7. JavaScript Events

Motto:

Do you think I can listen all day to such stuff?

– Lewis Carroll

Events

- Events and event handling
  - make web applications more responsive, dynamic and interactive
  - programming by callbacks
- JavaScript events
  - allow scripts to respond to user's interactions with elements on a web page
  - can initiate a modification of the page

Event Handlers

- Event handler
  - function that is called in when an event occurs
  - typically associated with an XHTML element
  - must be registered
    - i.e., the association must be specified

- Event handler registration
  - two methods
    - inline
    - programatically (traditional method)
    - event property of a DOM

Inline Registration

- Inline registration specifies JS code to be executed when an event occurs directly in the attribute of an XHTML element
  - attribute name corresponds to the event
    - e.g., `onclick="…"`, `onload="…"`
  - attribute value is the JS code
    - typically call of the event handler function
      - e.g., `onload="init()"`
    - but can be a any sequence of JS statements
      - e.g., `onclick="move();score();feedback()"`

Inline Registration Example

```
<html>
  <head>
    <script type="text/javascript">
      function init() {
        …
      }
      function doIt() {
        …
      }
    </script>
  </head>
  <body onload="init()">
    <input type="button" value="Do It!" onclick="doIt()"/>
  </body>
</html>
```

Programatic Registration

- Programatic registration assigns a handler within JS code
  - e.g.,
    - `element.onclick = handler;`
  - or
    - `element.onclick = function () {…};`
  - only a handler can be instantiated
  - allows for registration of handlers to numerous elements in a loop, e.g.

```
for (var i = 0; i < size; i++) {
  var element = document.getElementById("tile" + i);
  element.onmousedown = startDrag;
}
function startDrag () {…};
```

Registration Gotchas

- When using the programatic registration, you can't pass parameters to the event handler
- You can pass parameters using inline registration
  - but watch out for string delimiters
  - instead of `"` or `'`, you have to use entities, e.g. `&quot;`
  - e.g.

```
<table><tbody><tr>
  <td><div onclick="move("&quot;tile-1&quot;</td>
  <td><div onclick="move("&quot;tile-2&quot;</td>
</tr></tbody></table>
```

- When referring to DOM elements in `onload`
  - be sure that the elements already exist
  - e.g. just when `<head>` is has finished loading, `<body>` does not exist, yet

Event and `this`

- An event consists of three pieces of information
  - name of the event
  - element to which it was delivered
  - an event object

- Name is implicit
  - you know which handler was called
- Associated element
  - inside the handler, `this` refers to the element
  - you can use the same handler for different elements
    - then you can differentiate by the `id` of `this` element
- Event object
  - stores contains additional event information

Event Object Properties

- Properties of Event Object contain event information
  - `clientX`, `clientY` property
    - mouse coordinates relative to the element
  - `pageX`, `pageY` property
    - mouse coordinates relative to the page
  - `screenX`, `screenY` property
    - mouse coordinates relative to the screen
  - `shiftKey`, `ctrlKey`, `altKey` property
    - `true` if the *Ctrl, Shift* or *Alt* key was pressed, `false` otherwise
  - `which` property
    - unicode value of the key pressed
  - `timeStamp` property
    - when the event occured
- Browsers differ in other properties of the Event Object
  - `target` property in Firefox
    - when mouse event is fires on an `<img>` element, its `src` attribute
  - `srcElement` property in IE (Internet Explorer)
    - the same info

Event Object

- Browsers also differ in how to obtain the Event Object
  - in Firefox (and other W3C-compliant browsers)
    - Event Object is passed as the parameter to the handler
  - in IE (Internet Explorer)
    - Event Object is the property of `window`

- You must write code that distinguishes the browsers
  - Rule: **Don't determine which browser is being used, check for the existence of object(s) instead**
  - see code on the next slide
- Writing cross-browser functional code is arguably **the most annoying component of web programming**

Event Object Example

```
<html>
  <head>
    <script type="text/javascript">
      function doIt(event) {
        if (!event) {event = window.event;}
        if (event.shiftKey) {
          … // handle Shift-click
        } else if (event.ctrlKey) {
          … // handle Ctrl-click
        } else {
          … // handle regular click
        }
      }
    </script>
  </head><body>
    <input type="button" value="Do It!" onclick="doIt()"/>
  </body>
</html>
```

Mouse Events

Event	Fires When
`onclick`	mouse button is clicked
`ondblclick`	mouse button is double-clicked
`onmousedown`	mouse button is pressed down
`onmouseup`	mouse button is released
`onmousemove`	mouse moves
`onmouseover`	mouse enters an element
`onmouseout`	mouse leaves an element

- E.g., dragging is implemented using
  - `onmousedown` on an element to start dragging it
  - `onmousemove` to move the element being dragged
  - `onmouseup` to end dragging

Keyboard Events

Event	Fires When The User
`onkeypress`	presses then releases a key
`onkeydown`	pushes down a key
`onkeyup`	releases a key

Loading Events

Event	**Fires When**
`onload`	an element (incl. its subtree) has been loaded
`onunload`	page is about to be unloaded
`onabort`	image transfer has been interrupted by user

Selection and Focus Events

Event	Fires When
`onselect`	text selection begins (inside either <input type="text"> or <textarea>)
`onchange`	when a text input is changed and the element loses focus, or new choice is made in a select element
`onfocus`	form element gains focus
`onblur`	form element loses focus

Other Events

Event	**Fires When**
`onresize`	user resizes a window or a frame
`onsubmit`	form is submitted, i.e., the user clicks the reset button
`onreset`	form is reset, i.e., the user clicks the reset button

- Note:
  - Firefox and IE also define their own events

Default Event Handling

- Some elements have a default action for handling events
  - e.g., when the user clicks a "Submit" button in a form, the content of the form will be sent to the server

- You can determine whether the default action is executed
  - return **`false`** from your handler to avoid the default action
  - to execute the default action, either
    - return **`true`** from the handler
    - or don't return anything
    - the default action is executed after your handler

Event Bubbling

- Event bubbling
  - after an event is delivered to (and handled by) the corresponding element, it is also delivered to the parent element
    - and to the parent's parent, etc.
  - events “bubble up” to the ancestor elements
- Typically, once the event is handled, you don't need it to be also handled by the ancestors (and mostly you don't want it to)
  - then cancel the bubbling at the end of the event handler
    - set the **`cancelBubble`** property of the event object to **`true`**