FFmpeg can be built with MSVC 2012 or earlier using a C99-to-C89 conversion utility and wrapper, or with MSVC 2013 and ICL natively.

You will need the following prerequisites:

* [C99-to-C89 Converter & Wrapper](https://github.com/libav/c99-to-c89/) (if using MSVC 2012 or earlier)
* [msinttypes](http://code.google.com/p/msinttypes/) (if using MSVC 2012 or earlier)
* [MScd YS2](http://msys2.github.io/)
* [YASM](http://yasm.tortall.net/) (Also available via MSYS2’s package manager.)
* To set up a proper environment in MSYS2, you need to run C:\msys32\msys2\_shell.cmd from the Visual Studio or Intel Compiler command prompt.
* Place yasm.exe somewhere in your PATH. If using MSVC 2012 or earlier, place c99wrap.exe and c99conv.exe somewhere in your PATH as well.
* For MSVC:
* ./configure --toolchain=msvc
* cl is unable to create an executable file.
* If cl is a cross-compiler, use the --enable-cross-compile option.
* Only do this if you know what cross compiling means.
* C compiler test failed.

Run which link.exe it will be /usr/bin/link.exe

mv /usr/bin/link.exe /usr/bin/link\_old.exe

and export PATH=$PATH:"/c/Program Files (x86)/Microsoft Visual Studio 10.0/VC/bin"

Now

$ which link.exe

/c/Program Files (x86)/Microsoft Visual Studio 10.0/VC/bin/link.exe

cl.exe: error while loading shared libraries: mspdb100.dll

export PATH=$PATH:"/c/Program Files (x86)/Microsoft Visual Studio 10.0/Common7/IDE"

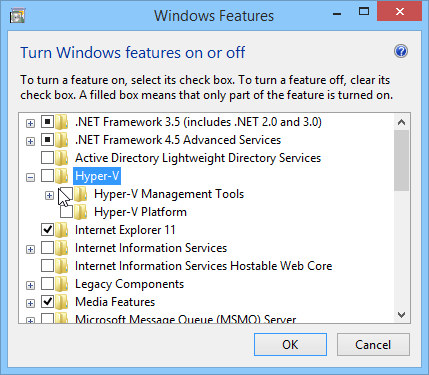
cl.exe: error while loading shared libraries: libcpmt.lib

export PATH=$PATH:"/c/Program Files (x86)/Microsoft Visual Studio 10.0/VC/lib"

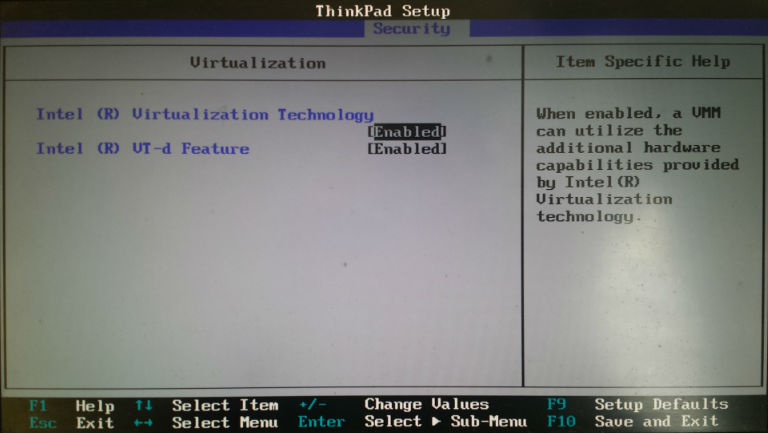
c99wrap cl is unable to create an executable file.

**Why is VirtualBox only showing 32 bit guest versions on my 64 bit host OS?**

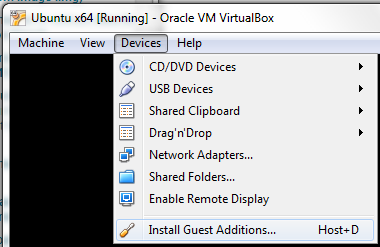
**1 type in cortana “**turn windows features on or off”



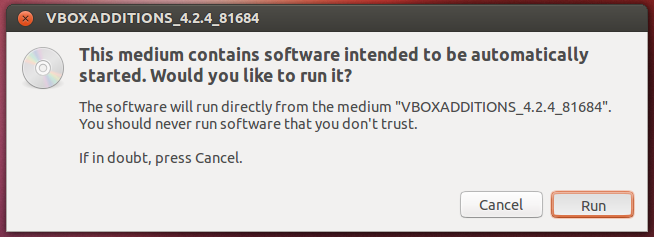
**2**



**Share folder**



**$ sudo mkdir /mount/ShareDir**



**$ sudo mount -t vboxsf WINDIRNAME /media/ShareDir**

<http://www.roman10.net/2013/08/18/how-to-build-ffmpeg-with-ndk-r9/>

This is a updated post for a previous post, where we [built ffmpeg 0.8 with Android NDK r5 and r6](http://www.roman10.net/how-to-build-ffmpeg-for-android/). This post will give instructions of how to build ffmpeg 2.0.1 with Android NDK r9.

**0. Download Android NDK**

The latest version of Android NDK can be downloaded at [Android NDK website](http://developer.android.com/tools/sdk/ndk/index.html). At the time of writing, the newest version is NDK r9. Note that the website provides both current and legacy toolchains. We only need the current toolchain to compile ffmpeg.

After download NDK, simply decompress the archive. Note that we’ll use $NDK to represent the root path of the decompressed NDK.

**1. Download ffmpeg source code**

FFMPEG source code can be downloaded from the [ffmpeg website](http://www.ffmpeg.org/download.html). The latest stable release is 2.0.1. Download the source code and decompress it to $NDK/sources folder. We’ll discuss about the reason for doing this later.

**2. Update configure file**

Open ffmpeg-2.0.1/configure file with a text editor, and locate the following lines.

SLIBNAME\_WITH\_MAJOR='$(SLIBNAME).$(LIBMAJOR)'

LIB\_INSTALL\_EXTRA\_CMD='$$(RANLIB) "$(LIBDIR)/$(LIBNAME)"'

SLIB\_INSTALL\_NAME='$(SLIBNAME\_WITH\_VERSION)'

SLIB\_INSTALL\_LINKS='$(SLIBNAME\_WITH\_MAJOR) $(SLIBNAME)'

This cause ffmpeg shared libraries to be compiled to libavcodec.so.<version> (e.g. libavcodec.so.55), which is not compatible with Android build system. Therefore we’ll need to replace the above lines with the following lines.

SLIBNAME\_WITH\_MAJOR='$(SLIBPREF)$(FULLNAME)-$(LIBMAJOR)$(SLIBSUF)'

LIB\_INSTALL\_EXTRA\_CMD='$$(RANLIB) "$(LIBDIR)/$(LIBNAME)"'

SLIB\_INSTALL\_NAME='$(SLIBNAME\_WITH\_MAJOR)'

SLIB\_INSTALL\_LINKS='$(SLIBNAME)'

**3. Build ffmpeg**

Copy the following text to a text editor and save it as build\_android.sh.

#!/bin/bash

NDK=$HOME/Desktop/adt/android-ndk-r9

SYSROOT=$NDK/platforms/android-9/arch-arm/

TOOLCHAIN=$NDK/toolchains/arm-linux-androideabi-4.8/prebuilt/linux-x86\_64

function build\_one

{

./configure

--prefix=$PREFIX

--enable-shared

--disable-static

--disable-doc

--disable-ffmpeg

--disable-ffplay

--disable-ffprobe

--disable-ffserver

--disable-avdevice

--disable-doc

--disable-symver

--cross-prefix=$TOOLCHAIN/bin/arm-linux-androideabi-

--target-os=linux

--arch=arm

--enable-cross-compile

--sysroot=$SYSROOT

--extra-cflags="-Os -fpic $ADDI\_CFLAGS"

--extra-ldflags="$ADDI\_LDFLAGS"

$ADDITIONAL\_CONFIGURE\_FLAG

make clean

make

make install

}

CPU=arm

PREFIX=$(pwd)/android/$CPU

ADDI\_CFLAGS="-marm"

build\_one

We disabled static library and enabled shared library. Note that the build script is not optimized for a particular CPU. One should refer to ffmpeg documentation for detailed information about available configure options.

Once the file is saved, make sure the script is executable by the command below,

*sudo chmod +x build\_android.sh*

Then execute the script by the command,

*./build\_android.sh*

**4. Build Output**

The build can take a while to finish depending on your computer speed. Once it’s done, you should be able to find a folder $NDK/sources/ffmpeg-2.0.1/android, which contains arm/lib and arm/include folders.

The arm/lib folder contains the shared libraries, while arm/include folder contains the header files for libavcodec, libavformat, libavfilter, libavutil, libswscale etc.

Note that the arm/lib folder contains both the library files (e.g.: libavcodec-55.so) and symbolic links (e.g.: libavcodec.so) to them. We can remove the symbolic links to avoid confusion.

Compile on Windows : <https://pracucci.com/compile-ffmpeg-on-windows-with-visual-studio-compiler.html>

### **MSYS2**

1. Download and run the installer at [​http://msys2.github.io](https://pracucci.com/%E2%80%8Bhttp:/msys2.github.io). Follow the instructions and install it in C:\workspace\windows\msys32
2. Install required tools: pacman -S make gcc diffutils
3. Rename C:\workspace\windows\msys32\usr\bin\link.exe to C:\workspace\windows\msys32\usr\bin\link\_orig.exe, in order to use MSVC link.exe (naming conflict)

## **Ready to compile**

### **Launch msys2 shell from Visual Studio Code shell**

1. Run VS201X prompt
2. Run C:\workspace\windows\msys32\msys\_shell.bat: will open msys2 shell inheriting %PATH%from VS201X prompt
3. Ensure which cl exists and which link points to MSVC

If not found run $ export PATH=$PATH:"/c/Program Files (x86)/Microsoft Visual Studio 14.0/VC/bin" command

1. cd /c/path/to/ffmpeg
2. ./configure and make

Basic configure switches:

./configure \

--toolchain**=**msvc \

--arch**=**x86 \

--enable-yasm \

--enable-asm\

--enable-shared \

--disable-static

**EXAMPLE**

http://dranger.com/ffmpeg/tutorial01.c

// tutorial01.c

// Code based on a tutorial by Martin Bohme (boehme@inb.uni-luebeckREMOVETHIS.de)

// Tested on Gentoo, CVS version 5/01/07 compiled with GCC 4.1.1

// With updates from https://github.com/chelyaev/ffmpeg-tutorial

// Updates tested on:

// LAVC 54.59.100, LAVF 54.29.104, LSWS 2.1.101

// on GCC 4.7.2 in Debian February 2015

//A sample program that shows how to use libavformat and libavcodec to

// read video from a file.

// Use gcc -o tutorial01 tutorial01.c -lavformat -lavcodec -lswscale -lz

// to build (assuming libavformat and libavcodec are correctly installed

// your system).

// Run using

// tutorial01 myvideofile.mpg

// to write the first five frames from "myvideofile.mpg" to disk in PPM

// format.

#include <libavcodec/avcodec.h>

#include <libavformat/avformat.h>

#include <libswscale/swscale.h>

#include <stdio.h>

// compatibility with newer API

#if LIBAVCODEC\_VERSION\_INT < AV\_VERSION\_INT(55,28,1)

#define av\_frame\_alloc avcodec\_alloc\_frame

#define av\_frame\_free avcodec\_free\_frame

#endif

void SaveFrame(AVFrame \*pFrame, int width, int height, int iFrame) {

FILE \*pFile;

char szFilename[32];

int y;

// Open file

sprintf(szFilename, "frame%d.ppm", iFrame);

pFile=fopen(szFilename, "wb");

if(pFile==NULL)

return;

// Write header

fprintf(pFile, "P6\n%d %d\n255\n", width, height);

// Write pixel data

for(y=0; y<height; y++)

fwrite(pFrame->data[0]+y\*pFrame->linesize[0], 1, width\*3, pFile);

// Close file

fclose(pFile);

}

int main(int argc, char \*argv[]) {

// Initalizing these to NULL prevents segfaults!

AVFormatContext \*pFormatCtx = NULL;

int i, videoStream;

AVCodecContext \*pCodecCtxOrig = NULL;

AVCodecContext \*pCodecCtx = NULL;

AVCodec \*pCodec = NULL;

AVFrame \*pFrame = NULL;

AVFrame \*pFrameRGB = NULL;

AVPacket packet;

int frameFinished;

int numBytes;

uint8\_t \*buffer = NULL;

struct SwsContext \*sws\_ctx = NULL;

if(argc < 2) {

printf("Please provide a movie file\n");

return -1;

}

// Register all formats and codecs

av\_register\_all();

// Open video file

if(avformat\_open\_input(&pFormatCtx, argv[1], NULL, NULL)!=0)

return -1; // Couldn't open file

// Retrieve stream information

if(avformat\_find\_stream\_info(pFormatCtx, NULL)<0)

return -1; // Couldn't find stream information

// Dump information about file onto standard error

av\_dump\_format(pFormatCtx, 0, argv[1], 0);

// Find the first video stream

videoStream=-1;

for(i=0; i<pFormatCtx->nb\_streams; i++)

if(pFormatCtx->streams[i]->codec->codec\_type==AVMEDIA\_TYPE\_VIDEO) {

videoStream=i;

break;

}

if(videoStream==-1)

return -1; // Didn't find a video stream

// Get a pointer to the codec context for the video stream

pCodecCtxOrig=pFormatCtx->streams[videoStream]->codec;

// Find the decoder for the video stream

pCodec=avcodec\_find\_decoder(pCodecCtxOrig->codec\_id);

if(pCodec==NULL) {

fprintf(stderr, "Unsupported codec!\n");

return -1; // Codec not found

}

// Copy context

pCodecCtx = avcodec\_alloc\_context3(pCodec);

if(avcodec\_copy\_context(pCodecCtx, pCodecCtxOrig) != 0) {

fprintf(stderr, "Couldn't copy codec context");

return -1; // Error copying codec context

}

// Open codec

if(avcodec\_open2(pCodecCtx, pCodec, NULL)<0)

return -1; // Could not open codec

// Allocate video frame

pFrame=av\_frame\_alloc();

// Allocate an AVFrame structure

pFrameRGB=av\_frame\_alloc();

if(pFrameRGB==NULL)

return -1;

// Determine required buffer size and allocate buffer

numBytes=avpicture\_get\_size(PIX\_FMT\_RGB24, pCodecCtx->width,

pCodecCtx->height);

buffer=(uint8\_t \*)av\_malloc(numBytes\*sizeof(uint8\_t));

// Assign appropriate parts of buffer to image planes in pFrameRGB

// Note that pFrameRGB is an AVFrame, but AVFrame is a superset

// of AVPicture

avpicture\_fill((AVPicture \*)pFrameRGB, buffer, PIX\_FMT\_RGB24,

pCodecCtx->width, pCodecCtx->height);

// initialize SWS context for software scaling

sws\_ctx = sws\_getContext(pCodecCtx->width,

pCodecCtx->height,

pCodecCtx->pix\_fmt,

pCodecCtx->width,

pCodecCtx->height,

PIX\_FMT\_RGB24,

SWS\_BILINEAR,

NULL,

NULL,

NULL

);

// Read frames and save first five frames to disk

i=0;

while(av\_read\_frame(pFormatCtx, &packet)>=0) {

// Is this a packet from the video stream?

if(packet.stream\_index==videoStream) {

// Decode video frame

avcodec\_decode\_video2(pCodecCtx, pFrame, &frameFinished, &packet);

// Did we get a video frame?

if(frameFinished) {

// Convert the image from its native format to RGB

sws\_scale(sws\_ctx, (uint8\_t const \* const \*)pFrame->data,

pFrame->linesize, 0, pCodecCtx->height,

pFrameRGB->data, pFrameRGB->linesize);

// Save the frame to disk

if(++i<=5)

SaveFrame(pFrameRGB, pCodecCtx->width, pCodecCtx->height,

i);

}

}

// Free the packet that was allocated by av\_read\_frame

av\_free\_packet(&packet);

}

// Free the RGB image

av\_free(buffer);

av\_frame\_free(&pFrameRGB);

// Free the YUV frame

av\_frame\_free(&pFrame);

// Close the codecs

avcodec\_close(pCodecCtx);

avcodec\_close(pCodecCtxOrig);

// Close the video file

avformat\_close\_input(&pFormatCtx);

return 0;

}