Introduction to SDL and using Third Party Libraries in your Programs

Table of Contents

Third Party Libraries

A **library** is essentially code that somebody has written, intended for reuse. They are generally precompiled into .dll or other library files (.a, .so, …), and your program can **link** to these libraries in order to use the functionality within them.

Often, libraries will also have homepages where you can find in-depth **documentation**.

Libraries can be for anything – from text to speech, to UI, to game development.

SDL

http://libsdl.org/index.php

SDL (Simple DirectMedia Layer) is a C library that will handle things like input, graphics, audio, threading, timers, and more for you. SDL has been popular in the past for creating cross-platform games, and many games for Linux use SDL.

The 2.0 version of SDL adds support for building for mobile, so not only can you build for Windows, OSX, and Linux, but also for Android and iOS.

Getting Started

For this tutorial, the assumed operating system is Windows, and the assumed IDE is Code::Blocks. Linking in Code::Blocks on Linux is similar – it's mostly about locating the library files (.dll, .a, or .so), and pointing the search directories to the appropriate folders.

1. Download the SDL library

Go to http://libsdl.org/download-2.0.php – scroll to the **Development Libraries**, and under Windows download the MinGW version (SDL2-devel-2.0.4-mingw.tar.gz (MinGW 32/64-bit) – Code::Blocks uses MinGW as its compiler.

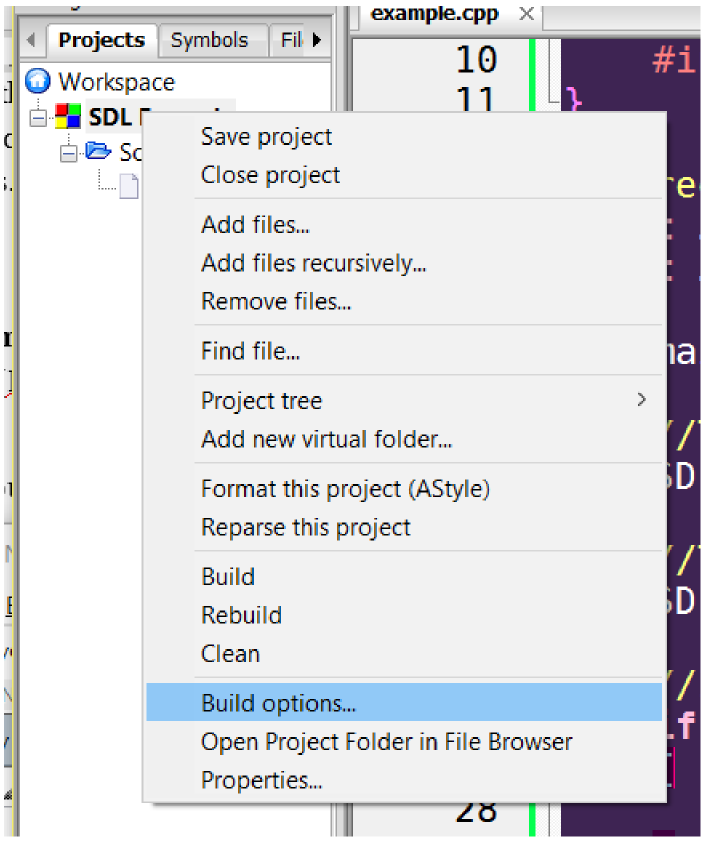
If you don't have a tool that can decompress .tar.gz files, you should check out 7-Zip: http://www.7-zip.org/

Decompress the library into a folder on your hard-drive. I've stored it at:

```
C:\Libraries\SDL2-devel-2.0.4-mingw
```

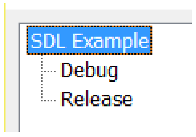
Within the folder, there should be directories such as **docs, include, lib,** and more. There should also be a folder named **i686-w64-mingw32** … This folder contains the libraries that we need for our program.

2. Create a new project



In Code::Blocks, create a new **Empty Project**, and add a source file to it. In the project pane, you will want to right-click on your project and select **Build Options…**

In the Project build options window, make sure to select your project's name, instead of Debug or Release.



This is so that your changes will affect both Debug and Release, instead of just one or the other.

3. Set up the Project Build Options

First, select the **Linker settings** tab. There are two sections: **Link libraries** and **Other linker options**.

1. In the **Link libraries** section, click the **Add** button. Navigate to the `i686-w64-mingw32\lib\` path. Select both **libSDL2main.a** and **libSDL2.a** libraries (you can ctrl+click to select multiple). Click **OK**.
2. In the **Other linker options** section, add the following lines:
   ```
   -lmingw32
   -lSDL2main
   -lSDL2
   ```
   Note that each of these begin with a dash, then a lower-case L (for library).

Next, go to the **Search directories** tab.

1. Select the **Compiler** sub-tab. Click on the **Add** button. Navigate to `i686-w64-mingw32\include\SDL2` and use this directory (you're selecting a directory, not a file).

2. Select the **Linker** sub-tab. Click on the **Add** button. Navigate to **`i686-w64-mingw32\lib`** and use this directory.

Click **OK** to leave the project build options.

4. Super basic SDL program

To test to see whether the libraries were successfully linked, write a short program that uses the SDL library and build. If the build is successful and the program opens (and then closes), we should be good to go.

If you're getting an error about SDL.h not being found, check the path in the Search directories part. Note that you have to have the **argc / args** parameters of main() otherwise you'll get a different build error.

Now we can start doing more with SDL.

```
#include <SDL.h>

extern "C"
{
    #include "SDL.h"
}

int main( int argc, char* args[] )
{
    SDL_Init( SDL_INIT_VIDEO );
    SDL_Quit();

   return 0;
}
```

SDL Super Basics

1. Creating a window

Before you can start drawing anything, you need to create a window. This will be different from the terminal that we have normally be writing our programs in – though you can still continue using **cout** to display information to that terminal.

For now, we are just working within **main()**, but eventually you should strive to work with clean code, and abstract a lot of this out into other functions or classes.

The SDL_Init function will return a negative value if there is an error, so you should check for a result.

```
int result = SDL_Init( SDL_INIT_VIDEO );
if ( result < 0 )
{
    cerr << "Error on SDL_Init!"
        << endl << SDL_GetError() << endl;
    return -1;
}
```

You will need an object called a **SDL_Window** that will be initialized to be

```
SDL_Window* window = NULL;
SDL_Surface* screenSurface = NULL;
```

the window, as well as an **SDL_Surface**, which will be a surface that we draw to.

We will create a Window object with the SDL_CreateWindow function.

http://wiki.libsdl.org/SDL_CreateWindow

```
SDL_Window* SDL_CreateWindow(
    const char* title,
    int         x,
    int         y,
    int         w,
    int         h,
    Uint32      flags)
```

You can use SDL_WINDOWPOS_UNDEFINED in place of the X, Y coordinates if you'd like.

If window is still NULL after the create function, then there was an error.

```
window = SDL_CreateWindow( "PROJECT NAME",
    SDL_WINDOWPOS_UNDEFINED, // x
    SDL_WINDOWPOS_UNDEFINED, // y
    640, 480, SDL_WINDOW_SHOWN );
```

```
if ( window == NULL )
{
    cerr << "Error on SDL_CreateWindow!"
        << endl << SDL_GetError() << endl;
    return -2;
}
```

Next, we will have to create the screen surface that we're drawing to.

```
screenSurface =
   SDL_GetWindowSurface( window );
```

We can draw some shapes to the screen as a test…

```
int SDL_FillRect(
    SDL_Surface*     dst,
    const SDL_Rect*  rect,
    Uint32           color)
```

```
// Background
SDL_Rect fullScreen = { 0, 0, 640, 480 };

SDL_FillRect( screenSurface,
    &fullScreen, // x,y/w,h
    SDL_MapRGB( screenSurface->format,
    0x7B, 0xBF, 0xFF ) ); // r,g,b

// Smaller box
SDL_Rect subRegion = { 20, 20, 150, 200 };

SDL_FillRect( screenSurface,
    &subRegion, // x,y/w,h
    SDL_MapRGB( screenSurface->format,
    0x5C, 0x09, 0x9C ) ); // r,g,b
```

Then we update the window.

```
SDL_UpdateWindowSurface( window );
```

We don't have a game loop set up yet, so let's pause the program for a few seconds (so we can see it) before it quits.

```
SDL_Delay( 2000 );
```

Finally, we must remember to use SDL_DestroyWindow at the end, then SDL_Quit.

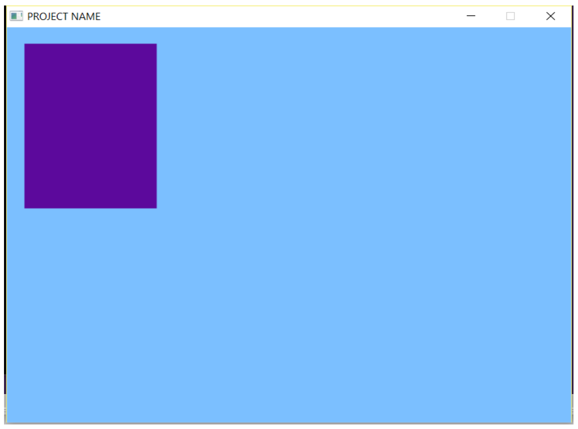
```
SDL_DestroyWindow( window );
SDL_Quit();
```

The result!

The result should be a window with some rectangles:

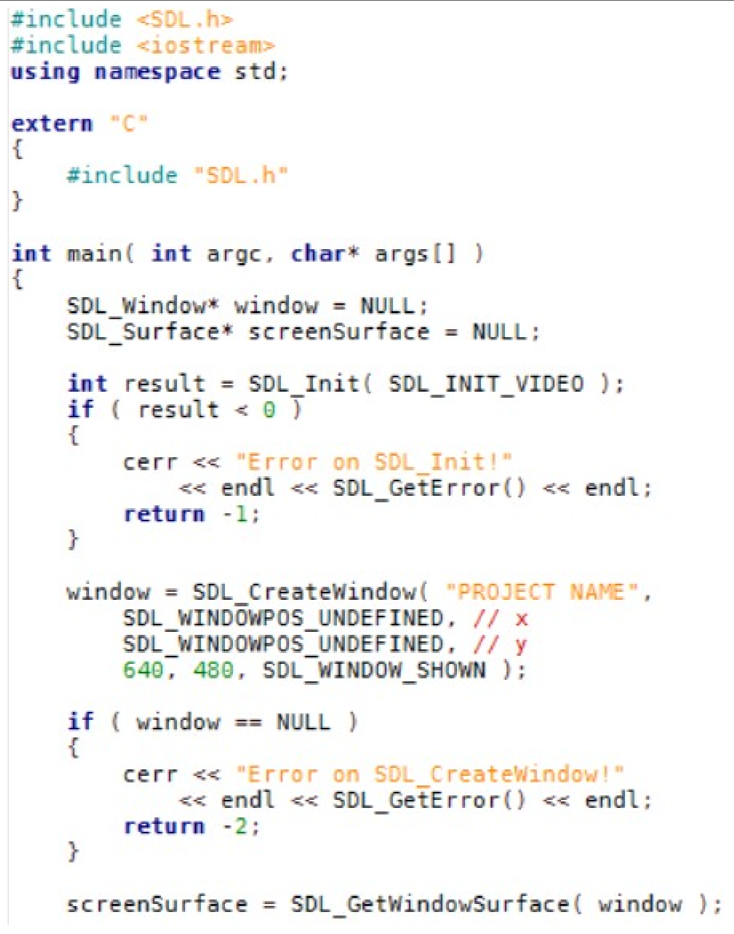
The first step towards graphical programming!

The full source is on the next page.



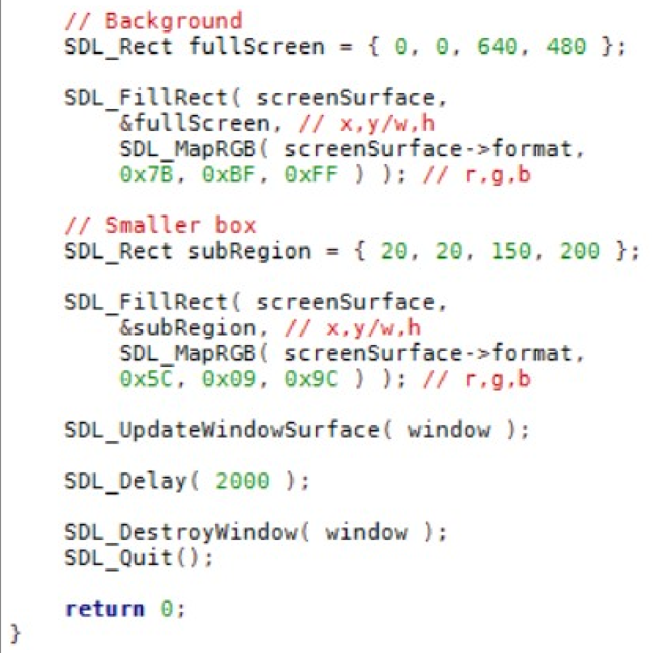
Full source code

Page 1



```
#include <SDL.h>
#include <iostream>
using namespace std;

extern "C"
{
    #include "SDL.h"
}

int main( int argc, char* args[] )
{
    SDL_Window* window = NULL;
    SDL_Surface* screenSurface = NULL;

    int result = SDL_Init( SDL_INIT_VIDEO );
    if ( result < 0 )
    {
        cerr << "Error on SDL_Init!"
            << endl << SDL_GetError() << endl;
        return -1;
    }

    window = SDL_CreateWindow( "PROJECT NAME",
        SDL_WINDOWPOS_UNDEFINED, // x
        SDL_WINDOWPOS_UNDEFINED, // y
        640, 480, SDL_WINDOW_SHOWN );

    if ( window == NULL )
    {
        cerr << "Error on SDL_CreateWindow!"
            << endl << SDL_GetError() << endl;
        return -2;
    }

    screenSurface = SDL_GetWindowSurface( window );
```

Page 2



```
    // Background
    SDL_Rect fullScreen = { 0, 0, 640, 480 };

    SDL_FillRect( screenSurface,
        &fullScreen, // x,y/w,h
        SDL_MapRGB( screenSurface->format,
        0x7B, 0xBF, 0xFF ) ); // r,g,b

    // Smaller box
    SDL_Rect subRegion = { 20, 20, 150, 200 };

    SDL_FillRect( screenSurface,
        &subRegion, // x,y/w,h
        SDL_MapRGB( screenSurface->format,
        0x5C, 0x09, 0x9C ) ); // r,g,b

    SDL_UpdateWindowSurface( window );

    SDL_Delay( 2000 );

    SDL_DestroyWindow( window );
    SDL_Quit();

    return 0;
}
```

2. Loading and drawing images

SDL_Surface objects are where images are stored. We can draw to a surface (like the window's surface in the above example), or we can load images to a surface.

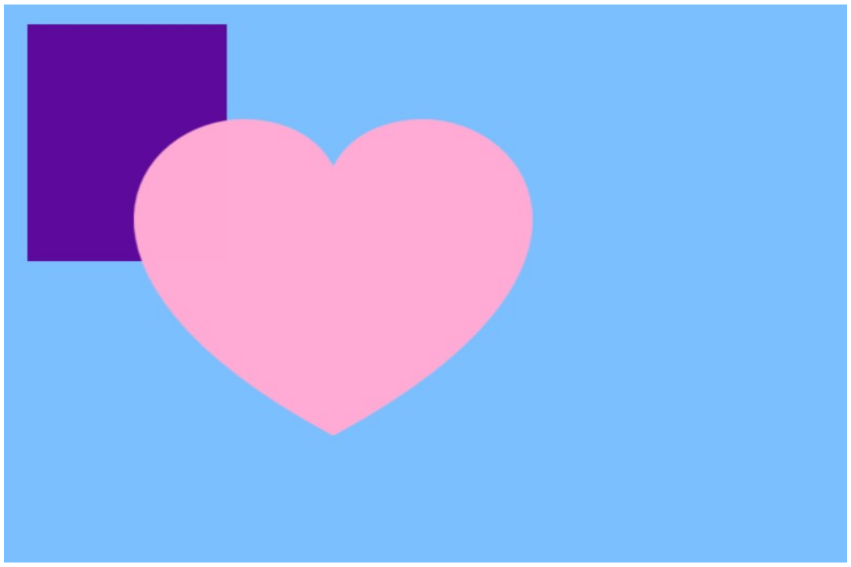
After we draw the rectangles to the screen and before we call SDL_UpdateWindow, let's create and load an image:

```
// Image
SDL_Surface* image = SDL_LoadBMP( "pink-heart-md.bmp" );
SDL_Rect bitmapSlice = { 0, 0, 300, 267 };
SDL_Rect imagePosition = { 100, 100, 150, 130 };
SDL_BlitSurface( image, &bitmapSlice, screenSurface, &imagePosition );
```

When we create a new SDL_Surface*, we will want to make sure to free it at the end of the program:

```
SDL_FreeSurface( image );
SDL_DestroyWindow( window );
SDL_Quit();
```

The result should be something like this:



This is only drawing bitmaps (.bmp), but we can include additional SDL libraries in order to load in other image types, like pngs.

Now what?

There are a lot of tutorials over SDL available at the Lazyfoo webpage:

http://lazyfoo.net/tutorials/SDL/index.php