Initialization and Setup

To initialize SDL you call **SDL_Init()** with a flag of what to initialize.

```
SDL_Init( SDL_INIT_VIDEO );
```

A window is created by calling the function **SDL_CreateWindow()** which returns a **SDL_Window***, this can then be assigned to a corresponding variable.

```
SDL_Window* window = SDL_CreateWindow( "Title",
                                SDL_WINDOWPOS_UNDEFINED,
                                SDL_WINDOWPOS_UNDEFINED,
                                SCREEN_WIDTH,
                                SCREEN_HEIGHT,
                                SDL_WINDOW_SHOWN );
```

A renderer is created by calling the function **SDL_CreateRenderer()** which returns a **SDL_Renderer***, this can then be assigned to a corresponding variable.

```
SDL_Renderer* renderer = SDL_CreateRenderer( window, -1,
                                        SDL_RENDERER_ACCELERATED);
```

Creating the window and renderer can be done together in a more concise way by using the **SDL_CreateWindowAndRenderer()** function which creates a window with a default renderer. A Window flag can be called in (the 3rd argument). To add a title to the application, you can use the **SDL_SetTitle()** function, which takes in a **SDL_Window*** and a string with the title you would like to have.

```
SDL_CreateWindowAndRenderer(640, 480, 0, &window, &renderer);
SDL_SetWindowTitle(window, “Title”);
```

To uninitialize SDL you call the **SDL_Quit()** function. You can then destroy the window and renderer with calls to **SDL_DestroyWindow()** and **SDL_DestroyRenderer()** with the **SDL_Window*** and **SDL_Renderer*** variables. You can then set those values to NULL.

```
SDL_DestroyWindow( window );
SDL_DestroyRenderer( renderer );
window = NULL;
renderer = NULL;
SDL_Quit();
```

Event Handling

SDL handles events by looping through the **SDL_PollEvent()** function which takes in the address of an **SDL_Event** structure variable.

```
SDL_Event e;
while( SDL_PollEvent(&e) != 0 ) {
```

```
        if( e.type == SDL_QUIT ) {
            running = FALSE;
        }
    }
```

An **SDL_Event** variable has multiple data fields, which themselves have components. Some of these components are: **SDL_QUIT**, **SDL_MOUSEMOTION**, **SDL_MOUSEBUTTONUP**, and **SDL_KEYDOWN** and **SDL_KEYUP**. These are types of data fields: **SDL_MouseMotionEvent**, **SDL_MouseButtonEvent** and **SDL_KeyBoardEvent**.

While looping through the events, you can test if an event of a specific type has occurred in the window. After that you can call on the event through its data field type and that component of that event structure.

```
SDL_Event event;
while( SDL_PollEvent(&event) != 0 ) {
    if( e.type == SDL_MOUSEMOTION ) {
        printf("Mouse position: ("%d,&d)\n",event.motion.x,
                                            event.motion.y);
    }
    if( e.type == SDL_KEYDOWN ) {
        SDL_Keycode code = event.key.keysym.sym;
        if(code == SDLK_ESCAPE ) {
            running = false;
        }
    }
}
```

You can also get information about the mouse with a call to **SDL_GetMouseState()** which returns, as output parameters, the x and y position of the mouse. This basically gives a *snapshot* of the current state of the mouse when the function is called.

```
int mouseX, mouseY;
SDL_GetMouseState(&mouseX, &mouseY);
```

Information about the keyboard can also be obtained using the **SDL_GetKeyboardState()** which returns a pointer to an array of key states. Indexes intro this array are obtained using SDL_Scancode values. This also gives a *snapshot* of the keyboard state when the function was called.

```
const Uint8* state = SDL_GetKeyboardState(NULL);
if (state[ SDL_SCANCODE_LEFT ]){
}
```

Rendering

Basics

To set the color before rendering anything to the screen you must call the **SDL_SetRenderDrawColor()** function, which takes as it's input the **SDL_Renderer*** and the rgba values you would like to set the renderer to.

```
SDL_SetRenderDrawColor(renderer, 255, 255, 255, 255);
```

The screen can be cleared using the **SDL_RenderClear()** function which takes a **SDL_Renderer*** as its input. This will fill the entire screen with whatever the draw color is.

```
SDL_RenderClear(renderer);
```

To finally update the screen and swap the buffers onto the window, call the **SDL_RenderPresent()** function which takes in a **SDL_Renderer*** as its input.

```
SDL_RenderPresent(renderer);
```

To display a texture onto a portion of the screen using hardware accelerated rending, you must call the **SDL_RenderCopy()** function, which takes in a **SDL_Renderer*** variable, a **SDL_Texture*** variable, a **SDL_Rect*** for the source, and a **SDL_Rect*** for the destination.

```
SDL_RenderCopy(renderer, texture, &srcRect, &dstRect);
```

Primitive Shapes

To draw a rectangle that is filled with the rendering color: call the function **SDL_RenderFillRect()** which takes in a **SDL_Renderer*** variable and a **SDL_Rect*** variable as its arguments.

```
SDL_Renderer* renderer;
SDL_Rect rect = {0, 0, 40, 40};
SDL_RenderFillRect(renderer, &rect);
```

To draw an empty rectangle (just the borders), use the **SDL_RenderDrawRect()** function, which takes in an **SDL_Renderer*** variable and a **SDL_Rect*** variable.

```
SDL_Renderer* renderer;
SDL_Rect rect = {0, 0, 40, 40};
SDL_RenderDrawRect(renderer, &rect);
```

To draw a line use the **SDL_RenderDrawLine()** which takes a **SDL_Render*** variable along with the x1, y1, x2, and y2 of the line.

```
SDL_RenderDrawLine(renderer, x1, y1, x2, y2);
```

Color Keying

To modify an imported image so that a color will become transparent in that image, use the **SDL_SetColorKey()** function, which takes as its arguments: a **SDL_Surface*** variable, a flag (**SDL_TRUE** to enable color key) and a Uint32 key which is the pixel format. The pixel format

of the image can be found by using the **SDL_MapRGB()** function which takes in a **SDL_PixelFormat** and the rgb values that are to be made transparent.

```
SDL_Surface* surface = SDL_LoadBMP("/location/of/image.bmp");
SDL_SetColorKey(surface, SDL_TRUE, SDL_MapRGB(surface->format(0,
                                        0, 0));
```

Image Loading

To load an image, you must call either **BMP_Load()** or **IMG_Load()** (from the SDL_Image extension library), which both take in a string that represents the *file path* of the image file to be loaded and return an **SDL_Surface***.

```
SDL_Surface* surface;
loadedSurface = IMG_Load("the/file/path");
```

To convert this surface into a texture, you must call the **SDL_CreateTextureFromSurface()** function, which wakes in an **SDL_Renderer*** and a **SDL_Surface*** and returns an **SDL_Texture***.

```
SDL_Texture* texture;
texture = SDL_CreateTextureFromSurface(renderer, surface);
SDL_FreeSurface(surface);
```

DO NOT FORGET: to FREE THE SURFACE after the texture is created with **SDL_FreeSurface(),** and to DESTROY THE TEXTURE at the end of the program with **SDL_DestroyTexture()**.

Font Loading

To load a font you must call the **TTF_OpenFont()** function (from the SDL_ttf extension library) which takes in a char*(string) string that represent the file path of the ttf file you wish to load and the point (size) of the font. This returns an **TTF_Font***.

```
TTF_Font* font;
font = TTF_OpenFont("~/Library/Fonts/Arial.ttf", 24);
```

You then use the font to create a surface that can be rendered onto the screen. This is done with the **TTF_RenderText_Solid()** function , which takes in an **TTF_Font*** variable, a char* variable, and an **SDL_Color** variable.

```
SDL_Surface* surface;
SDL_Color black = { 0, 0, 0 };
surface = TTF_RenderText_Solid(font, "Hello World!", black);
```

To convert this surface into a texture use the **SDL_CreateTextureFromSurface()** function which takes as its parameters an **SDL_Renderer*** variable and a **SDL_Surface*** variable.

```
SDL_Texture *message;
message = SDL_CreateTextureFromSurface(renderer, surface);
SDL_FreeSurface(surface);
```

DO NOT FORGET: to free the surface and texture using the functions: **`SDL_FreeSurface()`** function and the **`SDL_DestroyTexture()`**.

Timers

The application can be paused using the **`SDL_Delay()`** function, which takes in an integer that represents the milliseconds the program should pause on.

```
SDL_Delay(1000);
```

Special Cases

In trying to use **`SDL_Textures*`** as output parameters, I came across a problem with the code. Simply putting **`SDL_Texture*`** texture as an output parameter does not seem to change the value of the parameter:

```
void loadTexture(SDL_Texture *texture) {
      texture = SDL_CreateTextureFromSurface(renderer, surface);
}
```

Instead, in order to really change the value of the texture most create a pointer to a pointer of the **`SDL_Texture*`**: **`SDL_Texture**`** texture, which would require the address of the SDL_Texture* to be put into the calling function.

```
void loadTexture(SDL_Texture** texture) {
      *texture = SDL_CreateTextureFromSurface(renderer, surface);
}
```

More research is needed to understand why this happens, and how it works. At first it seems intuitive that **`SDL_Texture*`** variables must be manipulated through its address like with most functions. Look into pointers and dynamic memory management. Also look at SDL_CreateWindowAndRenderer, is is does a similar thing with **`SDL_Window**`** and **`SDL_Renderer**`**.