Event Handling

This tutorial is a detailed list of window events. It describes them, and shows how to (and how not to) use them.

The sf::Event

sf::Event holds all the informations about a system event that just happened. Events are retrieved using the sf::Window::pollEvent and sf::Window::waitEvent functions. A sf::Event instance contains the type of the event (mouse moved, key pressed, window closed, ...) as well as the details about this particular event. Please note that the event parameters are defined in a union, which means that only the member matching the type of the event will be properly filled; all other members will have undefined values and must not be read if the type of the event doesn't match. For example, if you received a KeyPressed event, then you must read the event.key member, all other members such as event.mouseMove or event.text will have undefined values.

```
sf::Event event;
while (window.pollEvent(event))
{
    // Request for closing the window
    if (event.type == sf::Event::Closed)
        window.close();
    // The escape key was pressed
    if ((event.type == sf::Event::KeyPressed) && (event.key.code == sf::Keyboard::Escape))
        window.close();
    // The window was resized
    if (event.type == sf::Event::Resized)
        doSomethingWithTheNewSize(event.size.width, event.size.height);
    // etc ...
}
```

The sf::Event type

Before dealing with events, it is important to understand what the `sf::Event` type is, and how to correctly use it. `sf::Event` is a **union,** which means that only one of its members is valid at a time (remember your C++ lesson: all the members of a union share the same memory space). The valid member is the one that matches the event type, for example `event.key` for a `KeyPressed` event. Trying to read any other member will result in an undefined behavior (most likely: random or invalid values). It it important to never try to use an event member that doesn't match its type.

`sf::Event` instances are filled by the `pollEvent` (or `waitEvent`) function of the `sf::Window` class. Only these two functions can produce valid events, any attempt to use an `sf::Event` which was not returned by successful call to `pollEvent` (or `waitEvent`) will result in the same undefined behavior that was mentioned above.

To be clear, here is what a typical event loop looks like :

```
  sf::Event event;

// while there are pending events...
while (window.pollEvent(event))
{
    // check the type of the event...
```

```
    switch (event.type)
    {
        // window closed
        case sf::Event::Closed:
            window.close();
            break;

        // key pressed
        case sf::Event::KeyPressed:
            ...
            break;

        // we don't process other types of events
        default:
            break;
    }
}
```

The Closed Event

The sf::Event::Closed event is triggered when the user wants to close the window, through any possible methods the window manager provides ”close button, keyboard shortcut, etc.

This event only represents a close request, the window is not yet closed when the event is received.

Typical code will just call `window.close()` in reaction to this event, to actually close the window. However, you may also want to do something else first, like saving the current application state or asking the user what to do. If you don't do anything, the window remains open.

There's no member associated with this event in the `sf::Event` union.

```
if (event.type == sf::Event::Closed)
    window.close();
```

The Resized Event

The sf::Event::Resized event is triggered when the window is resized, either through user action or programmatically by calling window.setsize( ).

You can use this event to adjust the rendering settings: the viewport if you use OpenGL directly, or the current view if you use sfml-graphics.

The member associated with this event is `event.size`, it contains the new size of the window.

```
if (event.type == sf::Event::Resized)
{
    std::cout << "new width: " << event.size.width << std::endl;
    std::cout << "new height: " << event.size.height << std::endl;
}
```

The LostFocus and GainedFocus Events

The `sf::Event::LostFocus` and `sf::Event::GainedFocus` events are triggered when the window loses/gains focus, which happens when the user switches the currently active window. When the window is out of focus, it

doesn't receive keyboard events.

This event can be used e.g. if you want to pause your game when the window is inactive. There's no member associated with these events in the `sf::Event` union.

```
if (event.type == sf::Event::LostFocus)
    myGame.pause();

if (event.type == sf::Event::GainedFocus)
    myGame.resume();
```

The TextEntered Event

A single key press/release can be detected and handled In a relative and straightforward manner, the tricky part comes when we use two keys at the same time, that time handling the event like the previous events Is not suitable for example- using (shift + 1) for exclamation mark ! won’t work by using previous technique.

SFML provides a easiest way to solve this problem by providing TextEntered Event. This event only fired when a combination of keys representig a character are pressed, meaning that a single key (only shift for example) might not trigger the event. But if a key which represent a character may fired the event for example pressing/releasing K or T etc.

```
#include <SFML/Window.hpp>
using namespace sf;

int main()
{

    String text;
    Window window(VideoMode(700, 400), "window", Style::Default);

    while (window.isOpen())
    {

        Event event;
        while (window.pollEvent(event))
        {
            switch (event.type)
            {
            case Event::EventType::Closed:
                window.close();
                break;

            // Trigger the TextEntered Event by:
            // pressing one key at a time like- pressing K or T etc...
            // pressing two keys at a time like- pressing shift + 1 for exclamation mark etc...
            case Event::EventType::TextEntered:
            // change the unicode value to a printiable character and store into a text(variable) of sf::String type
                text += event.text.unicode;
                break;

            case Event::EventType::KeyReleased:
                if (event.key.code == Keyboard::Key::Return) // press enter to change the window tittle
                {
                    window.setTitle(text); // set the window tittle
                    text.clear();
                }
                break;
```

```
            default:
                break;
            }
        }
    }
}
```

The member associated with this event is `event.text`, it contains the Unicode value of the entered character. You can either put it directly in a `sf::String`, or cast it to a `char` after making sure that it is in the ASCII range (0 - 127).

```
if (event.type == sf::Event::TextEntered)
{
    if (event.text.unicode < 128)
        std::cout << "ASCII character typed: " << static_cast<char>(event.text.unicode) << std::endl;
}
```

Sf : : string

The sf : : strings class Is used to handle the conversions between strings types and encodings. As such we do not have to worry about the language or symbols on a keyboard layout It can store any character from any language.

Note

Many programmers use the `KeyPressed` event to get user input, and start to implement crazy algorithms that try to interpret all the possible key combinations to produce correct characters. Don't do that!

The KeyPressed and KeyReleased Events

The `sf::Event::KeyPressed` and `sf::Event::KeyReleased` events are triggered when a keyboard key is pressed/released.

If a key is held, multiple `KeyPressed` events will be generated, at the default operating system delay (ie. the same delay that applies when you hold a letter in a text editor). To disable repeated `KeyPressed` events, you can call `window.setKeyRepeatEnabled(false)`. On the flip side, it is obvious that `KeyReleased` events can never be repeated. This event is the one to use if you want to trigger an action exactly once when a key is pressed or released, like making a character jump with space, or exiting something with escape.

```
if (event.type == sf::Event::KeyPressed)
{
    if (event.key.code == sf::Keyboard::Escape)
    {
        std::cout << "the escape key was pressed" << std::endl;
        std::cout << "control:" << event.key.control << std::endl;
        std::cout << "alt:" << event.key.alt << std::endl;
        std::cout << "shift:" << event.key.shift << std::endl;
        std::cout << "system:" << event.key.system << std::endl;
    }
}
```

Note

Note that some keys have a special meaning for the operating system, and will lead to unexpected behavior. An example is the F10 key on Windows, which "steals" the focus, or the F12 key which starts the debugger when using Visual Studio. This will probably be solved in a future version of SFML.

The MouseWheelScrolled Event

The `sf::Event::MouseWheelScrolled` event is triggered when a mouse wheel moves up or down

```
if (event.type == sf::Event::MouseWheelScrolled)
{
    if (event.mouseWheelScroll.wheel == sf::Mouse::VerticalWheel)
        std::cout << "wheel type: vertical" << std::endl;
    else if (event.mouseWheelScroll.wheel == sf::Mouse::HorizontalWheel)
        std::cout << "wheel type: horizontal" << std::endl;
    else
        std::cout << "wheel type: unknown" << std::endl;
    std::cout << "wheel movement: " << event.mouseWheelScroll.delta << std::endl; // delta- speed of wheel
    std::cout << "mouse x: " << event.mouseWheelScroll.x << std::endl;
    std::cout << "mouse y: " << event.mouseWheelScroll.y << std::endl;
}
```

The MouseButtonPressed and MouseButtonReleased Events

The `sf::Event::MouseButtonPressed` and `sf::Event::MouseButtonReleased` events are triggered when a mouse button is pressed/released.

SFML supports 5 mouse buttons: left, right, middle (wheel), extra #1 and extra #2 (side buttons).

The member associated with these events is `event.mouseButton`, it contains the code of the pressed/released button, as well as the current position of the mouse cursor.

```
if (event.type == sf::Event::MouseButtonPressed)
{
    if (event.mouseButton.button == sf::Mouse::Right)
    {
        std::cout << "the right button was pressed" << std::endl;
        std::cout << "mouse x: " << event.mouseButton.x << std::endl;
        std::cout << "mouse y: " << event.mouseButton.y << std::endl;
    }
}
```

The MouseMoved Event

The `sf::Event::MouseMoved` event is triggered when the mouse moves within the window.

This event is triggered even if the window isn't focused. However, it is triggered only when the mouse moves within the inner area of the window, not when it moves over the title bar or borders.

The member associated with this event is `event.mouseMove`, it contains the current position of the mouse cursor relative to the window.

```
if (event.type == sf::Event::MouseMoved)
{
    std::cout << "new mouse x: " << event.mouseMove.x << std::endl;
```

```
    std::cout << "new mouse y: " << event.mouseMove.y << std::endl;
}
```

The MouseEntered and MouseLeft event

The `sf::Event::MouseEntered` and `sf::Event::MouseLeft` events are triggered when the mouse cursor enters/leaves the window.

```
if (event.type == sf::Event::MouseEntered)
    std::cout << "the mouse cursor has entered the window" << std::endl;

if (event.type == sf::Event::MouseLeft)
    std::cout << "the mouse cursor has left the window" << std::endl;
```

List of events :)

Window related events

Enum value	Member associated	Description
`Event::Closed`	None	This event is triggered when the OS detects that the user wants to close a window—the close button, key combination, and so on.
`Event::Resized`	`Event::size` holds the new size of the window	This event is triggered when the OS detects that the window has been resized manually, or when `Window::setSize()` has been used.
`Event::LostFocus` `Event::GainedFocus`	None	This event is triggered when the window loses or gains focus. Windows which are out of focus don't receive keyboard events.

Keyboard related events

Enum value	Member associated	Description
`Event::KeyPressed` `Event::KeyReleased`	`Event::key` holds the pressed/released key	This event is triggered when a single button is pressed or released on a focused window.
`Event::TextEntered`	`Event::text` holds the UTF-32 unicode value of the character	This event is triggered every time a character is typed. This produces a printable character from the user input, and is useful for text fields.

Mouse related events

Enum value	Member associated	Description
`Event::MouseMoved`	`Event::mouseMove` holds the new mouse position	This event is triggered when the mouse changes its position inside the window.
`Event::MouseButtonPressed` `Event::MouseButtonReleased`	`Event::mouseButton` holds the pressed/ released button and the mouse position	This event is triggered when a mouse button is pressed inside a window. Five buttons are currently supported—left, right, middle, extra button 1, and extra button 2.
`Event::MouseWheelMoved`	`Event::mouseWheel` holds the delta ticks of the wheel and the mouse position	This event is triggered when the scroll wheel moves inside a window.