Installation Guide

OpenShot™ Video Library | libopenshot (C++)

Introduction

The **OpenShot Video Library** (also known as *libopenshot*) was created to provide a free and

open-source video editing, composition, animation, and playback library, which focuses on professional features, stability, cross-platform support, and multi-core processor support. The library is written in C++ and includes full bindings for Python and Ruby. These instructions are designed for programmers, packagers, and adventurers who are interested in compiling and installing the source code.

Features

- Cross-Platform (Linux, Windows, and Mac)
- Animation Curves (Bézier, Linear, Constant)
- Multi-Layer Compositing
- Time Mapping (Curve-based Slow Down, Speed Up, Reverse)
- Audio Mixing (Curve-based)
- Audio Plug-ins (VST & AU)
- Audio Drivers (ASIO, WASAPI, DirectSound, CoreAudio, iPhone Audio, ALSA, JACK, and Android)
- Telecine and Inverse Telecine (Film to TV, TV to Film)
- Frame Rate Conversions
- Multi-Processor Support (Performance)
- Python and Ruby Bindings (All Features Supported)
- Unit Tests (Stability)
- All FFmpeg Formats and Codecs Supported
- [Full Documentation](#) with Examples (Doxygen Generated)

License

Copyright (c) 2008-2015 OpenShot Studios, LLC <http://www.openshotstudios.com/>.

OpenShot Library (**libopenshot**) is free software: you can redistribute it and/or modify it under the terms of the GNU Lesser General Public License as published by the Free Software Foundation, either version 3 of the License, or (at your option) any later version.

OpenShot Library (**libopenshot**) is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU Lesser General Public License for more details.

You should have received a copy of the GNU Lesser General Public License along with OpenShot Library. If not, see <http://www.gnu.org/licenses/>.
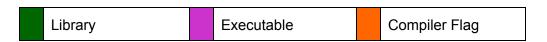
Getting Started

The best way to get started with libopenshot, is to learn about our build system, obtain all the source code, install a development IDE and tools, and better understand our dependencies. So, please read through the following sections, and follow all the instructions. And keep in mind, that your computer is likely different than the one used when writing these instructions. Your file paths and versions of applications might be slightly different, so keep an eye out for subtle file path differences in the commands you type.

Build Tools

CMake is the backbone of our build system. It is a cross-platform build system, which checks for dependencies, locates header files and libraries, generates makefiles, and supports the cross-platform compiling of libopenshot and libopenshot-audio. CMake uses an **out-of-source** build concept, where all temporary build files, such as makefiles, object files, and even the final binaries, are created outside of the source code folder, inside a /build/ folder. This prevents the build process from cluttering up the source code. These instructions have only been tested with the GNU compiler (including MinGW for Windows).

Dependencies

The following libraries are required to build libopenshot. Instructions on how to install these dependencies are listed under each operating system below. Libraries and Executables have been color-coded in the list below to help distinguish between them.

Library	Executable	Compiler Flag

- **FFmpeg** (libavformat, libavcodec, libavutil, libavdevice, libavresample, libswscale)
    - http://www.ffmpeg.org/
    - This library is used to decode and encode video, audio, and image files. It is also used to obtain information about media files, such as frame rate, sample rate, aspect ratio, and other common attributes.

- **ImageMagick++** (libMagick++, libMagickWand, libMagickCore)
    - http://www.imagemagick.org/script/magick++.php
    - This library is used to decode and encode, resize, composite, and apply effects to images.

- **OpenShot Audio Library** (libopenshot-audio)
    - This library is used to mix, resample, host plug-ins, and play audio. It is based on the JUCE project, which is an outstanding audio library used by many different applications

- **Qt 5** (libqt5)
  - http://www.qt.io/qt5/
  - Qt5 is used to display video, store image data, composite images, apply image effects, and many other utility functions, such as file system manipulation, high resolution timers, JSON parsing, etc...

- **CMake** (cmake)
  - http://www.cmake.org/
  - This executable is used to automate the generation of Makefiles, check for dependencies, and is the backbone of libopenshot's cross-platform build process.

- **SWIG** (swig)
  - http://www.swig.org/
  - This executable is used to generate the Python and Ruby bindings for libopenshot. It is a simple and powerful wrapper for C++ libraries, and supports many languages.

- **Python** (libpython)
  - http://www.python.org/
  - This library is used by swig to create the Python bindings for libopenshot. This is also the official language used by OpenShot Video Editor (a graphical interface to libopenshot).

- **Doxygen** (doxygen)
  - http://www.stack.nl/~dimitri/doxygen/
  - This executable is used to auto-generate the documentation used by libopenshot.

- **UnitTest++** (libunittest++)
  - https://github.com/unittest-cpp/
  - This library is used to execute unit tests for libopenshot. It contains many macros used to keep our unit testing code very clean and simple.

- **OpenMP** (-fopenmp)
  - http://openmp.org/wp/
  - If your compiler supports this flag (GCC, Clang, and most other compilers), it provides libopenshot with easy methods of using parallel programming techniques to improve performance and take advantage of multi-core processors.

Obtaining Source Code

The first step in installing libopenshot is to obtain the most recent source code. The source

code is available on launchpad.net/libopenshot/. You can pull the latest version using the Bazaar source control system, which is very similar to **Git**. There are two separate libraries you need to compile: **libopenshot-audio** and **libopenshot**. You will need to pull the source code for both libraries with the following commands.

- **bzr branch lp:libopenshot**
- **bzr branch lp:~openshot.code/libopenshot/libopenshot-audio**

Folder Structure

The source code is divided up into the following folders.

build/	This folder needs to be manually created, and is used by *cmake* to store the temporary build files, such as makefiles, as well as the final binaries (library and test executables).
cmake/	This folder contains custom modules not included by default in cmake, used to find dependency libraries and headers and determine if these libraries are installed.
doc/	This folder contains documentation and related files, such as logos and images required by the doxygen auto-generated documentation.
include/	This folder contains all headers (*.h) used by libopenshot.
src/	This folder contains all source code (*.cpp) used by libopenshot.
tests/	This folder contains all unit test code. Each class has it's own test file (*.cpp), and uses UnitTest++ macros to keep the test code simple and manageable.
thirdparty/	This folder contains code not written by the OpenShot team. For example, jsoncpp, an open-source JSON parser.

Developer Tools (IDE)

While any text editor and terminal will allow you to make changes and compile libopenshot, we recommend the following configuration (which works on Mac, Windows, and Linux).

- **Eclipse or Aptana 3.x**
  - http://www.eclipse.org/ or http://www.aptana.com/
  - Once Eclipse or Aptana is installed, you will need to install the *C/C++ Development Tools (CDT)*, using the Help->Install New Software menu.
  - Once the C++ tools are installed, you can switch to the C/C++ perspective, which arranges the windows and toolbars to the appropriate configuration.

- ○ Next, we need to import the Eclipse/Aptana project file into your *Project Explorer* window (on the left of the screen). Right click in the *Project Explorer*, choose *Import…-> General-> Existing Projects into Workspace*. Choose the libopenshot folder on your computer, and click *Finish*. This will load all the files in libopenshot into your *Project Explorer*.
- ○ Before you can compile / build libopenshot, you must first launch *cmake*, which you can do easily from Eclipse/Aptana. In the *C/C++ perspective*, on the far right side of the screen you should have a tab called *Make Targets*. On this tab, you will see different targets for each operating system (Mac, Windows, and Linux). Double click the one that matches your operating system. The *Debug* versions create debugging symbols, the *Release* versions do not create debug symbols. These *Make Targets* will launch *cmake*, which generates the makefiles, which contain the final commands to compile/build libopenshot.
- ○ Click the *Build* button (i.e hammer icon) on the top toolbar to compile libopenshot. If you are using Windows, you will need to choose the "Windows Debug" build configuration, which uses the *mingw32-make* command instead of the *make* command.
- ○ Here is a screenshot of Aptana 3.x configured correctly with libopenshot and the C/C++ tools:

Now that you have an overview of the project, structure of the source code, and recommended tools, jump to the instructions for your **operating system**:
- ➢ Linux Build Instructions
- ➢ Windows Build Instructions
- ➢ Mac Build Instructions

Linux Build Instructions

These build instructions are for a Debian-based Linux distribution. Other Linux distributions should be very similar, but might also be subtly different. So, exercise caution if you are using a non-Debian distribution.

Install Dependencies (via Package Manager)

Most packages needed by libopenshot can be installed easily with a package manager. Install the following **packages** using apt-get, Synaptic, or similar package manager:

- build-essential
- cmake
- libavformat-dev
- libavdevice-dev
- libswscale-dev
- libavresample-dev
- qt5-default
- qtbase5-dev
- qt5-qmake
- qtmultimedia5-dev
- libmagick++-dev
- python3-dev
- libunittest++-dev
- swig
- doxygen
- libxinerama-dev
- libxcursor-dev
- libasound2-dev

Build & Install libopenshot-audio (Dependency)

Since libopenshot-audio is not available in a Debian package, we need to go through a few additional steps to manually build and install it.
- Branch the latest version: `$ bzr branch lp:~openshot.code/libopenshot/libopenshot-audio`
- Launch a terminal, and enter:
    - `$ cd /home/user/libopenshot-audio/`
    - `$ mkdir build`
    - `$ cd build`
    - `$ cmake ../`
    - `$ make`
- If you are missing any dependencies for libopenshot-audio, you might receive error

messages at this point. Just install the missing packages (usually with a -dev suffix), and run the above commands again. Repeat until no error messages are displayed, and the build process completes.
- Once libopenshot-audio has been successfully built, we need to **install it** (i.e. copy it to the correct folder, so other libraries can find it)
  - `$ sudo make install`
- If everything worked correctly, this command should play a test sound:
  - `$ ./src/openshot-audio-test-sound`

Build & Install OpenShot Video Library (libopenshot) on Linux

The first step in installing libopenshot is to obtain the most recent source code. The source code is available on launchpad.net/libopenshot.

- Branch the latest version: `$ bzr branch lp:libopenshot`
- Launch a terminal, and enter:
  - `$ cd /home/user/libopenshot/`
  - `$ mkdir build`
  - `$ cd build`
  - `$ cmake ../`
  - `$ make`
- If you are missing any dependencies for libopenshot, you might receive error messages at this point. Just install the missing packages (usually with a -dev suffix), and run the above commands again. Repeat until no error messages are displayed, and the build process completes. Also, if you manually install Qt 5, you might need to specify the location for cmake:
  `$ cmake -DCMAKE_PREFIX_PATH=/qt5_path/qt5/5.2.0/ ../`
- To run all **unit tests** (and verify everything is working correctly), launch a terminal, and enter:
  - `$ make test`
  - If all tests pass, then you have successfully built libopenshot!
- To auto-generate the **documentation** for libopenshot, launch a terminal, and enter:
  - `$ make doc`
  - This will use doxygen to generate a folder of HTML files, with all classes and methods documented. The folder is located at **/home/user/libopenshot/build/doc/html/**
- Once libopenshot has been successfully built, we need to install it (i.e. copy it to the correct folder, so other libraries can find it)
  - `$ sudo make install`
  - This should copy the binary file to /usr/local/lib/libopenshot.so, and the header files to /usr/local/include/openshot/... This is where other projects will likely look for the libopenshot files when building.
- The Python bindings are located in /libopenshot/build/src/bindings/python/. To test

them, launch a terminal, and enter:
- ○ `$ cd /home/user/libopenshot/build/src/bindings/python/`
- ○ `$ python`
- ○ `>>> import openshot`
- ○ If no errors are displayed, you may now access all the classes and methods of libopenshot through the openshot module in Python.

Windows Build Instructions

These build instructions are for Windows 7, but should work for other versions of Windows as well. There are many additional steps when using Windows to build libopenshot, and will require the downloading and installation of many different applications, libraries, and tools. These instructions are based on the MinGW (Minimalist GNU for Windows) compiler. Visual Studio can **not** be used with these instructions. In fact, libraries that are built with Visual Studio can **not** be linked correctly using MinGW, so all dependency libraries need to be built with MinGW as well.

TIP: When running commands in the command line or other terminal shells, be sure and "Run as administrator" whenever possible. Many of these commands will create folders or copy files, and might not have permission. Also, I recommend using a better terminal than cmd.exe, for example Clink.

Install Dependencies (Manually)

Unfortunately, Windows does not have a simple way to install commonly used libraries and applications, similar to a Linux package manager. So, you will need to manually find, download, and install each dependency application and library, making sure the binaries are compiled with MinGW (and not Visual Studio). In a few cases, you will have to compile a library manually with MinGW, if no suitable MinGW binaries can be found online. This is going to take a while, so be sure you have some free time before continuing beyond this point. Good luck!

- ● **CMake**
  - ○ http://www.cmake.org/cmake/resources/software.html
  - ○ Download and Install the Win32 Installer
  - ○ This will install a new executable that you can use from the command line: "cmake". If the command is not found, you might need to add the cmake folder to your PATH environment variable.

- ● **FFmpeg / Libav**
  - ○ http://builds.libav.org/windows/release-lgpl/
  - ○ Both the FFmpeg and Libav libraries are supported by libopenshot, but currently, it is easier to install Libav on Windows.

- ○ Download and Install the version for either 32-bit or 64-bit Windows.
- ○ Create an environment variable called **FFMPEGDIR** and set the value to **C:\libav-folder-name\usr** (your path might be different). This is the folder which contains both the 'include' folder (for headers) and 'bin' folder (for dlls).
- ○ This environment variable will be used by CMake to find the binary and header files.

- ● **Qt 5**
  - ○ http://qt-project.org/downloads
  - ○ Download and Install the newest MinGW version (for example: http://download.qt-project.org/official_releases/qt/5.1/5.1.1/qt-windows-opensource-5.1.1-mingw48_opengl-x86-offline.exe).
  - ○ Add the Qt install folder to your PATH (if it is not there after installation). For example: c:\Qt\5.1.1\bin.
  - ○ **Troubleshooting Qt5:**
    - ■ If you receive compile errors related to Qt5, you might have to manually fix a bug in the Qt headers (your path might be a bit different).
    - ■ Comment out the following line in C:/Qt/Qt5.1.1/5.1.1/mingw48_32/include/QtGui/qopenglversionfunctions.h
    - ■ // void (QOPENGLF_APIENTRYP MemoryBarrier)(GLbitfield barriers);
    - ■ Eventually this bug will be fixed on upstream Qt, but until that happens, this work-around should solve any compile errors.

- ● **MinGW** (Minimalist GNU for Windows)
  - ○ A version of MinGW is included with Qt 5. Be sure to add the /bin/ folder containing to the MinGW executable (mingw32-make) to the PATH environment variable.
  - ○ Once installed, you should be able to run the following command from the command line:
    - ■ $ mingw32-make

- ● **Python3**
  - ○ http://www.python.org/download/
  - ○ Download and Install the Python 3.X Windows Installer (32bit version / x86). This will also install the headers and library files for Python.
  - ○ **Troubleshooting Python:**
    - ■ If cmake cannot find PythonLibs, it might be due to the following bug in cmake's FindPythonLibs.cmake file. It seems that in some versions of cmake, the find_library command does not parse registry paths.
    - ■ Please see our **CMake Flags** section below for tips on manually pointing to your Python3 include folder and library.
    - ■ Or, here is a workaround to fix the above bug. Add the following line to FindPythonLibs.cmake inside the first foreach loop:

- GET_FILENAME_COMPONENT(HKEY_LOCAL_MACHINE_PATH "[HKEY_LOCAL_MACHINE\\SOFTWARE\\Python\\PythonCore\\${_CURRENT_VERSION}\\InstallPath]" ABSOLUTE)
- Then, add ${HKEY_LOCAL_MACHINE_PATH} to the PATHS on any **find_library** commands you find in the file. This should correctly parse the registry entry, and search that path for the dlls. Also, be sure to include the correct suffix: /libs or /include.

- **Swig**
  - http://www.swig.org/download.html
  - Download and Install the "swigwin" Windows installer. This will install a new command that you can use from the command line: "swig". If the command is not found, you might need to add the swig folder to your PATH environment variable.

- **Doxygen**
  - http://www.stack.nl/~dimitri/doxygen/download.html#latestsrc
  - Download and run the Windows Installer. This will install a new executable that you can use from the command line: "doxygen". If the command is not found, you might need to add the doxygen folder to your PATH environment variable.

- **UnitTest++**
  - https://github.com/unittest-cpp/unittest-cpp/
  - Download and Extract the UnitTest++ source code .ZIP to a local folder: **C:\UnitTest++**. No binary files are available to download, so we will need to build the source code. The source code comes with Visual Studio project files, but MinGW will be unable to link a Visual Studio DLL with libopenshot. So, we need to manually build UnitTest++ with MinGW and CMake.
  - Create a folder called \build\ inside the \UnitTest++\ folder: **C:\UnitTest++\build**
    - This will be used to store the final binary of libUnitTest++.dll
  - Copy and rename a file from the libopenshot source code into the \UnitTest++\ folder.
    - C:\libopenshot\cmake\Modules\UnitTest++_CMakeLists.txt **-->** C:\UnitTest++\CMakeLists.txt
    - This file (CMakeLists.txt) will be used by CMake to compile UnitTest++. Be sure to rename the file.
  - Launch a terminal, and enter:
    - `$ cd C:\UnitTest++\build\`
    - `$ cmake -G "MinGW Makefiles" ../`
    - `$ mingw32-make`
  - Create an environment variable called **UNITTEST_DIR** and set the value to **C:\UnitTest++**
  - This environment variable will be used by CMake to find the binary and header

files.

- **MSYS**
  - http://downloads.sourceforge.net/mingw/MSYS-1.0.11.exe
  - MSYS is a collection of GNU utilities, commands, and a shell that allows compiling Linux applications with MinGW. ImageMagick++ does not have any MinGW compiled binaries available to download, and MSYS is the only way I've been successful building it on Windows with MinGW. It is a fairly straight forward process, but a little confusing if it's your first time learning about MSYS. However, the basic idea is that MSYS gives you a terminal shell that can understand linux file paths, and common linux commands (cp, mv, make, etc…), but the compiling of source code still uses MinGW.
  - Download and Install the MSYS (exe) listed above.
  - Enter "**C:\msys\1.0**", when prompted where to install MSYS.
  - Enter "**C:\mingw**", when prompted where MinGW is installed.
  - Install the MSYS DTK (http://downloads.sourceforge.net/mingw/msysDTK-1.0.1.exe) in the **C:\msys\1.0** folder**.**
  - Install the MSYS Core (http://sourceforge.net/projects/mingw/files/MSYS/BaseSystem/msys-1.0.11/msysCORE-1.0.11-bin.tar.gz/download). It is an archive. Extract it in the **C:\msys\1.0** folder.
  - Set the environment variable HOME to **C:\msys\1.0\home**
  - Now, you should have a shortcut to the MSYS shell on your desktop, and in your program list. This will launch the MSYS shell, which behaves very similar to a Linux terminal. This shell is required to install ImageMagick++, because their build process uses Linux file paths and Linux commands which do not work in the Windows command line.

- **DLfcn**
  - https://github.com/dlfcn-win32/dlfcn-win32
  - Download and Extract the Win32 Static (.tar.bz2) archive to a local folder: **C:\libdl**
  - Create an environment variable called **DL_DIR** and set the value to **C:\libdl**
  - This environment variable will be used by CMake to find the binary and header file.

- **Zlib**
  - http://gnuwin32.sourceforge.net/packages/zlib.htm
  - Download and Install the Windows Setup program.

- **FreeType**
  - http://gnuwin32.sourceforge.net/packages/freetype.htm
  - Download and Install the Windows Setup program.
  - Create an environment variable called **FREETYPE_DIR** and set the value to

C:\Program Files\GnuWin32
- ○ This environment variable will be used by CMake to find the binaries and header files.

- **ImageMagick++**
  - ○ ftp://ftp.imagemagick.org/pub/ImageMagick/ImageMagick.tar.gz
  - ○ First, be sure you have installed and configured MSYS before attempting to install ImageMagick. Because ImageMagick must be compiled with MinGW (and **not** VisualStudio), a working MSYS shell is needed. Download and Extract the ImageMagick source code into your MSYS home directory: **C:\msys\1.0\home\USER\ImageMagick-6.7.1-0** (your path might be different)
  - ○ Launch the MSYS shell, and enter:
    - ■ NOTE: Copy the cmake/Windows/**build-imagemagick.sh** and **urls.txt** to a folder on Msys and execute the .sh file. This will automate the downloading, extracting, and installing of ImageMagick for Windows. Also, it must link to your Qt5's included version of MinGW. If this works, it should create the ImageMagick Dlls, and you can skip to the next step. If this fails, continue to manually build ImageMagick with the following commands.

    - ■ `$ cd /home/USER/ImageMagick*`
    - ■ `$ $ ./configure --enable-shared=yes --enable-static=no --with-x --prefix=/mingw LDFLAGS=-static-libstdc++`
    - ■ `$ make`
    - ■ `$ make install`
    - ■ If you get any errors, you might need to disable certain features, or install additional libraries needed by ImageMagick. To see all configure options, enter **./configure --help**
    - ■ The final step is to create the following registry key: [HKEY_LOCAL_MACHINE\SOFTWARE\ImageMagick\Current;BinPath] and set the value to **C:/MinGW/msys/1.0/local**
    - ■ This registry key will be used by CMake to locate the final ImageMagick binaries, and header files. **C:/MinGW/msys/1.0/local** should have an \include\ and \lib\ folder which contain ImageMagick files, once the "make install" command is run.
  - ○ **Troubleshooting cmake:**
    - ■ If cmake cannot find ImageMagick, it might be due to the following bug in cmake's FindImageMagick.cmake file. It seems that in some versions of cmake, the **find_library** command does not parse registry paths. Thus, the registry path entered above, does not get searched for the newly

compiled ImageMagick dlls. Here is a work around to fix that bug. Add the following line to FindImageMagick.cmake:

- GET_FILENAME_COMPONENT(REGISTRY_PATH "[HKEY_LOCAL_MACHINE\\SOFTWARE\\ImageMagick\\Current;BinPath]/lib" ABSOLUTE)
- Then, add **${REGISTRY_PATH}** to the PATHS on the first **find_library** command you find in the file. This should correctly parse the registry entry, and search that path for the dlls.

- **DirectX SDK**
  - http://www.microsoft.com/download/en/details.aspx?displaylang=en&id=6812
  - Download and Install the DirectX SDK Setup program. This is needed for the JUCE library to play audio on Windows.
  - Create an environment variable called **DXSDK_DIR** and set the value to **C:\Program Files\Microsoft DirectX SDK (June 2010)** (your path might be different)
  - This environment variable will be used by CMake to find the binaries and header files.

- **libSndFile**
  - http://www.mega-nerd.com/libsndfile/#Download
  - Download and Install the Win32 Setup program.
  - Create an environment variable called **SNDFILE_DIR** and set the value to **C:\Program Files\libsndfile**
  - This environment variable will be used by CMake to find the binary and header files.

- **libopenshot-audio**
  - Since libopenshot-audio is not available in an installer, we need to go through a few additional steps to manually build and install it.
  - Branch the latest version: `$ bzr branch lp:~openshot.code/libopenshot/libopenshot-audio`
  - Launch a terminal, and enter:
    - `$ cd /libopenshot-audio/`
    - `$ mkdir build`
    - `$ cd build`
    - `$ cmake -G "MinGW Makefiles" ../`
    - `$ mingw32-make`

  - If you are missing any dependencies for libopenshot-audio, you might receive error messages at this point. Just install the missing libraries, and run the above commands again. Repeat until no error messages are displayed, and the build process completes. If you get stuck at this point, take a look at the

CMakeLists.txt file inside the \libopenshot-audio\ folder for clues on how it's trying to locate dependencies, such as environment variables, registry keys, etc…

- If you are trying to build with Qt 5's version of MinGW, which is a **64 bit** version of MinGW, you "**might**" need to fix one of the header files in MinGW. Replace the contents of *iphlpapi.h* with this file: https://gitorious.org/w64/w64/raw/8debddd9a43d7b8d841d3ebc1dd8a46782eac120:mingw-w64-headers/include/iphlpapi.h
- Once libopenshot-audio has been successfully built, we need to install it (i.e. copy it to the correct folder, so other libraries can find it)
  - `$ mingw32-make install`
  - This should copy the binary file to C:\Program Files\Project\lib\libopenshot-audio.dll, and the header files to C:\Program Files\Project\include\libopenshot-audio\... This is where libopenshot will ultimately look for the libopenshot-audio files when building.
- If everything worked correctly, this command should play a test sound:
  - `$ openshot-audio-test-sound`
- Create an environment variable called **LIBOPENSHOT_AUDIO_DIR** and set the value to **C:\Program Files\Project\libopenshot-audio** (your path might be different)
- This environment variable will be used by CMake to find the binary and header files needed by libopenshot.

Build & Install OpenShot Video Library (libopenshot) on Windows

The first step in installing libopenshot is to obtain the most recent source code. The source code is available on launchpad.net/libopenshot.

- Branch the latest version: `$ bzr branch lp:libopenshot`
- Launch a terminal, and enter:
  - `$ cd C:\libopenshot\`
  - `$ mkdir build`
  - `$ cd build`
  - `$ cmake -G "MinGW Makefiles" -DPYTHON_INCLUDE_DIR="C:/Python34/include/" -DPYTHON_LIBRARY="C:/Python34/libs/libpython34.a" ../`
  - `$ mingw32-make`
- If you are missing any dependencies for libopenshot, you might receive error messages at this point. Just install the missing dependencies, and run the above commands again. Repeat until no error messages are displayed, and the build process completes. Also, be sure to point to your correct Python file paths in the above command.
- To run all **unit tests** (and verify everything is working correctly), launch a terminal, and enter:
  - `$ mingw32-make test`

- If all tests pass, then you have successfully built libopenshot!
- To auto-generate the **documentation** for libopenshot, launch a terminal, and enter:
  - `$ mingw32-make doc`
  - This will use doxygen to generate a folder of HTML files, with all classes and methods documented. The folder is located at **/home/user/libopenshot/build/doc/html/**.
- Once libopenshot has been successfully built, we need to install it (i.e. copy it to the correct folder, so other libraries can find it)
  - `$ mingw32-make install`
  - This should copy the binary file to C:\Program Files\openshot\lib\libopenshot.dll, and the header files to C:\Program Files\openshot\include\... This is where other projects will likely look for the libopenshot files when building.
- The Python bindings are located in /libopenshot/build/src/. To test them, launch a terminal, and enter:
  - `$ cd C:\YOURPROJECTS\libopenshot\build\src\`
  - `$ python`
  - `>>> import openshot`
  - If no errors are displayed, you may now access all the classes and methods of libopenshot through the openshot module in Python.

Environment Variables

Many environment variables will need to be set during this Windows installation guide. The command line will need to be closed and re-launched after any changes to your environment variables. Also, dependency libraries will not be found during linking or execution without being found in the PATH environment variable. So, if you get errors related to missing commands or libraries, double check the PATH variable.

The following environment variables need to be added to your "System Variables". Be sure to check each folder path for accuracy, as your paths will likely be different than this list.

Example Variables

DL_DIR	C:\libdl
DXSDK_DIR	C:\Program Files\Microsoft DirectX SDK (June 2010)\
FFMPEGDIR	C:\ffmpeg-git-95f163b-win32-dev
FREETYPE_DIR	C:\Program Files\GnuWin32
HOME	C:\msys\1.0\home
LIBOPENSHOT_AUDIO _DIR	C:\Program Files\libopenshot-audio

QTDIR	C:\qt5
SNDFILE_DIR	C:\Program Files\libsndfile
UNITTEST_DIR	C:\UnitTest++
PATH	The following paths were appended: C:\Qt5\bin; C:\Qt5\MinGW\bin\; C:\msys\1.0\local\lib; C:\Program Files\CMake 2.8\bin; C:\UnitTest++\build; C:\libopenshot\build\src; C:\Program Files\doxygen\bin; C:\ffmpeg-git-95f163b-win32-dev\lib; C:\swigwin-2.0.4; C:\Python33; C:\Program Files\Project\lib;

Mac Build Instructions

These build instructions are for Mac OS X (Mountain Lion). Other OS X versions should be very similar, but might also be subtly different. So, exercise caution if you are using a non-Mountain Lion version.

Install Dependencies (Homebrew)

Most packages needed by libopenshot can be installed easily with MacPorts or Homebrew. However, before MacPorts can be installed, you must first install Xcode with the following options ("UNIX Development", "System Tools", "Command Line Tools", or "Command Line Support"). Be sure to refresh your list of MacPort packages with the "**sudo port selfupdate**" command and update your Homebrew packages with "**brew update**".

NOTE: Homebrew seems to work much better for most users, so I am going to focus on brew for the rest of this guide.

Install the following **packages** using the Homebrew package installer (http://brew.sh/). Pay close attention to any warnings or errors during these brew installs. NOTE: You might have some conflicting libraries in your /usr/local/ folders, so follow the directions from brew if these are detected.

- **brew install gcc48 --enable-all-languages**
- **brew install ffmpeg**
- **brew install librsvg**
- **brew install imagemagick --with-librsvg**
- **brew install swig**
- **brew install doxygen**
- **brew install unittest-cpp --cc=gcc-4.8**. You must specify the c++ compiler with the --cc

flag to be 4.7 or 4.8.
- **brew install qt5**
- **brew install cmake**

Install the following packages manually (because Homebrew does not have them):
- X11 (XQuartz) - http://xquartz.macosforge.org/landing/
  - This library is used to display images with ImageMagick and libopenshot. Some version of Mac OS X include X11, and some do not. You can install the newest version of XQuartz from the link above.

Build & Install libopenshot-audio (Dependency)

Since libopenshot-audio is not available in a Homebrew or MacPorts package, we need to go through a few additional steps to manually build and install it.

- Branch the latest version: `$ bzr branch lp:~openshot.code/libopenshot/libopenshot-audio`
- Launch a terminal and enter:
  - `$ cd /home/user/libopenshot-audio/`
  - `$ mkdir build`
  - `$ cd build`
  - `$ cmake -d -G "Unix Makefiles" -DCMAKE_CXX_COMPILER=clang++ -DCMAKE_C_COMPILER=clang ../` **(CLang must be used due to GNU incompatible Objective-C code in some of the Apple frameworks)**
  - `$ make`
  - `$ sudo make install`
  - `$ ./src/openshot-audio-test-sound` (This should play a test sound)

Build & Install OpenShot Video Library (libopenshot) on Mac

The first step in installing libopenshot is to obtain the most recent source code. The source code is available on launchpad.net/libopenshot.

- Branch the latest version: `$ bzr branch lp:libopenshot`
- Launch a terminal, and enter:
  - `$ cd /home/user/libopenshot/`
  - `$ mkdir build`
  - `$ cd build`
  - `$ cmake -G "Unix Makefiles" -DCMAKE_CXX_COMPILER=/usr/local/opt/gcc48/bin/g++-4.8 -DCMAKE_C_COMPILER=/usr/local/opt/gcc48/bin/gcc-4.8 -DCMAKE_PREFIX_PATH=/usr/local/Cellar/qt5/5.4.2/ -DPYTHON_INCLUDE_DIR=/usr/local/Cellar/python3/3.3.2/Frameworks/Python.framework/Versions/3.3/include/python3.3m/`

```
-DPYTHON_LIBRARY=/usr/local/Cellar/python3/3.3.2/Framework
s/Python.framework/Versions/3.3/lib/libpython3.3.dylib
-DPython_FRAMEWORKS=/usr/local/Cellar/python3/3.3.2/Framew
orks/Python.framework/ ../
-D"CMAKE_BUILD_TYPE:STRING=Debug"
```
The extra arguments on the cmake command make sure the compiler will be gcc4.8 and that cmake knows where to look for the Qt header files and Python library. Double check these file paths, as yours might be different.

- `$ make`
- If you are missing any dependencies for libopenshot, you will receive error messages at this point. Just install the missing dependencies, and run the above commands again. Repeat until no error messages are displayed and the build process completes.
- Also, if you are having trouble building, please see the **CMake Flags** section below, as it might provide a solution for finding a missing folder path, missing Python library, etc...
- To run all **unit tests** (and verify everything is working correctly), launch a terminal, and enter:
  - `$ make test`
  - If all tests pass, then you have successfully built libopenshot! Congratulations!
- To auto-generate the **documentation** for libopenshot, launch a terminal, and enter:
  - `$ make doc`
  - This will use doxygen to generate a folder of HTML files, with all classes and methods documented. The folder is located at **/home/user/libopenshot/build/doc/html/**.
- Once libopenshot has been successfully built, we need to install it (i.e. copy it to the correct folder, so other libraries can find it)
  - `$ sudo make install`
  - This should copy the binary file to /usr/local/lib/libopenshot.so, and the header files to /usr/local/include/openshot/... This is where other projects will likely look for the libopenshot files when building.
- The Python bindings are located in /libopenshot/build/src/bindings/python/. To test them, launch a terminal, and enter:
  - `$ cd /home/user/libopenshot/build/src/bindings/python/`
  - `$ python` (or `python3`)
  - `>>> import openshot`
  - If no errors are displayed, you can now access all the classes and methods of libopenshot through the openshot module in Python.

CMake Flags (Optional)

There are many different build flags that can be passed to cmake to adjust how libopenshot is compiled. Some of these flags might be required when compiling on certain OSes, just depending on how your build environment is setup. To add a build flag, follow this general syntax: `$ cmake -DMAGICKCORE_HDRI_ENABLE=1 -DENABLE_TESTS=1 ../`

MAGICKCORE_HDRI_ENABLE	0
MAGICKCORE_QUANTUM_DEPTH	16
OPENSHOT_IMAGEMAGICK_COMPATIBILITY	0
DISABLE_TESTS	0
CMAKE_PREFIX_PATH	/location/to/missing/library/
PYTHON_INCLUDE_DIR	/location/to/python/include/
PYTHON_LIBRARY	/location/to/python/lib.a
PYTHON_FRAMEWORKS	/usr/local/Cellar/python3/3.3.2/Frameworks/Python.framework/
CMAKE_CXX_COMPILER	/location/to/mingw/g++
CMAKE_C_COMPILER	/location/to/mingw/gcc

Installers / Packaging

Many different installers and packages (as they are called on Linux and Mac) have been created for libopenshot and libopenshot-audio. The source code for these packages is located in a separate Bazaar branch.

Packaging Bazaar Branch

https://code.launchpad.net/~openshot.packagers/libopenshot/packaging.

Daily PPA for Ubuntu

For Ubuntu, we also have a **daily** built repository / PPA for libopenshot and libopenshot-audio: https://launchpad.net/~openshot.developers/+archive/ubuntu/libopenshot-daily

PyQt Interface (Optional)

Once you have successfully installed libopenshot and libopenshot-audio, you now have the

option of installing our PyQt5 desktop video editor: OpenShot Video Editor 2.0. Here are the general instructions for installing OpenShot 2.0. *Note: OpenShot 1.x does **not** use PyQt or libopenshot (and is only compatible with Linux).*

Obtaining Source Code

The first step to installing our desktop video editor (i.e. OpenShot Video Editor 2.0), is to download the source code. And luckily, this source code is Python and JavaScript, and does not need to be compiled. The source code is available on LaunchPad, but is still highly experimental and has many known issues.

Launch a terminal, and enter:
- $ bzr branch lp:openshot/2.0/ openshot-qt

Folder Structure

The source code is divided up into the following folders.

bin/	This folder contains the script that launches OpenShot 2.0
doc/	This folder contains documentation and related files, such as logos and images required by the doxygen auto-generated documentation.
installer/	This folder contains misc files related to installers, or the creation of installers for OpenShot 2.0.
src/	This folder contains all source code (*.py and *.js) used by libopenshot.
… /blender/	This folder contains the Blender 3D animation files, used for animated titles and other 3D effects.
… /classes/	This folder contains most of our primary classes and utility Python methods. Such as our app.py, info.py, and other top level functions.
… /effects/	This folder contains the meta data for our effects. Such as effect thumbnails, XML files, etc...
… /images/	This folder contains misc images used by OpenShot 2.0 (i.e. logo).
… /locale/	This folder contains language translations for OpenShot 2.0.
… /presets/	This folder contains presets for our export dialog. The presets are XML files that describe common combinations of export settings, such as DVD NTSC.
… /profiles/	This folder contains profiles for different screen resolutions, frame rates, aspect ratios, etc...

… /settings/	This folder contains the default settings file used by OpenShot 2.0, as well as the default timeline JSON file.
… /tests/	This folder contains Python unit tests for OpenShot 2.0.
… /timeline/	This folder contains the JavaScript source code and HTML used by the timeline widget. It uses Angular.js and JQuery UI to bind to the timeline's JSON structure and render the timeline. This HTML can actually be tested and debugged outside of the PyQt application, using a recent version of Google Chrome.
… /titles/	This folder contains the SVG vector titles uses by OpenShot 2.0.
… /transitions/	This folder contains the grayscale transition images, used to fade between different clips / videos.
… /uploads/	This folder contains the YouTube and Vimeo source code, to upload exported videos directly to a user's account.
… /windows/	This folder contains all the .UI files and Window classes used by OpenShot 2.0. Much of our source code is inside this folder, as most of our source code relates to windows, dialogs, and interfaces.
xdg/	This folder contains some Linux specific files, related to packaging, such as the .desktop launcher file, icon, and mime type data.

Linux Instructions

Here is a list of the **packages** you need to install via your Linux package manager. These are the package names on Ubuntu, so your OS might have different names for these packages.

- python3
- python3-pyqt5
- python3-pyqt5.qtmultimedia
- python3-pyqt5.qtwebkit
- libqt5svg5
- python3-simplejson

Windows Instructions

To run our desktop video editor on Windows, you need to install the frameworks and dependencies that are required, such as PyQt5 and Python3.

- **PyQt** - http://www.riverbankcomputing.co.uk/software/pyqt/download5
  - Be sure to download the **.exe** for your correct architecture (32 bit vs 64 bit)

- - You also need to be sure to install the version of PyQt5 that matches your Qt version from the previous steps above. So, if you previously installed Qt 5.3, be sure to install the version of PyQt that is based on Qt 5.3.
  - **Python3**
    - http://www.python.org/download/
    - Download and Install the Python 3.X Windows Installer (32bit version / x86). This will also install the headers and library files for Python.

Mac Instructions

These packages are needed by OpenShot 2.0, and can be installed easily with MacPorts or Homebrew. *NOTE: Homebrew seems to work much better for most users, so I am going to focus on **brew** for the rest of this guide.*

Install the following **packages** using the Homebrew package installer (http://brew.sh/).

- **brew install python3**
- **brew install pyqt5**

Launching the App

To launch OpenShot 2.0 from source code, you must use the terminal / command line. Navigate to the correct folder: /openshot-qt/, and run the following command:

- `$ python3 src/launch.py`
- or (`python src/launch.py`)

If you see any error messages printed to the terminal, it usually means that a dependency is missing, such as libopenshot or PyQt. Hopefully with some careful googling, you can figure out what is missing and fix it. Good luck!

Build an Installer

To create an installer / redistributable package for OpenShot Video Editor 2.0, we are utilizing a program called "cx_Freeze", which bundles up all the related dependencies, python libraries, Qt libraries, and other required files into a single executable.

- **cx_Freeze** - http://cx-freeze.sourceforge.net/
  - cx_Freeze is a Python application, and can be installed using PIP:
  - $ pip install cx_Freeze

OpenShot Video Editor comes with an easy wrapper for cx_Freeze, called "freeze.py", located in the main source code folder.

- **Linux & Windows** Syntax
  - $ python3 freeze.py build
- **Mac** Syntax, which creates an Application Bundle
  - $ python3 freeze.py bdist_mac --include-frameworks "/usr/local/Cellar/qt5/5.3.1/Frameworks/QtCore.framework,/usr/local/Cellar/qt5/5.3.1/Frameworks/QtGui.framework,/usr/local/Cellar/qt5/5.3.1/Frameworks/QtMultimedia.framework,/usr/local/Cellar/qt5/5.3.1/Frameworks/QtMultimediaWidgets.framework,/usr/local/Cellar/qt5/5.3.1/Frameworks/QtNetwork.framework,/usr/local/Cellar/qt5/5.3.1/Frameworks/QtWidgets.framework" --qt-menu-nib="/usr/local/Cellar/qt5/5.3.1/plugins/platforms/" --iconfile=../openshot.icns --custom-info-plist=installer/Info.plist --bundle-name="OpenShot Video Editor"
  - Change Contents/Info.plist to use **launch-mac.sh** as the Executable name
  - Manually fix RSVG executable:
    - $ sudo dylibbundler -od -of -b -x ~/apps/rsvg/rsvg-convert -d ./rsvg-libs/ -p @executable_path/rsvg-libs/

- **Troubleshooting:** If you encounter an error while attempting to freeze the PyQt5/uic/port_v2, remove the __init__.py in that folder. And if you are manually compiling PyQt5 on Windows, remove the -strip line from the Makefile.

Conclusion

I hope this Installation Guide was useful to you, and I hope you are now enjoying the OpenShot Video Library. Be sure to contact me if you use this library for anything interesting or exciting, as I would love to hear about it and share it with the community. Also, if you are interested in obtaining a commercial license, please contact me for more details: jonathan@openshot.org.

Donations

If you find any use or enjoyment from this library, I encourage you to donate a few dollars to our project. Even the smallest contributions add up and help fund meetings, conferences, travel expenses, and of course, my time to continue to improve this exciting project. For more information on donations, please visit www.openshot.org/donate/. **Thank you so much for your support!**