Lesson 03 – Events

The Event Loop

Most multimedia programs rely on an events system to process input. SDL provides a flexible API for processing input events. Essentially, SDL records input from devices (like the keyboard, mouse, or controller) as events, storing them in the "event queue." You can think of this structure just like a waiting line—events are queued in the back of the line and taken from the front of the line.

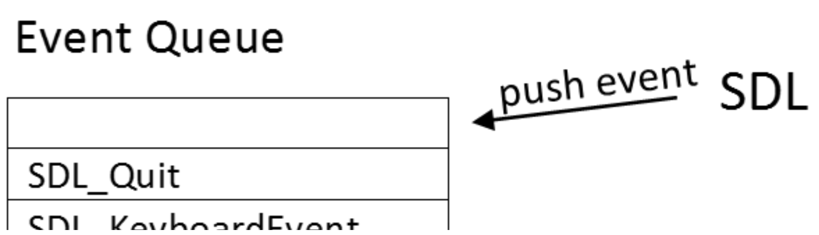
Example Program

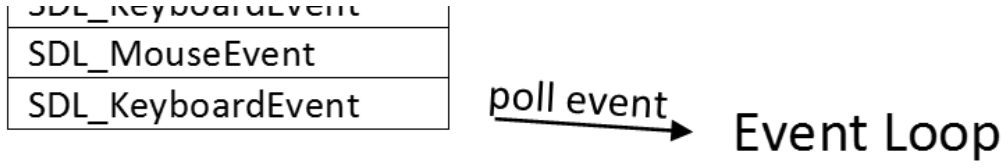
Download

```
#include <iostream>
#include <SDL.h>

using namespace std;

bool init();
void kill();
bool load();
bool loop();

// Pointers to our window and
surfaces
SDL_Window* window;
SDL_Surface* winSurface;
SDL_Surface* image1;
SDL_Surface* image2;

int main(int argc, char** args) {

	if ( !init() ) return 1;

	if ( !load() ) return 1;

	while ( loop() ) {
		// wait before processing
the next frame
		SDL_Delay(10);
	}
```

```
    kill();
    return 0;
}

bool loop() {

    static bool renderImage2;
    SDL_Event e;

    // Blit image to entire window
    SDL_BlitSurface( image1,
NULL, winSurface, NULL );

    while( SDL_PollEvent( &e ) != 0
) {
        switch (e.type) {
            case SDL_QUIT:
                return false;
            case
SDL_KEYDOWN:
                renderImage2 =
true;
                break;
            case SDL_KEYUP:
                renderImage2 =
false;
                // can also test
individual keys, modifier flags, etc,
etc.
                break;
            case
SDL_MOUSEMOTION:
                // etc.
                break;
        }
    }

    if (renderImage2) {
        // Blit image to scaled
portion of window
        SDL_Rect dest;
        dest.x = 160;
        dest.y = 120;
        dest.w = 320;
        dest.h = 240;
        SDL_BlitScaled(image2,
NULL, winSurface, &dest);
    }

    // Update window
    SDL_UpdateWindowSurface(
        SDL_UpdateWindowSurface(
window );

        return true;
}

bool load() {
        // Temporary surfaces to load
images into
                // This should use only 1
temp surface, but for conciseness we
use two
        SDL_Surface *temp1, *temp2;

        // Load images
        temp1 =
SDL_LoadBMP("test1.bmp");
        temp2 =
SDL_LoadBMP("test2.bmp");

        // Make sure loads succeeded
        if ( !temp1 || !temp2 ) {
                cout << "Error loading
image: " << SDL_GetError() << endl;
                system("pause");
                return false;
        }

        // Format surfaces
        image1 = SDL_ConvertSurface(
temp1, winSurface->format, 0 );
        image2 = SDL_ConvertSurface(
temp2, winSurface->format, 0 );

        // Free temporary surfaces
        SDL_FreeSurface( temp1 );
        SDL_FreeSurface( temp2 );

        // Make sure format succeeded
        if ( !image1 || !image2 ) {
                cout << "Error converting
surface: " << SDL_GetError() <<
endl;
                system("pause");
                return false;
        }
        return true;
}

bool init() {
        // See last example for
```

```
comments
    if ( SDL_Init(
SDL_INIT_EVERYTHING ) < 0 ) {
        cout << "Error initializing
SDL: " << SDL_GetError() << endl;
        system("pause");
        return false;
    }

    window = SDL_CreateWindow(
"Example",
SDL_WINDOWPOS_UNDEFINED,
SDL_WINDOWPOS_UNDEFINED,
640, 480, SDL_WINDOW_SHOWN );
    if ( !window ) {
        cout << "Error creating
window: " << SDL_GetError()  <<
endl;
        system("pause");
        return false;
    }

    winSurface =
SDL_GetWindowSurface( window );
    if ( !winSurface ) {
        cout << "Error getting
surface: " << SDL_GetError() <<
endl;
        system("pause");
        return false;
    }
    return true;
}

void kill() {
    // Free images
    SDL_FreeSurface( image1 );
    SDL_FreeSurface( image2 );

    // Quit
    SDL_DestroyWindow( window );
    SDL_Quit();
}
```





In your programs, you will always have an event (or "game," or "main") loop that processes these events and runs your program based on the input. Each time the event loop is run, you must pull each event off the event queue (in order) to process the input. This is done with the function SDL_PollEvent(). This function removes the first event from the queue, copying the value into a parameter of type SDL_Event. If the event queue was empty, the function will return 0.

Once you have polled your event, you can use it in a logic chain to deduce what the input was and how to respond.

```
SDL_Event ev;
bool running = true;

// Main loop
while ( running ) {
        // Event loop
        while ( SDL_PolLEvent( &ev ) != 0 ) {
                // Test members of ev
        }

        // Wait before next frame
        SDL_Delay(100);
}
```

SDL_Event

SDL_Event contains one of any sub-event. This is possible through the use of a union. A union describes several mutually exclusive data members within a structure. This means the sub-event-types are all stored in the same memory, so SDL_Event can be flexible without wasting space. However, this system makes syntax slightly clunkier—to access the sub-event-data, one must first access the sub-event within SDL_Event.

For example, accessing a SDL_KeyboardEvent...

```
SDL_Event evt;
SDL_PollEvent( &evt );

if (evt.type == SDL_KEYDOWN) {
        switch ( evt.key.sym.sym ) {   // Note evt.key accesses the real
data,
                                        // the SDL_KeyboardEvent
            // ...
        }
}
```

Quitting

Your event loop will receive an event of type SDL_QUIT when the user wishes to close the program. This includes pressing the 'x' on the window, pressing ALT+F4, or otherwise requesting the program end. This does not include ending the process or sending CTRL+C to the console—those are uncontrolled, immediate aborts.

Hence, when your program receives an SDL_QUIT event, it should gracefully shut down (or prompt the user for more information). The type of an event is accessible through its "type" member.

```
SDL_Event ev;
bool running = true;

// Main loop
while ( running ) {
	// Event loop
	while ( SDL_PollEvent( &ev ) != 0 ) {
		// check event type
		switch (ev.type) {
			case SDL_QUIT:
				// shut down
				running = false;
				break;
		}
	}

	// Wait before next frame
	SDL_Delay(100);
}
```

Keyboard Events

Keyboard events come in two flavors—SDL_KEYDOWN and SDL_KEYUP. Both of these types are associated SDL_KeyboardEvent, which includes a keycode and flags representing the input event.

Whether the key is pressed/released/repeated can be determined through the

Whether the key is pressed/released/repeated can be determined through the state and repeat members of SDL_KeyboardEvent, whereas the keycode and modifier keys are specified in the keysym member (SDL_Keysym).

```
SDL_Event ev;
bool running = true;

// Main loop
while ( running ) {
    // Event loop
    while ( SDL_PollEvent( &ev ) != 0 ) {
        // check event type
        switch (ev.type) {
            case SDL_QUIT:
                // shut down
                running = false;
                break;
            case SDL_KEYDOWN:
                // test keycode
                switch ( ev.key.keysym.sym ) {
                    case SDLK_w:
                        break;
                    case SDLK_s:
                        break;
                    // etc
                }
                break;
        }
    }

    // Wait before next frame
    SDL_Delay(100);
}
```

Check out SDL_KeyboardEvent for more detail.

Keyboard Polling

There is a way to get keyboard input without the event system. This is by polling the keyboard directly. This is not recommended, as polling the keyboard gives you the state at that moment in time, not a log of each event that happened since the last poll. However, polling can still occasionally be useful, so SDL provides the function SDL_GetKeyboardState(). This function returns an array of values containing the key values. This array is persistent—and it will be updated as keyboard events are processed.

To access data within the key array, you can use a SDL_Scancode. Scancodes are like the key values from SDL_KeyboardEvent, but instead serve as indexes into like the key values from SDL_KeyboardEvent, but instead serve as indexes into the keyboard state array.

```
char* keys = SDL_GetKeyboardState(NULL);

// Test W key
if ( keys[SDL_SCANCODE_W] ) {
	// ...
}
```

Mouse Events

All types of events are similar to keyboard events in that they have the event type included with several data members describing the input. Mouse events can be of types SDL_MOUSEMOTION, SDL_MOUSEBUTTONDOWN, SDL_MOUSEBUTTONUP, and SDL_MOUSEWHEEL.

These types are associated with SDL_MouseButtonEvent, SDL_MouseMotionEvent, and SDL_MouseWheelEvent. All of these types include the x and y coordinates of the mouse event, as well as extra data and modifiers.

```
SDL_Event ev;
bool running = true;

// Main loop
while ( running ) {
	// Event loop
	while ( SDL_PollEvent( &ev ) != 0 ) {
		// check event type
		switch (ev.type) {
			case SDL_QUIT:
				// shut down
				running = false;
				break;
			case SDL_KEYDOWN:
				// test keycode
				switch ( ev.key.keysym.sym ) {
					case SDLK_w:
						break;
					case SDLK_s:
						break;
					// etc
				}
				break;
			case SDL_MOUSEBUTTONUP:
				// test button
				switch ( ev.button.button ) {
```

```
                    case SDL_BUTTON_LEFT:
                        break;
                    case SDL_BUTTON_RIGHT:
                        break;
                    case SDL_BUTTON_X1:
                        break;
                    // etc
                }
        }
    }

    // Wait before next frame
    SDL_Delay(100);
}
```

Other Events

There are many other types of events that these notes won't cover. I highly recommend taking a look through the SDL documentation website to learn how to use these other types.

Event Types:

- SDL_TextEditingEvent
- SDL_TextInputEvent
- SDL_MouseMotionEvent
- SDL_MouseButtonEvent
- SDL_MouseWheelEvent
- SDL_JoyAxisEvent
- SDL_JoyBallEvent

- SDL_JoyHatEvent
- SDL_JoyButtonEvent
- SDL_JoyDeviceEvent
- SDL_ControllerAxisEvent
- SDL_ControllerButtonEvent
- SDL_ControllerDeviceEvent
- SDL_AudioDeviceEvent

- SDL_QuitEvent

- SDL_UserEvent
- SDL_SysWMEvent
- SDL_TouchFingerEvent
- SDL_MultiGestureEvent
- SDL_DollarGestureEvent
- SDL_DropEvent

User Events

One more short section—user defined events. SDL provides the structure SDL_UserEvent for this purpose; it has arbitrary data members for the user to specify. This structure is used in conjunction with SDL_RegisterEvents() and SDL_PushEvent().

SDL_RegisterEvents() is used to allocate a range of values for your user-defined event types. These numbers are used as the value of the "type" member of the generalized SDL_Event structure. SDL_PushEvent() allows you to add an event to the queue. This can include your user defined events.

```
int userType = SDL_RegisterEvents(1);

if (userType == ((uint32_t) -1)) {
	// failure
}

SDL_Event ev;
ev.type = userType;

ev.user.code = someEvtCode;
ev.user.data1 = &someData;
ev.user.data2 = 0;

SDL_PushEvent(&ev);
```