WIKIPEDIA

Event (computing)

In computing, an **event** is an action or occurrence recognized by software, often originating asynchronously from the external environment, that may be handled by the software. Because an event is an entity which encapsulates the action and the contextual variables triggering the action we can use the acrostic mnemonic of an event as an "**E**xecution **V**ariable **E**ncapsulating **N**amed **T**rigger" to clarify the concept. Computer events can be generated or triggered by the system, by the user or in other ways. Typically, events are handled synchronously with the program flow, that is, the software may have one or more dedicated places where events are handled, frequently an event loop. A source of events includes the user, who may interact with the software by way of, for example, keystrokes on the keyboard. Another source is a hardware device such as a timer. Software can also trigger its own set of events into the event loop, e.g. to communicate the completion of a task. Software that changes its behavior in response to events is said to be event-driven, often with the goal of being interactive.

Contents

Description

Delegate event model

Event handler

Event notification

User generated events

- Mouse events
- Keyboard events
- Joystick events
- Touchscreen events
- Device events

See also

References

External links

Description

Event driven systems are typically used when there is some asynchronous external activity that needs to be handled by a program; for example, a user who presses a button on his mouse. An event driven system typically runs an event loop, that keeps waiting for such activities, e.g. input from devices or internal alarms. When one of these occurs, it collects data about the event and dispatches the event to the *event handler* software that will deal with it.

A program can choose to ignore events, and there may be libraries to dispatch an event to multiple handlers that may be programmed to *listen* for a particular event. The data associated with an event at a minimum specifies what type of event it is, but may include other information such as when it occurred, who or what caused it to occur, and extra data provided by the event *source* to the handler about how the event should be processed.

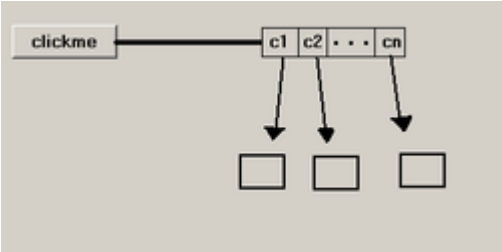
Events are typically used in user interfaces, where actions in the outside world (mouse clicks, window-resizing, keyboard presses, messages from other programs, etc.) are handled by the program as a series of events. Programs written for many windowing environments consist predominantly of event handlers.

Events can also be used at instruction set level, where they complement interrupts. Compared to interrupts, events are normally handled synchronously: the program explicitly waits for an event to be serviced (typically by calling an instruction that dispatches the next event), whereas an interrupt can demand service at any time.

Delegate event model

A common variant in object-oriented programming is the delegate event model, which is provided by some graphic user interfaces. This model is based on three entities:

- a control, which is the event source
- listeners, also called event handlers, that receive the event notification from the source
- interfaces (in the broader meaning of the term) that describe the protocol by which the event is to be communicated.



Delegate event model. `clickme` is the event source –a button in this example–, and it contains a list of listeners.

Furthermore, the model requires that:

- every listener must implement the interface for the event it wants to listen to
- every listener must register with the source to declare its desire to listen to the event
- every time the source generates the event, it communicates it to the registered listeners, following the protocol of the interface.

C# uses events as special delegates that can only be fired by the class that declares it. This allows for better abstraction, for example:[1]

```
delegate void Notifier (string sender);

class Model {
    public event Notifier notifyViews;
    public void Change() { ... notifyViews("Model"); }
}

class View1 {
    public View1(Model m) {
        m.notifyViews += new Notifier(this.Update1);
    }

    void Update1(string sender) {
        Console.WriteLine(sender + " was changed during update");
    }
}

class View2 {
    public View2(Model m) {
        m.notifyViews += new Notifier(this.Update2);
    }

    void Update2(string sender) {
        Console.WriteLine(sender + " was changed");
    }
}

class Test {
    static void Main() {
        Model model = new Model();

        new View1(model);
        new View2(model);
        model.Change();
    }
}
```

Event handler

In computer programming, an *event handler* is a callback subroutine that handles inputs received in a program (called a *listener* in Java and JavaScript[2]). Each *event* is a piece of application-level information from the underlying framework, typically the GUI toolkit. GUI events include key presses, mouse movement, action selections, and timers expiring. On a lower level, events can represent availability of new data for reading a file or network stream. Event handlers are a central concept in event-driven programming.

The events are created by the framework based on interpreting lower-level inputs, which may be lower-level events themselves. For example, mouse movements and clicks are interpreted as menu selections. The events initially originate from actions on the operating system level, such as interrupts generated by hardware devices, software interrupt instructions, or state changes in polling. On this level, interrupt handlers and signal handlers correspond to event handlers.

Created events are first processed by an *event dispatcher* within the framework. It typically manages the associations between events and event handlers, and may queue event handlers or events for later processing. Event dispatchers may call event handlers directly, or wait for events to be dequeued with information about the handler to be executed.

Event notification

Event notification is a term used in conjunction with communications software for linking applications that generate small messages (the "events") to applications that monitor the associated conditions and may take actions triggered by events.

Event notification is an important feature in modern database systems (used to inform applications when conditions they are watching for have occurred), modern operating systems (used to inform applications when they should take some action, such as refreshing a window), and modern distributed systems, where the producer of an event might be on a different machine than the consumer, or consumers. Event notification platforms are normally designed so that the application producing events does not need to know which applications will consume them, or even how many applications will monitor the event stream.

It is sometimes used as a synonym for publish-subscribe, a term that relates to one class of products supporting event notification in networked settings. The virtual synchrony model is sometimes used to endow event notification systems, and publish-subscribe systems, with stronger fault-tolerance and consistency guarantees.

User generated events

There are a large number of situations or events that a program or system may generate or respond to. Some common user generated events include:

Mouse events

A pointing device can generate a number of software recognisable pointing device gestures. A mouse can generate a number of mouse events, such as mouse move (including direction of move and distance), mouse left/right button up/down[3] and mouse wheel motion, or a comtbination of these gestures.

Keyboard events

Pressing a key on a keyboard or a combination of keys generates a keyboard event, enabling the program currently running to respond to the introduced data such as which key/s the user pressed.[3]

Joystick events

Moving a joystick generates an X-Y analogue signal. They often have multiple buttons to trigger events. Some gamepads for popular game boxes use joysticks.

Touchscreen events

The events generated using a touchscreen are commonly referred to as touch events or gestures.

Device events

Device events include action by or to a device, such as a shake, tilt, rotation, move etc.

See also

- Callback (computer programming)
- Database trigger
- DOM events
- Event-driven programming
- Exception handling
- Interrupt handler
- Interrupts
- Observer pattern (e.g., Event listener)
- Reactor pattern vs. Proactor pattern
- Signal programming
- Virtual synchrony

References

1. Mössenböck, Hanspeter (2002-03-25). "Advanced C#: Variable Number of Parameters" (http://ssw.jku.at/Teaching/Lectures/CSharp/Tutorial/Part2.pdf) (PDF). http://ssw.jku.at/Teaching/Lectures/CSharp/Tutorial/: Institut für Systemsoftware, Johannes Kepler Universität Linz, Fachbereich Informatik. p. 26. Retrieved 2011-08-05.
2. https://developer.mozilla.org/en-US/docs/Web/API/EventTarget.addEventListener
3. Mouse and Keyboard Events in Windows Forms (http://msdn2.microsoft.com/en-us/library/aa983670(VS.71).aspx). Microsoft. Retrieved on February 12, 2008.

External links

- Article *Event Handlers and Callback Functions (http://w3future.com/html/stories/callbacks.xml)*
- A High Level Design of the Sub-Farm Event Handler (https://web.archive.org/web/20070211111543/http://atddoc.cern.ch/Atlas/Notes/061/Note061-1.html)
- An Events Syntax for XML (http://www.w3.org/TR/xml-events/Overview.html#section-eventhandlers)
- Distributed Events and Notifications (http://www.jini.org/wiki/Jini_Distributed_Events_Specification#Distributed_Events_and_Notifications)
- Event order (http://www.quirksmode.org/js/events_order.html)
- `Java DOM Interface Event (https://docs.oracle.com/javase/10/docs/api/org/w3c/dom/events/Event.html)` Javadoc documentation
- `java.awt.event (https://docs.oracle.com/javase/10/docs/api/java/awt/event/package-summary.html)` Java package Javadoc API documentation
- `javax.swing.event (https://docs.oracle.com/javase/10/docs/api/javax/swing/event/package-summary.html)` Java package Javadoc API documentation
- Write an Event Handler (http://hikwww2.fzk.de/hik/orga/verdi/rs/Dokumentation/Cpp/Cpp/ioc/tasks/t90us014.htm)