NP2LAS
DYNAMIC LIBRARY TO USE LASTOOLS READ AND WRITE FUNCTIONS IN PYTHON

CONTENTS

1. INTRODUCTION

1.1 INITIAL DESCRIPTION

NP2LAS is a source code for using LAStools library functionalities to read and write las/.laz files to/from numpy arrays. The source code was developed in C language and generates as output a dynamic library. The dynamic library is load at a reader and writer python functions using the python library *ctypes* (ReadLaz.py and WriteLaz.py). The source code in C language can be compile at Windows or Linux platform to generate the dynamic library (.dll or .so) file. LASzip and Laslib libraries are not provided (https://github.com/LAStools/LAStools), which must be obtained directly from the owners company with the appropriate copyright. The main idea of the reading tool is to define in the python functions pointers to empty numpy arrays, which will be full field after las/laz file reading using Laslib. In the other hand, the write tool will define pointer to numpy arrays containing the fields information, such as X, Y and Z coordinates, Intensity, Return Number and Number of returns, which will be write in las/laz format using Laslib and exported. Therefore, we would like to highlight that this tool was developed in the context of developer's needs. The codes provided can be used as examples for other code developments. Some code adaptations can be required according to the user application. The default fields (pointers) for reading and writing are: X, Y, Z, Intensity, Return Number, Number of returns and extra parameters (e.g. Reflectance, Range and Deviation). The output fields can be change according to the users need before compilation. For instance, classification, edge of flight line, RGB, user data, scan angle rank and scan direction flag fields can be add. As default Laz or Las files will be saved after writing as Version 1.4 and data point format 1.0, which can also be modified in header specifications in the C source code.

Attention, please make sure that python and GCC compilers have the same architecture (32 bits or 64 bits), otherwise problem loading the dynamic library with *ctypes* will occur.

Authors

The authors welcome all comments for improving the toolbox and its documentation.
Mariana Batista Campos has contributed to code writing, testing and documentation.
Eetu Puttonen has contributed to code debug, development and documentation

Acknowledgements

Martin Isenburg, for creating, developing, and maintaining the LAStools and LASlib, rapidlasso.com

LAStools mailing list
http://groups.google.com/group/lastools

Funding

The work of authors has been partly supported from the following projects: Academy of Finland project no.316096/320075, "Upscaling of carbon intake and water balance models of individual trees to wider areas with short interval laser scanning time series" and from the Strategic Research Council at the Academy of Finland project no. 293389/314312, "Competence Based Growth Through Integrated Disruptive Technologies of 3D Digitalization, Robotics, Geospatial Information and Image Processing/Computing - Point Cloud Ecosystem (COMBAT)

2. INSTRUCTIONS FOR COMPILING

2.1 NP2LAS IN WINDOWS

This is a short guide for compiling Nplas and LAStools libraries with Code Blocks (MinGW GCC 8.1). Please, check the compatibility between your python version and the GCC compiler used in Code Blocks. Make sure that python and GCC compilers have the same architecture (32 bits or 64 bits), otherwise problem loading the dynamic library with *ctypes* will occur. These instructions have been tested with Python 3.7 (64 bits), MinGW GCC 8.1 (64 bits) and LAStools v200131.

PROJECT CONFIGURATION

First, install Code Blocks (v.16.01 was tested) [http://www.codeblocks.org/]

Download LASTools source code: http://lastools.github.io/download/LAStools.zip
Attention: Please check LASTools copyright and license.

At Code Blocks → Open dynamic library project
Change compiler at:
Settings → Compiler → Global compiler settings →Toolchain Executable → Find MinGW GCC 8.1.
Common path: C:\Program Files\CodeBlocks\MinGW

PROJECT → Build options → Compiler settings → Compiler Flags [-std=c++11]

PROJECT → Build options → Compiler settings → OtherCompilerOptions:

-fexceptions
-fpermissive

PROJECT → Build options → Linker settings → Linker options →include standard C/C++ libs:

kernel32
user32
gdi32
winspool
comdlg32
advapi32
shell32
ole32
oleaut32
uuid
odbc32

PROJECT →Build options → Search directory → compiler → point to

Lastools/Laslib/src
Lastools/Laslib/inc
Lastools/Laszip/src
Lastools/Laszip/dll

PROJECT →Build options → Search directory → linker → point to

Lastools/Laslib/src
Lastools/Laslib/inc
Lastools/Laszip/src
Lastools/Laszip/dll

INPUT FILES

On the workspace → Right click on the project name → add files

HEADERS:

LASZIP – ALL HEADERS (.h) from LASzip/src and LASzip/dll
LASLIB – ALL HEADERS (.h) at LASlib/inc
MAIN HEADER - main.h

SOURCES:

LASZIP – ALL SOURCES(.cpp) from LASzip/src and LASzip/dll
LASLIB – ALL SOURCES(.cpp) at LASlib/src
MAIN SOUCE CODE - main.cpp

At this point modifications at main.h and main.cpp can be required according to user application and the Las fields required to read and write. If more pointers are add in the C source code, ReadLaz.py and WriteLaz.py scripts also need to be modified.

BUILD THE PROJECT

OUTPUT: dynamic library file (.dll) at bin/Debug folder or bin/Release

LOAD DLL AT PYTHON

At Python scripts:

First, if more pointers were add in the C source code, please update ReadLaz.py and WriteLaz.py scripts include the same number of pointers to numpy arrays.

Example of use at Main.py:
Include the path to the generated dll → DLLPATH=" /bin/Release/Las2Array_StaticLibrary.dll"
Include the path to the input las file → inputLas="Data/SampleData.laz"
Include the path to the output las file → outputLas=" SampleData /teste.laz"

2.2 NP2LAS IN LINUX (UBUNTU 18.04)

ATTENTION: !!! GCC VERSION AND RVLIB VERSION MUST BE COMPATIBLE!!!

This project was build using:

✓ GCC VERSION 8.3

✓ LASzip DLL v3.4 r1 (build 190728) – [https://rapidlasso.com/laszip/]

✓ Make installation ubuntu:

[sudo apt-get update]
[sudo apt-get install cmake]
[sudo apt-get upgrade]

PROJECT CONFIGURATION

Open the project folder. Example: PythonCodes/ubuntu_np2las
mkdir PythonCodes → cd PythonCodes → mkdir ubuntu_ np2las → cd ubuntu_ np2las

Setup and Compile Laslib and LASZip:

git clone https://github.com/LASlib/LASlib.git
git clone https://github.com/LASzip/LASzip.git
go to LASlib /LASzip folder
cd LASlib /LASzip
mkdir build
cd build
cmake ..
sudo make install

You can also build Laslib/ LASZip source codes by command line.
Example of **ONE** (laswritepoint.cpp.o) Laslib function compilation by command line:

g++ -fPIC -g -O2 -Wall -std=c++11 -DBUILD_DLL -g -fpermissive -DBUILD_DLL -ILAStools/LASlib/inc -ILAStools/LASlib/src -ILAStools/LASzip/src -ILAStools/LASlib/inc -ILAStools/LASlib/src -ILAStools/LASzip/src -ILAStools/LASzip/dll -c /usr/local/PythonCodes/LAStools/LASzip/src/**laswritepoint.cpp -o** obj/Release/LAStools/LASzip/src/**laswritepoint.cpp.o**

Compile Main.cpp and use the command line below to build the .so library, including/changing the path to the object files (.o). Example in Release mode:

g++ -g -O2 -Wall -std=c++11 -g -fpermissive -DBUILD_DLL -fPIC -shared -rdynamic -g -LLAStools/LASzip/dll -LLAStools/LASlib/inc -LLAStools/LASlib/src -LLAStools/LASzip/src obj/Release/LAStools/LASlib/src/fopen_compressed.cpp.o obj/Release/LAStools/LASlib/src/lasfilter.cpp.o obj/Release/LAStools/LASlib/src/lasignore.cpp.o obj/Release/LAStools/LASlib/src/lasreader.cpp.o obj/Release/LAStools/LASlib/src/lasreader_asc.cpp.o obj/Release/LAStools/LASlib/src/lasreader_bil.cpp.o obj/Release/LAStools/LASlib/src/lasreader_bin.cpp.o obj/Release/LAStools/LASlib/src/lasreader_dtm.cpp.o obj/Release/LAStools/LASlib/src/lasreader_las.cpp.o obj/Release/LAStools/LASlib/src/lasreader_ply.cpp.o obj/Release/LAStools/LASlib/src/lasreader_qfit.cpp.o obj/Release/LAStools/LASlib/src/lasreader_shp.cpp.o obj/Release/LAStools/LASlib/src/lasreader_txt.cpp.o obj/Release/LAStools/LASlib/src/lasreaderbuffered.cpp.o obj/Release/LAStools/LASlib/src/lasreadermerged.cpp.o obj/Release/LAStools/LASlib/src/lasreaderpipeon.cpp.o obj/Release/LAStools/LASlib/src/lasreaderstored.cpp.o obj/Release/LAStools/LASlib/src/lastransform.cpp.o obj/Release/LAStools/LASlib/src/lasutility.cpp.o obj/Release/LAStools/LASlib/src/laswaveform13reader.cpp.o obj/Release/LAStools/LASlib/src/laswaveform13writer.cpp.o obj/Release/LAStools/LASlib/src/laswriter.cpp.o obj/Release/LAStools/LASlib/src/laswriter_bin.cpp.o obj/Release/LAStools/LASlib/src/laswriter_las.cpp.o obj/Release/LAStools/LASlib/src/laswriter_qfit.cpp.o obj/Release/LAStools/LASlib/src/laswriter_txt.cpp.o obj/Release/LAStools/LASlib/src/laswriter_wrl.cpp.o obj/Release/LAStools/LASlib/src/laswritercompatible.cpp.o obj/Release/LAStools/LASzip/dll/laszip_api.c.o obj/Release/LAStools/LASzip/src/arithmeticdecoder.cpp.o obj/Release/LAStools/LASzip/src/arithmeticencoder.cpp.o obj/Release/LAStools/LASzip/src/arithmeticmodel.cpp.o obj/Release/LAStools/LASzip/src/integercompressor.cpp.o

obj/Release/LAStools/LASzip/src/lasindex.cpp.o obj/Release/LAStools/LASzip/src/lasinterval.cpp.o
obj/Release/LAStools/LASzip/src/lasquadtree.cpp.o obj/Release/LAStools/LASzip/src/lasreaditemcompressed_v1.cpp.o
obj/Release/LAStools/LASzip/src/lasreaditemcompressed_v2.cpp.o
obj/Release/LAStools/LASzip/src/lasreaditemcompressed_v3.cpp.o
obj/Release/LAStools/LASzip/src/lasreaditemcompressed_v4.cpp.o obj/Release/LAStools/LASzip/src/lasreadpoint.cpp.o
obj/Release/LAStools/LASzip/src/laswriteitemcompressed_v1.cpp.o
obj/Release/LAStools/LASzip/src/laswriteitemcompressed_v2.cpp.o
obj/Release/LAStools/LASzip/src/laswriteitemcompressed_v3.cpp.o
obj/Release/LAStools/LASzip/src/laswriteitemcompressed_v4.cpp.o obj/Release/LAStools/LASzip/src/laswritepoint.cpp.o
obj/Release/LAStools/LASzip/src/laszip.cpp.o obj/Release/main.cpp.o -o bin/Release/Las2Array_StaticLibrary.so

At this point modifications at main.h and main.cpp can be required according to user application and the Las fields required to read and write. If more pointers are add in the C source code, ReadLaz.py and WriteLaz.py scripts also need to be modified.

BUILD THE PROJECT

OUTPUT: library file (.so) at bin/Release

LOAD DLL AT PYTHON

At Python scripts:

First, if more pointers were add in the C source code, please update ReadLaz.py and WriteLaz.py scripts include the same number of pointers to numpy arrays.

Example of use at Main.py:
Include the path to the generated dll → DLLPATH=" /bin/Release/Las2Array_StaticLibrary.dll"
Include the path to the input las file → inputLas="Data/SampleData.laz"
Include the path to the output las file → outputLas=" SampleData /teste.laz"

Run the code: [sudo python3 Main.py]