bmp` ...)
- Sound files with `libsndfile` (supports `wav`, `ogg` files)
- NumPy compressed file format (`npz`). Note: support for uncompressed NumPy files (`npz`) is available in core `xtensor`.



---

## Enabling `xtensor-io` in your C++ libraries

---

`xtensor` and `xtensor-io` require a modern C++ compiler supporting C++14. The following C++ compilers are supported:

- On Windows platforms, Visual C++ 2015 Update 2, or more recent
- On Unix platforms, gcc 4.9 or a recent version of Clang





We use a shared copyright model that enables all contributors to maintain the copyright on their contributions. This software is licensed under the BSD-3-Clause license. See the LICENSE file for details.

## 2.1 Installation

xtensor-io is a header-only library but depends on some traditional libraries that need to be installed. On Linux, installation of the dependencies can be done through the package manager, anaconda or manual compilation.

### 2.1.1 Using the conda package

A package for xtensor-io is available on the conda package manager. The package will also pull all the dependencies (OpenImageIO, libsndfile and zlib).

```
conda install xtensor-io -c conda-forge
```

The easiest way to make use of xtensor-io in your code is by using cmake for your project. In order for cmake to pick up the xtensor-io dependency, just utilize the interface target and link the xtensor-io library to your target.

```
add_executable(my_exec my_exec.cpp)
target_link_libraries(my_exec
    PUBLIC
        xtensor-io
)
```

This should be enough to add the correct directories to the include\_directories and link the required libraries. However, depending on your system setup there might be problems upon executing, as the dynamic library is not picked up correctly. So if you run into errors that read something like “Library could not be opened

... “, then set the RPATH, the runtime library search path, to the conda library path of your environment. We utilize the CMAKE\_INSTALL\_PREFIX for this purpose, so if you call cmake like this `cmake .. -DCMAKE_INSTALL_PREFIX=$CONDA_PREFIX` and add the following to your `CMakeLists.txt`.

```
set_target_properties(my_exec
  PROPERTIES
    INSTALL_RPATH "${CMAKE_INSTALL_PREFIX}/lib;${CMAKE_INSTALL_PREFIX}/${CMAKE_
↪INSTALL_LIBDIR}"
    BUILD_WITH_INSTALL_RPATH ON
)
```

## 2.1.2 From source with cmake

You can also install `xtensor-io` from source with `cmake`. On Unix platforms, from the source directory: However, you need to make sure to have the required libraries available on your machine. Note: you don't need all libraries if you only use parts of `xtensor-io`. `libsndfile` is required for `xaudio`, `OpenImageIO` for `ximage` and `zlib` for `xnpz`.

Installation of the dependencies on Linux:

```
# Ubuntu / Debian
sudo apt-get install libsndfile-dev libopenimageio-dev zlib1g-dev
# Fedora
sudo dnf install libsndfile-devel OpenImageIO-devel zlib-devel
```

```
mkdir build
cd build
cmake -DCMAKE_INSTALL_PREFIX=/path/to/prefix ..
make install
```

On Windows platforms, from the source directory:

```
mkdir build
cd build
cmake -G "NMake Makefiles" -DCMAKE_INSTALL_PREFIX=/path/to/prefix ..
nmake
nmake install
```

## 2.2 Basic Usage

### 2.2.1 Example : Reading images, audio and NPZ files

```
#include <iostream>
#include <xtensor/xbuilder.hpp>
#include <xtensor/xmath.hpp>
#include <xtensor-io/xnpz.hpp>
#include <xtensor-io/ximage.hpp>
#include <xtensor-io/xaudio.hpp>

int main()
{
```

(continues on next page)

(continued from previous page)

```

// loads png image into xarray with shape HEIGHT x WIDTH x CHANNELS
auto arr = xt::load_image("test.png");

// write xarray out to JPEG image
xt::dump_image("dumptest.jpg", arr + 5);

// load npz file containing multiple arrays
auto npz_map = xt::load_npz("test.npz");

auto arr_0 = npz_map["arr_0"].cast<double>();
auto arr_1 = npz_map["arr_1"].cast<unsigned long>();

// open a wav file
auto audio = xt::load_audio("files/xtensor.wav");
std::cout << "Sampling Frequency: " << std::get<0>(audio) << std::endl;
auto& arr = std::get<1>(audio); // audio contents (like scipy.io.wavfile results)

// save a sine wave sound
int freq = 2000;
int sampling_freq = 44100;
double duration = 1.0;

auto t = xt::arange(0.0, duration, 1.0 / sampling_freq);
auto y = xt::sin(2.0 * numeric_constants<double>::PI * freq * t);

xt::dump_audio("files/sine.wav", y, sampling_freq);

return 0;
}

```

## 2.2.2 Example : Reading and writing HDF5 files

```

#include <xtensor/xio.hpp>
#include <xtensor-io/xhighfive.hpp>

int main()
{
    xt::xtensor<double, 2> A = xt::ones<double>({10, 5});

    xt::dump_hdf5("example.h5", "/path/to/data", A);

    A = xt::load_hdf5<xt::xtensor<double, 2>>("example.h5", "/path/to/data");

    std::cout << A << std::endl;

    return 0;
}

```

## 2.3 HDF5 interface

### 2.3.1 Basic interface

The basic interface consists of two functions:

- `xt::dump_hdf5(filename, path, data[, xt::file_mode, xt::dump_mode])`  
Dump data (e.g. a matrix) to a DataSet in a HDF5 file. The data can be: scalar (incl. `std::string`), `std::vector<scalar>` (incl. `std::string`), `xt::xarray<scalar>`, and `xt::xtensor<scalar, dim>`.
- `data = xt::load_hdf5<...>(filename, path)`  
Read data (e.g. a matrix) from a DataSet in a HDF5 file. The same overloads as for dump are available.

For example:

```
#include <xtensor/xio.hpp>
#include <xtensor-io/xhighfive.hpp>

int main()
{
    xt::xtensor<double, 2> A = xt::ones<double>({10, 5});

    xt::dump_hdf5("example.h5", "/path/to/data", A);

    A = xt::load_hdf5<xt::xtensor<double, 2>>("example.h5", "/path/to/data");

    std::cout << A << std::endl;

    return 0;
}
```

### 2.3.2 Advanced interface

The advanced interface provides simple free-functions that write to or read from an open `HighFive::File`. It consists of the following functions:

- `xt::dump(file, path, data[, xt::dump_mode])`  
Dump data (e.g. a matrix) to a DataSet in a HDF5 file. The data can be: scalar (incl. `std::string`), `std::vector<scalar>` (incl. `std::string`), `xt::xarray<scalar>`, and `xt::xtensor<scalar, dim>`.
- `xt::dump(file, path, data, {i, ...})`  
Dump scalar to the index `{i, ...}` of a DataSet. If the DataSet does not yet exist, an extendible DataSet of the appropriate rank and shape is created. If it does exist, the DataSet is resized if necessary.
- `data = xt::load<...>(file, path)`  
Read data (e.g. a matrix) from a DataSet in a HDF5 file. The same overloads as for dump are available.
- `data = xt::load<...>(file, path, {i, ...})`  
Read scalar as index `{i, ...}` of a DataSet.

For example:

```

#include <xtensor/xio.hpp>
#include <xtensor-io/xhighfive.hpp>

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

    xt::xtensor<double,2> A = xt::ones<double>({10,5});

    xt::dump(file, "/path/to/data", A);

    xt::dump(file, "/path/to/data", A, xt::dump_mode::overwrite);

    A = xt::load<xt::xtensor<double,2>>(file, "/path/to/data");

    std::cout << A << std::endl;

    return 0;
}

```

### 2.3.3 Compiling & dependencies

This library uses header only `HighFive` library and the `HDF5` library. Both should be available upon compiling and linking respectively.

#### Manual

Compiling can then proceed through

```

g++ -std=c++14 -lhdf5 main.cpp

# manually set paths
g++ -I... -L... -std=c++14 -lhdf5 main.cpp

```

Alternatively, `HDF5` provides a wrapper command that sets the paths to the `HDF5` library (not to `HighFive`):

```

h5c++ -std=c++14 main.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)

```

(continues on next page)

```

# set optimization level
set(CMAKE_BUILD_TYPE Release)

# ...

# find HighFive
find_package(HighFive REQUIRED)

# find HDF5
find_package(HDF5 REQUIRED)
include_directories(${HDF5_INCLUDE_DIRS})
set(PROJECT_LIBS ${HDF5_C_LIBRARIES})

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

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

```

## 2.4 API Reference

### 2.4.1 NPZ files

Defined in `xtensor-io/xnpz.hpp`

auto `xt::load_npz` (std::string *filename*)

Loads a npz file.

This function returns a map. The individual arrays are not casted to a specific file format. This has to be done before they can be used as xarrays (e.g. `auto arr_0 = npz_map["arr_0"].cast<double>();`)

**Return** returns a map with the stored arrays. Array names are the keys.

#### Parameters

- `filename`: The filename of the npz file

template<class **T**>

xarray<**T**> `xt::load_npz` (std::string *filename*, std::string *search\_varname*)

Loads a specific array indicated by `search_varname` from npz file.

All other data in the npz file is ignored.

**Return** xarray with the contents of the loaded array

#### Parameters

- `filename`: The npz filename
- `search_varname`: The array name to be loaded

template<class **E**>

void `xt::dump_npz` (std::string *filename*, std::string *varname*, **const** xt::xexpression<**E**> &*e*, bool *compression* = false, bool *append\_to\_existing\_file* = true)

Save a xarray or xtensor to a NPZ file.

If a npz file with `filename` exists already, the new data is appended to the existing file by default. Note: currently no checking of name-collision is performed!

#### Parameters

- `filename`: filename to save to
- `varname`: desired name of the variable
- `e`: xexpression to save
- `compression`: true enables compression, otherwise store uncompressed (default false)
- `append_to_existing_file`: If true, appends new data to existing file

## 2.4.2 Images

Defined in `xtensor-io/ximage.hpp`

```
template<class T = unsigned char>
xarray<T> xt::load_image (std::string filename)
```

Load an image from file at `filename`.

Storage format is deduced from file ending.

**Return** xarray with image contents. The shape of the xarray is HEIGHT x WIDTH x CHANNELS of the loaded image, where CHANNELS are ordered in standard R G B (A).

#### Parameters

- `filename`: The path of the file to load

```
template<class E>
void xt::dump_image (std::string filename, const xexpression<E> &data, dump_image_options const
&options = dump_image_options())
```

Save image to disk.

The desired image format is deduced from `filename`. Supported formats are those supported by OpenImageIO. Most common formats are supported (jpg, png, gif, bmp, tiff). The shape of the array must be HEIGHT x WIDTH or HEIGHT x WIDTH x CHANNELS.

#### Parameters

- `filename`: The path to the desired file
- `data`: Image data
- `options`: Pass a `dump_image_options` object to fine-tune image export

## 2.4.3 Audio files

Defined in `xtensor-io/xaudio.hpp`

```
template<class T = short>
auto xt::load_audio (std::string filename)
```

Read a WAV or OGG file This function reads a WAV file at `filename`.

**Return** tuple with (samplerate, xarray holding the data). The shape of the xarray is FRAMES x CHANNELS.

#### Parameters

- filename: File to load.

### Template Parameters

- T: select type (default: short, 16bit)

template<class E>

void xt::dump\_audio (std::string filename, const xexpression<E> &data, int samplerate, int format = SF\_FORMAT\_WAV | SF\_FORMAT\_PCM\_16)

Save an xarray in WAV or OGG sound file format Please consult the libsndfile documentation for more format flags.

### Parameters

- filename: save under filename
- data: xarray/xexpression data to save
- samplerate: The samplerate of the data
- format: select format (see sndfile documentation). Default is 16 bit PCM WAV format.

## 2.4.4 HDF5 files

Defined in xtensor-io/xhighfive.hpp

template<class T>

void xt::dump\_hdf5 (const std::string &fname, const std::string &path, const T &data, xt::file\_mode fmode = xt::file\_mode::create, xt::dump\_mode dmode = xt::dump\_mode::create)

Write field to a new DataSet in an HDF5 file.

**Return** dataset the newly created HighFive::DataSet (e.g. to add an attribute)

### Parameters

- file: opened HighFive::File (has to be writeable)
- path: path of the DataSet
- data: the data to write
- dmode: DataSet-write mode (xt::dump\_mode::create | xt::dump\_mode::overwrite)

template<class T>

auto xt::load\_hdf5 (const std::string &fname, const std::string &path)

Load a DataSet in an open HDF5 file to an object (templated).

**Return** data the read data

### Parameters

- file: opened HighFive::File (has to be writeable)
- path: path of the DataSet

template<class T>

HighFive::DataSet xt::dump (HighFive::File &file, const std::string &path, const xt::xarray<T> &data, xt::dump\_mode dmode = xt::dump\_mode::create)

Write “xt::xarray<T>” to a new DataSet in an open HDF5 file.

**Return** dataset the newly created HighFive::DataSet (e.g. to add an attribute)



**Parameters**

- `file`: opened `HighFive::File` (has to be writeable)
- `path`: path of the `DataSet`
- `data`: the data to write
- `dmode`: `DataSet`-write mode (`xt::dump_mode::create` | `xt::dump_mode::overwrite`)

```
template<class T, std::size_t rank>
HighFive::DataSet xt::dump (HighFive::File &file, const std::string &path, const xt::xtensor<T, rank>
                           &data, xt::dump_mode dmode = xt::dump_mode::create)
Write “xt::xtensor<T,rank>” to a new DataSet in an open HDF5 file.
```

**Return** dataset the newly created `HighFive::DataSet` (e.g. to add an attribute)

**Parameters**

- `file`: opened `HighFive::File` (has to be writeable)
- `path`: path of the `DataSet`
- `data`: the data to write
- `dmode`: `DataSet`-write mode (`xt::dump_mode::create` | `xt::dump_mode::overwrite`)

```
template<class T>
HighFive::DataSet xt::dump (HighFive::File &file, const std::string &path, const std::vector<T> &data,
                           xt::dump_mode dmode = xt::dump_mode::create)
Write “std::vector<T>” to a new DataSet in an open HDF5 file.
```

**Return** dataset the newly created `HighFive::DataSet` (e.g. to add an attribute)

**Parameters**

- `file`: opened `HighFive::File` (has to be writeable)
- `path`: path of the `DataSet`
- `data`: the data to write
- `dmode`: `DataSet`-write mode (`xt::dump_mode::create` | `xt::dump_mode::overwrite`)

```
template<class T>
HighFive::DataSet xt::dump (HighFive::File &file, const std::string &path, const T &data,
                           xt::dump_mode dmode = xt::dump_mode::create)
Write scalar/string to a new DataSet in an open HDF5 file.
```

**Return** dataset the newly created `HighFive::DataSet` (e.g. to add an attribute)

**Parameters**

- `file`: opened `HighFive::File` (has to be writeable)
- `path`: path of the `DataSet`
- `data`: the data to write
- `dmode`: `DataSet`-write mode (`xt::dump_mode::create` | `xt::dump_mode::overwrite`)

```
template<class T>
```

HighFive::DataSet xt::dump (HighFive::File &file, const std::string &path, const T &data, const std::vector<std::size\_t> &idx)

Write a scalar to a (new, extendible) DataSet in an open HDF5 file.

**Return** dataset the (newly created) HighFive::DataSet (e.g. to add an attribute)

#### Parameters

- file: opened HighFive::File (has to be writeable)
- path: path of the DataSet
- data: the data to write
- idx: the indices to which to write

template<class T>

auto xt::load (const HighFive::File &file, const std::string &path, const std::vector<std::size\_t> &idx)

Load entry “{i,...}” from a DataSet in an open HDF5 file to a scalar.

**Return** data the read data

#### Parameters

- file: opened HighFive::File (has to be writeable)
- idx: the indices to load
- path: path of the DataSet

template<class T>

auto xt::load (const HighFive::File &file, const std::string &path)

Load a DataSet in an open HDF5 file to an object (templated).

**Return** data the read data

#### Parameters

- file: opened HighFive::File (has to be writeable)
- path: path of the DataSet

bool xt::extensions::exist (const HighFive::File &file, const std::string &path)

Check if a path exists (is a Group or DataSet) in an open HDF5 file.

#### Parameters

- file: opened HighFive::File
- path: path of the Group/DataSet

void xt::extensions::create\_group (HighFive::File &file, const std::string &path)

Recursively create groups in an open HDF5 file such that a DataSet can be created.

For example if the path = “/path/to/dataset”, this function will create the groups “/path” and “/path/to”.

#### Parameters

- file: opened HighFive::File
- path: path of the DataSet

`std::size_t xt::extensions::size (const HighFive::File &file, const std::string &path)`  
 Get the size of an existing DataSet in an open HDF5 file.

**Return** size the size of the HighFive::DataSet

**Parameters**

- `file`: opened HighFive::File
- `path`: path of the DataSet

`std::vector<std::size_t> xt::extensions::shape (const HighFive::File &file, const std::string &path)`  
 Get the shape of an existing DataSet in an open HDF5 file.

**Return** shape the shape of the HighFive::DataSet

**Parameters**

- `file`: opened HighFive::File
- `path`: path of the DataSet

## 2.5 Releasing xtensor-io

### 2.5.1 Releasing a new version

From the master branch of xtensor-io

- Make sure that you are in sync with the master branch of the upstream remote.
- In file `xtensor_io_config.hpp`, set the macros for `XTENSOR_IO_VERSION_MAJOR`, `XTENSOR_IO_VERSION_MINOR` and `XTENSOR_IO_VERSION_PATCH` to the desired values.
- Update the readme file w.r.t. dependencies on xtensor.
- Stage the changes (`git add`), commit the changes (`git commit`) and add a tag of the form `Major.minor.patch`. It is important to not add any other content to the tag name.
- Push the new commit and tag to the main repository. (`git push`, and `git push --tags`)

### 2.5.2 Updating the conda-forge recipe

xtensor-io has been packaged for the conda package manager. Once the new tag has been pushed on GitHub, edit the conda-forge recipe for xtensor in the following fashion:

- Update the version number to the new Major.minor.patch.
- Set the build number to 0.
- Update the hash of the source tarball.
- Check for the versions of the dependencies.
- Optionally, rerender the conda-forge feedstock.

### 2.5.3 Updating the stable branch

Once the conda-forge package has been updated, update the `stable` branch to the newly added tag.



## X

`xt::dump (C++ function)`, 12, 13  
`xt::dump_audio (C++ function)`, 12  
`xt::dump_hdf5 (C++ function)`, 12  
`xt::dump_image (C++ function)`, 11  
`xt::dump_npz (C++ function)`, 10  
`xt::extensions::create_group (C++ function)`, 14  
`xt::extensions::exist (C++ function)`, 14  
`xt::extensions::shape (C++ function)`, 15  
`xt::extensions::size (C++ function)`, 14  
`xt::load (C++ function)`, 14  
`xt::load_audio (C++ function)`, 11  
`xt::load_hdf5 (C++ function)`, 12  
`xt::load_image (C++ function)`, 11  
`xt::load_npz (C++ function)`, 10