**JavaScript Event Handling**

**Events** are basically actions that the user makes in order to interact with the web page which, in turn, signals the browser to make dynamic changes within the page. Some examples of these include: mouse clicks over certain elements, keyboard presses, occurrences of errors and others. After events are signaled, a function that is usually user defined, called an **event handler** (or **event listeners**)runs in response to it. When this function is defined, it is called **event registration** (MDN webdocs).

Some of the most common standard DOM events include the following (javascript.info):

* click
* contextmenu
* mouseover/mouseout
* mousedown/mouseup
* mousemove
* submit
* focus
* keydown, keyup
* trasitioned

Two common ways that JS notifies the Web platform of DOM events are by using the addEventListener() function and by specific on-event handlers.

**DOM on-event handlers** are a group of properties by DOM elements that tells specified elements on how to respond to the event (MDN webdocs). Some of the common on-event handlers are as follows:

* onchange
* onclick
* onmouseover
* onmouseout
* onkeydown
* onload

The addEventListener() function provides a generalized way to handle events. It specifies a function to be called whenever a specific event is being signaled to a target. Its syntax is as follows (MDN webdocs):

* target.addEventListeners(type, listener[, options]);

Where the parameters are:

* + type – string that represents the event type
  + listener – a JavaScript function or an object implementing the EventListener interface which receives the signal when the specified event occurs.
  + options – an optional parameter that specifies characteristics about the event listeners.

**Event Propagation Order for DOM-compliant browsers**

The Event Propagation Order is the order in which the event handlers are called according to the nesting of the elements in the HTML.

1. **Capturing Phase** – the capture phase is where propagation occurs from the window down to the event target element. This is implemented when the third parameter of the addEventListener() function has a value of true.
2. **At Target Phase** – the at target phase is where propagation is already at the event target element and all registered listeners will be invoked.
3. **Bubbling Phase –** the bubbling phase is where the propagation occurs from the event target element to the window. This is implemented when the third parameter of the addEventListener() function is false or if it has no third parameter which, by default, means that it is false.

**Event Registration**

1. **Inline HTML –** non-standard, universally supported, not recommended

e.g. <button onclick="this.innerHTML = Date()">Time</button>

1. **Pre-DOM L2 –** non-standard, universally supported

* can register only one handler
* remove the event by attaching the null attribute

e.g. button.onclick = function(){yourCode};

1. **DOM L2** – standard, well-supported

* Bubbling: addEventListener(‘click’, handler)
* Capturing: addEventListener(‘click’, handler, true)
* Can register more than one handler including the handlers for capturing and bubbling
* To remove an event listener, use the removeEventListener() function (has the same parameter as the addEventListener() function)
* Use the dispatchEvent() function o trigger an event programmatically

**NOTE:** You cannot remove anonymous functions specified as paramaters in the addEventListener() function because of its uniqueness.

e.g. button.addEventListener(‘click’, handler);

1. **Microsoft-specific –** non-standard, for Internet Explorer version 6-10

* has only the attachEvent() function for propagation (considered bubbling)
* can register more than one handler
* use the detachEvent() function to remove an event listener and fireEvent() to trigger an event programmatically.

e.g. (help)

1. **Cross-browser by use of feature detection**

e.g. (help)