JavaScript: Events

1. Events

1.1 Event Registration

JavasScript normally runs code sequentially which means in the order that the browser reads it from your file. However, we can write JavaScript code that will run only when something happens in the browser. that is because browsers can trigger events when things happen to elements on your page. JavaScript Events can happen at any time for example: when a page loads in the browser, when the user moves the mouse over a link, when a video has finished loading or when a form is submitted. All those things are events so if you want to take care of a task when something happens you have to capture that event and that is done through a process called Event Registration. This means telling the browser that you want to do something when an event takes place and there is several ways of doing that. Some are more compatible with older browsers than the others.

1. Using tag attributes: by adding an event handler to any element. It is considered extremely bad practice because it makes your code hard to update, it is much better to place the JavaScript outside the HTML tags. Example:

   ```
   <li><img src="images/pnk.gif" alt="pink" onclick="alert('pushed pink')" ></li>
   ```

2. Using dot notation: check for events by attaching them to a specific element. That means that you first specify which element in the DOM you want to get to and then specify which event you want to track. When that event happens to that element, your code will execute.

```
<li><img src=”images/pnk.gif” alt=”pink” id=”pink” ></li>
<script>
Document.getElementById(‘pink’).onclick=function() {
alert(‘clicked on pink’)
}
<script>
```

Note: The function literal (Or The Anonymous Function) just means a function without a name.

3. Using addEventListener() method : it is the more modern way of handling events. It asks for the type of the event, a function and the propagation type. The huge advantage to using this way is that it allows you to check for multiple events within a single call, this is known as event propagation. The other less common advantage is that events can be triggered by non-DOM elements, which means that with this method you can write your own events, and different parts of your code can listen for those events that you’ve created yourself. And then you can execute pieces of code based on that. One disadvantage is that it is not supported by IE 8 and older browsers but they do support a separate attach event method.

```
<li><img src=”images/pnk.gif” alt=”pink” id=”pink” ></li>
<script>
Document.getElementById(‘pink’).addEventListener(‘click’,
function() {
alert(‘clicked on pink’);
}, false);
<script>
```

1.2 Legacy Browsers

Although support for old browsers is becoming less important as the web platform matures, it is still a good idea to review some of the options that we have. When trying to make things work with legacy browsers this is a special problem with the event model because of differences in how older browsers handle events. The main problem happens in IE 8 and earlier. These browsers have a different model for handling events. Instead of using addEventListener() method that we use in modern browsers they use a similar method attachEvent() . Notice that it does not use event propagation parameter. If support to older browser is super critical for your project, you may consider a framework like jQuery. Thankfully, support for ancient browsers is becoming less critical.

Pure JavaScript support in older browsers:

www.dustindiaz.com/rock-solid-addevent

```
<li><img src=”images/pnk.gif” alt=”pink” id=”pink” ></li>

<script>
If (window.addEventListener) {
Document.getElementById(‘pink’).addEventListener(‘click’, function() {
alert(‘clicked on pink’);
}, false);
}  else if (window.attachEvent) {
Document.getElementById(‘pink’).atachEvent(‘onclick’, function() {
alert(‘clicked on pink’);
} );
}
<script>
```

1.3 Event Properties

Once you register and capture an event, you receive an event object back from the browser. The object that you get might look different depending on the browser that you are using and that is important to remember because you will not always be able to use all this information. The data you will receive will also depend on the type of event you requested but there is a lot of info that's common to all events. You can get a lot of info about your event and the browser environment.

```
<script>
Document.getElementById('pink').addEventListener('click', function(e) {
Console.log(e);      //output the event (MouseEvent).
}, false);
<script>
```

Note: e is the event object (event variable) being passed to the function. It return the information about the event when we click on an element.

Event Information

1. Event Info

type : is the type of the event. In this case, it will be a click event.
myEvent.type Returns a string containing the type of event.

timestamp : the time at which the event happened so you can do things based on when something happens.
myEvent.timestamp Returns the time (in milliseconds since the epoch) at which the event was created.

defaultPrevented : represent whether or not you're preventing the default behavior from happening when somebody does this event. For example when you click on a link but you want to prevent that link from going to the web page because you wanted to do something slightly different with it, this would show a true value because you prevented that default action.

myEvent.defaultPrevented Returns a boolean indicating whether or not event.preventDefault() was called on the event.

2. Targeting Info

currentTarget : the element the event was assigned to. That's not necessarily the event that you clicked on because you can assign the event to be tracked by another item. For example, right now we're tracking the img element itself but we could be tracking a click on the containing li element instead. And that's why those can be slightly different.
myEvent.currentTarget Identifies the current target for the event, as the event traverses the DOM. It always refers to the element the event handler has been attached to as opposed to event.target which identifies the element on which the event occurred.
Example
event.currentTarget is interesting to use when attaching the same event handler to several elements.

target : is the element that the event originated from which may differ from the element the event was assigned to.
myEvent.target This property of event objects is the object the event was dispatched on. It is different from event.currentTarget when the event handler is called in bubbling or capturing phase of the event.
Example
The event.target property can be used in order to implement **event delegation**.

srcElement : is the actual element that fired the event.

fromElement and toElement : they are kind of related to each other and they deal with mouseover and mouseout events.

Note: Targeting info is going to be vary between IE and other browsers:

www.javascriptkit.com/jsref/event.shtml

3. Coordinate Info

screenX and screenY : gives you the position relative to the user screen.
myEvent.screenX Returns the horizontal coordinate of the event within the screen as a whole (the offset from the left side of the screen in pixels).
myEvent.screenY Returns the vertical coordinate of the event within the screen as a whole (the offset from the top of the screen in pixels).

clientX and clientY : gives you the position relative to the window.
myEvent.clientX Returns the horizontal coordinate within the application's client area at which the event occurred (as opposed to the coordinates within the page). For example, clicking in the top-left corner of the client area will always result in a mouse event with a clientX value of 0, regardless of whether the page is scrolled horizontally.
myEvent.clientY Returns the vertical coordinate within the application's client area at which the event occurred (as opposed to the coordinates within the page). For example, clicking in the top-left corner of the client area will always result in a mouse event with a clientY value of 0, regardless of whether the page is scrolled vertically.

offsetX and offsetY : gives you the position relative to the element that fired the event.

pageX and pageY : gives you the position relative to the HTML document.
myEvent.pageX Returns the horizontal coordinate of the event relative to whole document. it is an integer value in pixels for the X coordinate of the mouse pointer, relative to the whole document, when the mouse event fired. This property takes into account any horizontal scrolling of the page.
myEvent.pageY Returns the vertical coordinate of the event relative to the whole document. It is an integer value in pixels for the Y coordinate of the mouse pointer, relative to the whole document, when the mouse event fired. This property takes into account any vertical scrolling of the page.

layerX and layerY : gives you the position relative to other positioned event. And that has to do with CSS positioning.
myEvent.layerX Returns the horizontal coordinate of the event relative to the current layer. It is an integer value in pixels for the X coordinate of the mouse pointer, when the mouse event fired.
myEvent.layerY Returns the vertical coordinate of the event relative to the current layer. It is an integer value in pixels for the Y coordinate of the mouse pointer, when the mouse event fired.

Note: so be careful when relying on these, they can be a little inaccurate and not all of them available in all browsers. To check these events and their compatibility with browsers visit: quirksmode.org

4. Keyboard and Mouse Info

charCode / keyCode : used for detecting which character was pressed. Returns the Unicode value of a non-character key in a keypress event or any key in any other type of keyboard event.
myEvent.keyCode returns the Unicode value of the key that was pressed.

altKey : Indicates whether the ALT key was pressed when the event fired.
myEvent.altKey returns true or false, depending on whether the alt key was held down or not, when the event fired.

ctrlKey : Indicates whether the CTRL key was pressed when the event fired.
myEvent.ctrlKey returns true or false, depending on whether the ctrl key was held down or not, when the event fired.

shiftKey : Indicates whether the SHIFT key was pressed when the event fired.
myEvent.shiftKey returns true or false, depending on whether the shift key was held down or not, when the event fired.

Note: for more info about keyboard events visit:
www.unixpapa.com/js/key.html

button : Indicates which mouse button caused the event.
myEvent.button This property returns an integer value indicating the button that changed state.

- 0 for standard "click"; this is usually the left button for a right-handed mouse and right button for a left-handed mouse.
- 1 for middle button; this is usually a click on the scroll wheel's button.
- 2 for right button; this is usually a right-click on a right-handed mouse and left-click on a left-handed mouse.

More info about event properties:
http://www.quirksmode.org/js/events_properties.html

1.4 Event Propagation

Event propagation lets you have a single element capture all the events of its children elements. It is the reason you want to use the newer addEventListener() model although that model is not compatible with older versions of IE.

So lets say that you have a group of images coded as a list of elements and you want to do something when the user clicks on any one of these images. In the older model, you have to create an event listener for each image on the list. But with event propagation I can check for any events happening inside the ul .

```
<ul id=”grid”>
   <li><img src=”images/pnk.gif” alt=”pink” id=”pink” ></li>
   <li><img src=”images/org.gif” alt=”orange” ></li>
   .
   .
   .
</ul>
```

```
Document.getElementById('grid').addEventListener('click', function(e) {
        Console.log(e.toElement.alt);
}, false );
Document.getElementById('pink').addEventListener('click', function(e) {
        Console.log("clicked inside the UL");
}, false );
```

It gives us the name (pink, orange …etc.) of the item that we've clicked on. So instead of having to add a lot of event listeners I was able to do the same with just one event listener.

Now that's sounds great but there is a downside of event propagation and that is that different browsers support different propagation models. The order the browser capture the events is important because it can catch events in two ways.

When the click happens it can move down the DOM and notice or register that the click happened inside the ul first then on the li and finally on the img this is called catching an event in the capturing phase (capturing goes down the DOM).

The other way to register the event is by having the browser notice the event first on the lowest element (the img) and then passing the event up the DOM chain until it reaches the ul. this is called catching an event in the bubbling phase because the event bubbles from the li to the ul (bubbling goes up the DOM) .

The last argument in the addEventListener() method controls the propagation type (true for the capturing phase or false for the bubbling phase). This doesn't make any sense unless you're trying to capture two different events.

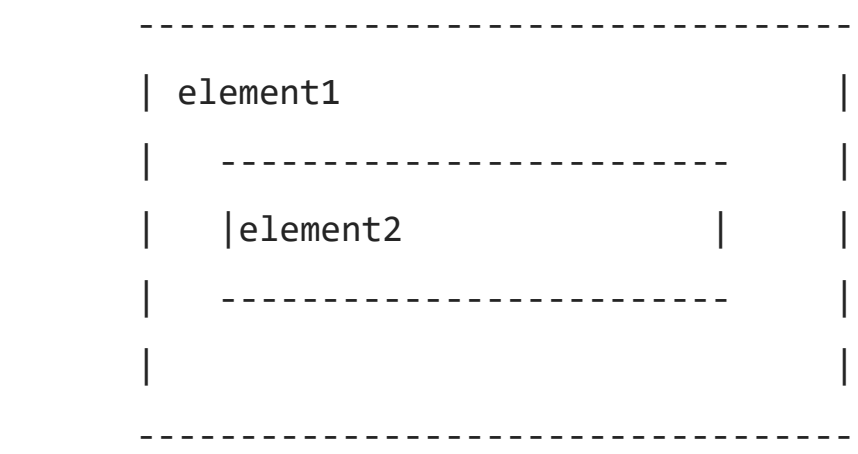
If both of the propagation arguments are set to false (bubbling phase) then it will first register the li (pink) and then the parent ul element.

If both of the propagation arguments are set to true (capturing phase) then it will first register the parent ul element and then the li (pink).

Event order

In the Introduction to events page I asked a question that at first sight seems incomprehensible: “If an element and one of its ancestors have an event handler for the same event, which one should fire first?” Not surprisingly, this depends on the browser.

The basic problem is very simple. Suppose you have a element inside an element



and both have an onClick event handler. If the user clicks on element2 he causes a click event in both element1 and element2. But which event fires first? Which event handler should be executed first? What, in other words, is the *event order*?

Two models

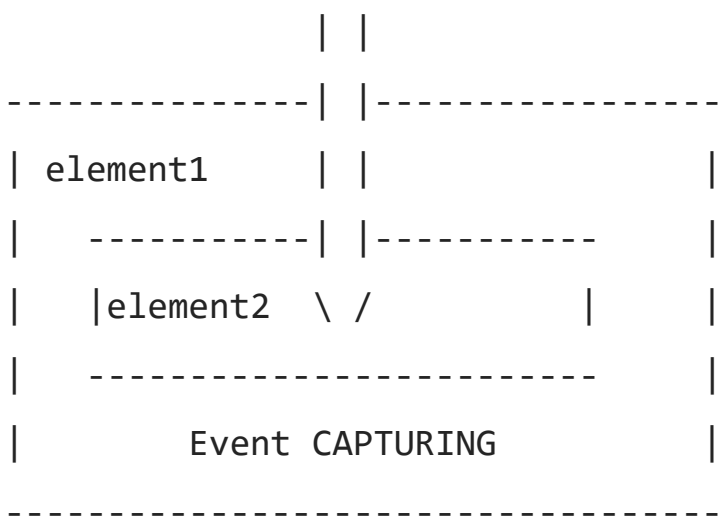
Not surprisingly, back in the bad old days Netscape and Microsoft came to different conclusions.

- Netscape said that the event on element1 takes place first. This is called event *capturing*.
- Microsoft maintained that the event on element2 takes precedence. This is called event *bubbling*.

The two event orders are radically opposed. Explorer only supports event bubbling. Mozilla, Opera 7 and Konqueror support both. Older Opera's and iCab support neither.

Event capturing

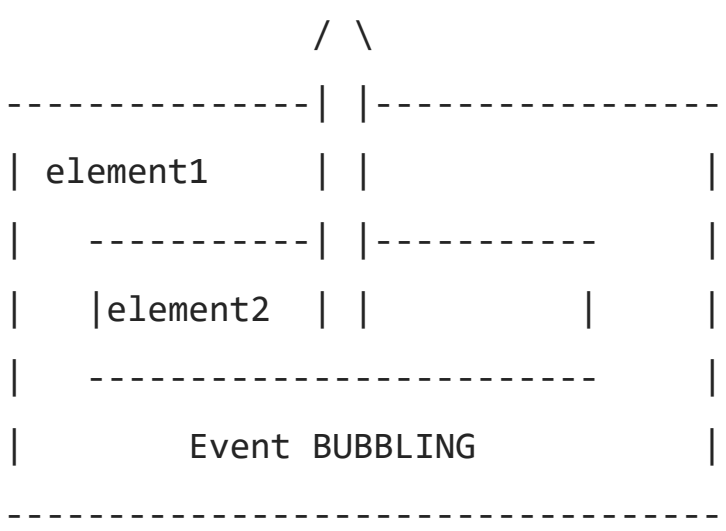
When you use event capturing



The event handler of element1 fires first, the event handler of element2 fires last.

Event bubbling

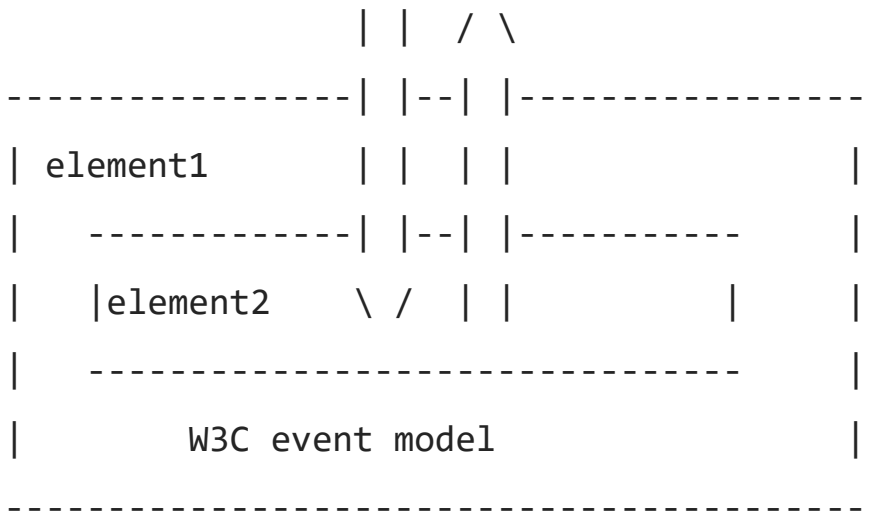
When you use event bubbling



The event handler of element2 fires first, the event handler of element1 fires last.

W3C model

W3C has very sensibly decided to take a middle position in this struggle. Any event-taking place in the W3C event model is first captured until it reaches the target element and then bubbles up again.



You, the web developer, can choose whether to register an event handler in the capturing or in the bubbling phase. This is done through theaddEventListener() method explained on the Advanced models page. If its last argument is true the event handler is set for the capturing phase, if it is false the event handler is set for the bubbling phase.

http://www.quirksmode.org/js/events_order.html

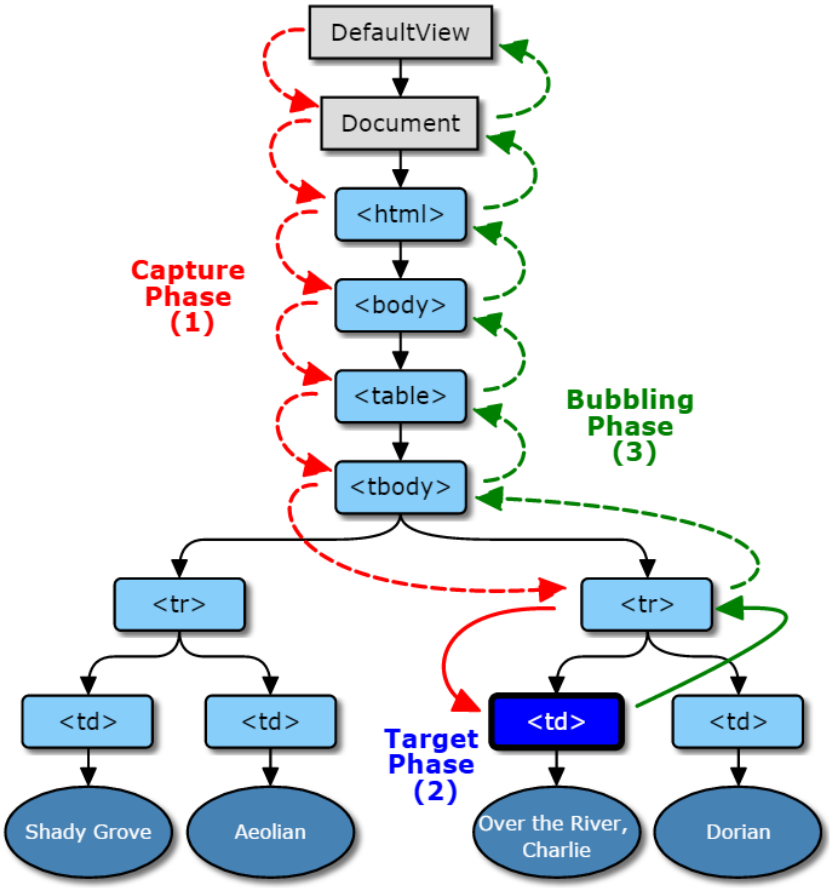


Fig. 1 Graphical representation of an event dispatched in a DOM tree using the DOM event flow

http://www.w3.org/TR/DOM-Level-3-Events/#event-flow

1.5 Stopping Propagation

Event propagation can save you a lot of time by letting you capture an event on a parent element instead of having to register events for individual items but on occasion, you may want to have an event stop propagating through the DOM chain. We can do that with the ***stopPropagation()*** method. If you are trying to support IE 8 or older versions of it, you can do this by setting the property ***cancelbubble*** = true.

Note: bubble is compatible with more browsers than capture.

Example: When we click on the pink item then only the pink will happen. It stops propagating the event up the DOM chain. The other ones will still work just fine but whenever I click on the pink item, it's just going to register that event without propagating back.

```
Document.getElementById('grid').addEventListener('click', function(e) {
Console.log(e.toElement.alt);
}, false );
Document.getElementById('pink').addEventListener('click', function(e) {
Console.log("clicked inside the UL");
e.stopPropagation();        //Prevents further propagation of the current event.
}, false );
```

So ***stopPropagation()*** is a great way to have control over how things happen in your web application. It's part of mastering events.

1.6 Preventing default behavior

A lot of events have default behaviors that take place when the event happen. So for example, when we click on a link the default behavior is to follow that link probably onto a website. Or when a user tries to submit a form the default behavior is to take that form data and pass it to a form processor you might want to stop that so you can do you your own validation. In order to control how an event reacts to it's default behavior we use a method called ***preventDefault()*** .

Example: When I click on any of the items, it's going to capture the event and it's going to display the event information in the console but it's not going to go to a website.

Add anchor tags to all the list items (wrapping the images).

```
Document.getElementById('grid').addEventListener('click', function(e) {
e.preventDefault();          // Cancels the event if it is cancelable, without stopping further propagation of the event.

Console.log(e.toElement.alt);
}, false);
```

2. Working with Common Events

2.1 Removing DOM Elements with Events

Once you learn how events work, you will find out that no matter what events you're working with they are all handled the same way. The type of event and the action that performed might be different but the sequence usually the same.
In the next few lessons, we are going to practice what we've learned about events. We are going to focus on the modern event model the one that allows you to place a single event handler for group of elements using ***addEventListener()*** .
We're going to start with an example which uses click events. It's going to be more practical example. You will find when you're working with real elements things will be a little bit different and you will have to take care of some additional things that you may have not thought of.

Art Eliminator Example: We got a page with a series of images and we want to help the user to pick his favorite piece of art from our list by the process of elimination. So when they click on each one of these images it's going to get rid of them. Then when they click on the last one, it should show them information about the one that they picked.

```
<div id=”art”>
 <p>general description</p>
 <ul class="grid">
   <li><img src="images/3_tn.jpg" alt="Hassum"></li>
   <li><img src="images/2_tn.jpg" alt="LaVone"></li>
   <li><img src="images/1_tn.jpg" alt="Lorenz"></li>
 </ul>
</div>
```

```
document.querySelector('.grid').addEventListener('click', function(e) {
 var removeTarget = e.target.parentNode;
 removeTarget.parentNode.removeChild(removeTarget);
}, false);
```

If we click on anything including the ul element it's going to fire all the events (the function code).

removeTarget //The li element that we want to remove.

e.target //The (target : IMG) property of the event handler.

e.target.parentNode //The parent of img element is the li element.
To remove the whole list item not just the image.

var oldChild = parentElement.removeChild(child);
The method removes a child node from the DOM. Returns removed node.
Since we want to remove a child from the ul .

- child is the child node to be removed from the DOM.
- oldChild holds a reference to the removed child node. oldChild === child.

2.2 Solving Event Issues

There are two problems:
First if we click inside the ul box then everything will go away because our code says that if we clicked on anything that has the grid class it needs to get rid of that element's parent.
Second we can click on all the list items and remove the last element which is not good because we are trying to pick a piece of art so it means that whenever we get to the last element it should give us some information about the last image.

Note: Use ***console.dir(e.target)*** to look at the properties of the object that we've clicked on.

We are going to fix the first problem by using the property ***(tagName: IMG)*** to make sure that it only executes the code if we're clicking on an image.
if (e.target.tagName === 'IMG')
To solve the second problem we're going to create a variable that's going to keep track of how many elements we have right now.
this.querySelectorAll('li') ask for li items within ***this*** element. The ***this*** element happened to be the node that we're on, so it going to include

anything that's related to event that we're currently on. In other words, it's just going to include the ul list.

```
console.log(e);              //Returns MouseEvent object.
console.log(e.target);       //Returns ul or img element depending on where you clicked.
console.log(this);           //Returns ul always, because that's the event that was caused by this listener querySelector('.grid') .
```

```
document.querySelector('.grid').addEventListener('click', function(e) {

  if (e.target.tagName === 'IMG') {
    var howmany = this.querySelectorAll('li').length;          //Count how many li items we have inside the ul every time we click on something.

    if (howmany > 1) {
        //Don't execute the following code if we have only one item.
      var removeTarget = e.target.parentNode;
removeTarget.parentNode.removeChild(removeTarget);

    } else {

      var photoTitle = e.target.alt;
      document.querySelector('#art p').innerHTML = "<p>You've picked: "
+ photoTitle + "</p>";

    } //How many.
  } // Check to see that I clicked on IMG only.
}, false); // Click event.
```

So whenever you'll work with events you'll find that it's not just about handling the click event it's about taking care of the whole interface that you're working with.

2.3 Creating DOM Elements with events

2.4 Removing an Event

This time we will work with different event called ***mouseover*** and in the old days of the web, developers used it to create rollovers. It's not popular today as it used to be because today we use CSS for that but it's still useful for certain things.

The mouseover event is fired when a pointing device is moved onto the element that has the listener attached or onto one of its children.
The mouseout event is fired when a pointing device (usually a mouse) is moved off the element that has the listener attached or off one of its children.
The click event is fired when a pointing device button (usually a mouse button) is pressed and released on a single element.

Example project: show a bigger version of the image thumbnail that we mouseover it.

Each image thumbnail has a high resolution version of it that has the exact same name without the (_tn.jpg) .

When you rollover the thumbnail images you'll get multiple high resolution versions showing up and when you'll rollover these big images you would get these other errors like empty images. So we want to deal with the ***mouseout*** event. With a lot of events especially an event like mouseover you have to worry not just about the event itself but any related events that may come up when you work with the JavaScript file.

Uncaught TypeError: Cannot read property 'parentNode' of null.
The errors increase exponentially in the console because some of the events are still being called. So what's happening is I get ***mouseout*** event and then the second time I rollover and I rollout I am getting 2 ***mouseout*** events. So there is bunch of mouse events that are staying in the event queue, that are not being processed and that's what causing this error. Events in JavaScript work by inserting themselves into a queue and they will remain there until they're needed. So if you don't remove some events it will cause problems like this. And it sort of hard to figure out because it's not doing anything visually so you're getting a bunch of errors and you don't even know it.

The HTML DOM during the rollover.

```
<ul class="grid">
  <li>...</li>
  <li>
    <img src=”image_tn.jpg”>
    <div class=”preview”>
      <img src=”image.jpg”>
    </div>
  </li>
  <li>...</li>
</ul>
```

```
document.querySelector('.grid').addEventListener('mouseover',
function(e) {
 if (e.target.tagName === 'IMG') {

   var myElement = document.createElement('div');
   myElement.className = 'preview';  // Add the CSS class.
   e.target.parentNode.appendChild(myElement);   //Attach the div
element to the li element that have the image inside it.

   var myImg = document.createElement('img');
   var imgLoc = e.target.src; //The location of the image that we've rolled
over with the (_tn.jpg) .
   myImg.src = imgLoc.substr(0, imgLoc.length-7) + '.jpg'; //Get rid of the
(_tn.jpg) to have the name of the high resolution image.
   myElement.appendChild(myImg);

   e.target.addEventListener('mouseout', function handler(d) {
//This new event is only going to happen when a mouseover happens.
     var myNode = d.target.parentNode.querySelector('div.preview');
     myNode.parentNode.removeChild(myNode);


     e.target.removeEventListener('mouseout', handler, false);
   }, false);

 } // check to see that I clicked on IMG only
}, false); // click event
```

Note: we need to give a name for the ***mouseout*** event function only because we want to remove it. ***mouseover*** function doesn't need a name because it doesn't need to be removed.

target.removeEventListener(type, listener[, useCapture])

Removes the event listener previously registered with EventTarget.addEventListener.

type
A string representing the event type being removed.

listener
The listener parameter indicates the EventListener function to be removed.

useCapture Optional
Specifies whether the EventListener being removed was registered as a capturing listener or not. If not specified, useCapture defaults to false.

So even though sometimes you don't see any errors visually with events you've got to be careful and always check your console. If you didn't get rid of that element it will eventually generate problems with memory that you have not known but your users might definitely feel whenever a browser crashes because there is too many events in the queue.

2.5 Preventing Default Events

Example project: Right click on the image to see a larger version. The preview will follow the mouse.
In this example we will prevent the default actions when working with events. We're going to use an event that handles the mouse right clicks called ***contextmenu*** . And we'll use another event called ***mousemove*** to track the position of the mouse.

When we right click, we will get the bigger photo however we're also getting the pop-up menu that's because the default action for right click is to bring the OS's menu. We can use ***contextmenu*** event to create drop-down menus that are customized to whatever we're doing at the moment and this is how this event works.

The contextmenu event is fired when the right button of the mouse is clicked (before the context menu is displayed), or when the context menu key is pressed (in which case the context menu is displayed at the bottom left of the focused element, unless the element is a tree, in which case the context menu is displayed at the bottom left of the current row).

The mousemove event is fired when a pointing device (usually a mouse) is moved while over an element.

```
document.querySelector('.grid').addEventListener('contextmenu',
function(e) {
  e.preventDefault(); //Prevent the default right click pop-up menu.
  if (e.target.tagName === 'IMG') {

    var myElement = document.createElement('div');
    myElement.className = 'preview';
    e.target.parentNode.appendChild(myElement);

    var myImg = document.createElement('img');
    var imgLoc = e.target.src;
    myImg.src = imgLoc.substr(0, imgLoc.length-7) + '.jpg';

    myElement.style.left = e.offsetX + 15 + 'px';
    myElement.style.top = e.offsetY + 15 + 'px';
    //The image appears at its original position wherever we put it before
and then it jumps as soon as we move the mouse but we want it to be at
the position that we want as soon as it's created.

    myElement.appendChild(myImg);

    e.target.addEventListener('mouseout', function handler(d) {
      var myNode = d.target.parentNode.querySelector('div.preview');
      myNode.parentNode.removeChild(myNode);
      e.target.removeEventListener('mouseout', handler, false);
    }, false);

    e.target.addEventListener('mousemove', function(f) {
      myElement.style.left = f.offsetX + 15 + 'px';
      myElement.style.top = f.offsetY + 15 + 'px';
//Modify the value of the left position and the top position of the style
sheet for this element by using the style properties.
    });

  } // check to see that I clicked on IMG only
}, false); // click event
```

3. Working with Time-Based Events

3.1 Creating a Spinner Graphic for Large Image Loads

There is a ton of events in the DOM and in HTML in general.
https://developer.mozilla.org/en-US/docs/Web/Events
In this chapter we're going to study the on-load events so things that happen whenever you load or something is finished loading are important in JavaScript.

Example Project: shows you how to do an image loading. You click on an image and you have to wait until the image is loaded or you see a preview of a little spinner that shows you that the image is loading and when the image is finished loading then you do something like display the image.

To get rid of the spinner once the image finished loading. To do that we must find out when the load event of that image has executed because every time you create an image that image brings in a load event. So when that image finishes loading it triggers the load event and then you can capture it.

The load event is fired when a resource and its dependent resources have finished loading.

So working with load events is very similar to other events but this is an event that get generated as a result of another event, the click event. It's a good idea to work with this event when working with data that is very large. That way you'll give the user a visual feedback so that they know what's happening.

```
<body>
  <img class="preview" src="images/taquito_tn.jpg" alt="Taquitos">
  <div class="overlay"></div>     //An overlay that appears partially black on top of the entire page while the image is loading and then it will be the background for the new image which is the high resolution version of the image.
  <script src="script.js"></script>
</body>
```

```css
.overlay {
  display: none;      // To hide the overlay image.
  position: absolute;
  top: 0;
  left: 0;
  background-color: rgba(0,0,0,0.7);    // 70% black.
  width: 100%;
  height: 100%;
  z-index: 5000;      // Put the overlay on top of everything.
}
```

```javascript
document.querySelector('img.preview').addEventListener('click',
function(e) {

  var myOverlay = document.querySelector('.overlay');
  myOverlay.style.display = 'block';  //To set the overlay display to block.

  var highRes = document.createElement('img');
  highRes.className = 'bgImg';
  var lowRes = e.target.src;
  highRes.src = lowRes.substr(0, lowRes.length-7) + '.jpg';
  myOverlay.appendChild(highRes);

  var mySpinner = document.createElement('img');
  mySpinner.className = 'spinner';
  mySpinner.src = 'images/spinner.gif';
  myOverlay.appendChild(mySpinner);

  highRes.addEventListener('load', function() {
        // Track load event of the high-resolution image.
    mySpinner.parentNode.removeChild(mySpinner);
  });

}, false);
```

3.2 Playing Media Events

Example project: Build an audio jukebox. This is a type of project that has different kinds of events and elements. So you'll be clicking on a list item, and that's actually going to generate an audio object. Now media elements are super easy to do with HTML5. All you really have to do to play a sound on your page is just create an audio element with a source and maybe some controls.
Now every one of those elements has a Javascript counterpart, and there are methods that will generate media events, like for example, playing a song. And then there's also methods that will be created when a song finishes playing. If you want to dig into the code for audio and media and JavaScript, make sure you check out this page.
http://www.w3schools.com/tags/ref_av_dom.asp

```
<body>
    <ul class="player">
        <li data-src="audio/Phoebex.mp3">Phoebex</li>
// To pass along the location of the MP3 file to the JavaScript when
I click on any one of these elements.
        <li data-src="audio/AmazingLee.mp3">AmazingLee</li>
        <li data-src="audio/NightKitty.mp3" id="playing" >Night Kitty</li>
        <li data-src="audio/EqueKenox.mp3">Eque Kenox</li>
        <li data-src="audio/Shiloah.mp3">Shiloah</li>
    </ul>
    <audio id="playerID" src="audio/NightKitty.mp3"></audio>
     //Created when clicking on any li element.
</body>
```

In CSS, different states for showing how the different list items should look when they're being played or a song has been paused.

```
ul.player li#playing {…}

ul.player li#paused {…}
```

```
var jukebox =
document.querySelector('ul.player');
jukebox.addEventListener('click', function(e) {

        var audioPlayer = document.createElement('audio');
        // audioPlayer is the HTML audio tag.
   audioPlayer.id = 'playerID';
   e.target.id = 'playing';
   audioPlayer.src = songName;
   document.body.appendChild(audioPlayer);
   audioPlayer.play();

}, false);
```

`var audioPlayer = document.createElement('audio');`
Create the audio player, which is going to be just an audio tag.

`document.body.appendChild(audioPlayer);`
Attaching the element to the body. We don't really need it to be related to our list element but we do need it somewhere on the page. I don't want to put it right into a list element because we could be clicking on any one of these songs and I don't want the audio player to be on the first list item I click on. I just want it to be somewhere on the page.

`e.target.id = 'playing';`
Identifying which song we're currently playing. Set the ID of the li element to (playing) that will attach the proper style.
`<li data-src="audio/Phoebex.mp3" id="playing" >Phoebex</li>`

`var songName = e.target.getAttribute('data-src');`
`audioPlayer.src = songName;`
Set the location of the audio player to be a variable song name, which is equal to the target of what I click on.

`audioPlayer.play();`
To play a sound, you have to execute the play method of the sound.
The play event is fired when playback has begun.

`audioPlayer.pause();`

The pause event is fired when playback has been paused.

3.3 Monitoring Media-Ended Events

The problems with this code right now:

1. It doesn't let you stop any of these.
2. You can play more than one song at the same time.
3. When a song finished playing, our player's not going to update. Note that the audio element audioPlayer is still there, and it should really be gone because the song is no longer playing.
4. The UI is not updating the id=”playing” in the li element is still there when a song has finished, and it should be gone.

Note: The nice thing about media is that audio and video elements work almost exactly the same way. You call them exactly the same way. They have the same properties, the same methods. Several properties:

myVid.paused Sets or returns if the audio/video is paused or not.

myVid.ended Returns if the playback of the audio/video has ended or not.

```
    audioPlayer.addEventListener('ended',
function() {

audioPlayer.parentNode.removeChild(audioPlayer);
      e.target.id='';
    }, false);
```

```
audioPlayer.addEventListener('ended', function()
{
```

Detect the ended event, which fires when a song has finished playing.

```
audioPlayer.parentNode.removeChild(audioPlayer);
```

Remove the Audio Player from existence (get rid of the sound elements).

The audio tag was automatically generated, and as soon as the song finished playing, the audio tag got deleted from the dom.

The ended event is fired when playback has stopped because the end of the media was reached.

Visually, it looks like this song is still playing. That's because, in the unordered list, list item, we still have the element with an ID of playing. So obviously, we need to tell it not to have any ID so that it looks like the other ones.

```
e.target.id='';
```

Reset the ID of this element so it looks like all the other elements.

You have to work with these type of event, these type of events generate their own object. And sometimes that object is not directly associated with the element that's started the event, in this case clicking on this image of the jukebox.

3.4 Handling Media Pauses

The only way to pause them is by hitting the refresh page, so I want to create a toggle, something that allows me to pause and continue to play a paused song.

```
audioPlayer.id = 'playerID';
```

Check the document for the existence of a player.

```
e.target.id = 'paused';
<li data-src="audio/Phoebex.mp3" id="paused" >Phoebex</li>
```

```
if (audioPlayer.paused)
```

Check to see if the song that's currently playing has been paused.

3.5 Starting a New Song

We need to find the song that's playing and take a look at whether or not that song is the same as the song that we're going to click on next.

```
if (songName===audioPlayer.getAttribute('src')) {
```

Comparing the source of the currently playing song (from the audio tag) and the source of the song that we've clicked on.

```
var jukebox = document.querySelector('ul.player');
jukebox.addEventListener('click', function(e) {

  var songName = e.target.getAttribute('data-src');

  var audioPlayer = document.querySelector('#playerID');
// Returns the audio tag if it exists or null if it's not.


  if (audioPlayer) {  //check to see if there is a selected song currently (playing or paused) or not.


    if (songName===audioPlayer.getAttribute('src')) {
      // If clicked on the same item (song).
      if (audioPlayer.paused) {
        audioPlayer.play();
        e.target.id = 'playing';
      } else {
        audioPlayer.pause();
        e.target.id = 'paused';
      }

    } else {
      // If clicked on another item (song).

      audioPlayer.src = songName;
      audioPlayer.play();

      // Remove both the playing or paused style of the previous song.
      if (document.querySelector('#playing')) {
        document.querySelector('#playing').id='';
      } else {
        document.querySelector('#paused').id='';
      }

        e.target.id = 'playing';
```

```
    }


  } else {        // if there is no song currently selected (playing or paused).
    var audioPlayer = document.createElement('audio');
    audioPlayer.id = 'playerID';
    e.target.id = 'playing';
    audioPlayer.src = songName;
    document.body.appendChild(audioPlayer);
    audioPlayer.play();


    audioPlayer.addEventListener('ended', function() {
      audioPlayer.parentNode.removeChild(audioPlayer);
      e.target.id='';
    }, false);
  }

}, false);
```

4. Drag and Drop Project

4.1 Introduction

This is going to be a fun drag and drop build your own snowman project. So it has different kinds of events and it also happens to be touch enabled. So you can use this on your iPad or on your iPhone. And you use your fingers to drag the elements around.
I'll start by showing you how to put this together by using an Adobe Illustrator file and taking each of these elements and exporting them as SVG files. That will let your game be more compatible with Retina devices.
We'll also work through some compatibility issues with the different positioning attributes that are available in different browsers. I'll show you how to work with both regular events as well as touch events. And I'll talk about understanding layering issues by resetting the elements layer positions.

4.2 Preparing SVG Assets

Open up the folder and you'll see that there's an artwork.eps and in this project, I wanted to work with images that would scale well in retina devices. So, instead of using PNGs, I wanted to use SVG files.
Now to work with those, the easiest thing to do is to bring these into a program like Adobe Illustrator and export each one of the elements individually.
So, I wanted it to use just a few of these, and what I needed to do to export these, is grab the element that I wanted.

So I'm going to grab the eyes, and I'm going to copy it onto the clip board and then I'm going to make a new file. So I'll do File>New. clicking OK, And then I get a new document with a full page, and just paste this somewhere in my new document. Now, I don't want to have all the white space around the image, so in order to delete that, go to the Object menu, and select Artboards, and then select Fit to Artwork Bounds. If you have the items selected, you can also go to Fit to Selected

Art. That will get rid to all the white space and leave the minimum amount of image in your document. Go to the file menu and select Save as, and in here, you want to choose the option that says SVG, make sure you don't choose SVG compressed. I used a name for each element that went in a different section. What I mean by that is, I used the name eyes and then I gave it a number. I've taken every one of my elements and saved them into images folder.
So, what this is going to do is always give you the best resolution regardless of the device, so it's a good idea to save your graphics in an SVG format.

4.3 Setting Up The Core Files

Alright, we've got an images folder with a bunch of svg documents.

```
<img draggable="true" id="snowman" src="images/body.svg"
alt="snowman" style="position: absolute; left: 50px; top: 120px; z-index:
5;">
```

include the draggable attribute, going to set that to true for all the files.

set the style attributes for every one of the images in the HTML. This is not something I would normally do but in this document, it makes a lot of sense because we're going to be using JavaScript to control the positioning. And JavaScript is going to end up injecting these into your HTML any ways. And this is a great place of previewing where the images are going to go, and helping us lay them out.

Style attribute, and then we're going to pass it along some CSS.
So I'll do style, and then set the position of the object to absolute. What that will do is, allow me place any one of the elements at a specific X and Y position because none of the containers of these images are set to relative, these will all align to the body tag.
Let's go back here we'll do position absolute and then we set up x and y coordinate here. Left, in this case, it would be 50 pixels. And top will be 120 pixels and I am also going to set the z index. The z index is the layering from top to bottom so think of these elements as a stack of cards

anything with a z index higher than nothing will be on top of everything else.
So in here we were just going to set them all to 5 And this is pretty much what we do for any one of the other elements.
Everything has an individual ID as well, so I just create an ID attribute, and make sure that everything has a unique name. They're pretty much matching the name of the files as well.

```
body {
  background: #E4F8F7;
  margin: 0;
  padding: 0;
  width: 1000px;
  height: 1000px;
}

[draggable] {
  cursor: move;
}
```

Set up a width, I'm going to be dealing with static sizes, so I'm going to use a static page width. This is not going to be a responsive layout.
And I'm also going to set a height. Normally I wouldn't set up a height, but in this kind of document you want to make sure that when you drag things there's going to be something behind. And since we're positioning all these elements absolutely, you're going to need the background to have a certain amount of height.

So I'll do draggable. And all I need to do here is modify the cursor. And offset the cursor to be the move attribute, so when you roll over these images, it's going to switch to a special cursor that's designed to work with moving images.

Now you are probably thinking I've spend a tone of time figuring out exactly the X and Y position of each one of these documents, but I am actually going to show you a cool shortcut that you can use when you are doing a project like this. What I've done is gone to the finished version of this file. So, I've already got these laid out exactly how I want to, but if I

wanted to change the way these were laid out all I have to do is look at the complete version of this project. And move these where ever I want em to lay em out. So, if I didn't like this layout and I want to spread things out a little bit or I wanted to just change the position of any one of these elements.
I can just go to the finished version of this project and then manually move things where I want them. And then, right click on any of these elements and select inspect element. And that will tell me the code for every one of those images and the new positions that I've placed them in. So, if we're right here at the very top, let's go ahead and hit this magnifying glass. And click on this one right here, so that's this element right here, since I have moved these with Java Script take a look at this positioning numbers right here.
If I move it again you'll see that they will change and now its just 344 px so whatever I do in the finished version of the code. I can just grab my HTML from here so I can right click on any of these and select Copy as HTML. And once I do that, then I can switch over to this version of the code, and just paste this into my layout. So if I paste this, you'll see that you get the new position for this element. So really, I didn't spend the time mathematically calculating where everything was going to go. I just simply lay them out as I wanted to, and then re-arrange them manually, and then copy all the html.
If you don't know where they're going to go, then what you could do also is get rid of the positioning attributes, so just get rid of all this. Positioning stuff right here. And then layout the file manually. I can't do that with this because we don't have the JavaScript done. But if you just get rid of the left and top positioning here then all the elements will arrange sort of right on top of each other. And then you can just move them around and then copy the HTML for any one of those positions. So, although I never use in-line style sheets, in this project it's going to make a lot of sense.
The JavaScript you create eventually is going to inject this in-line code into your projects any ways. And it's really going to help you design the project ifyou keep this code in your HTML at the beginning.

```
<!doctype html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <title>Snowman</title>
```

```
  <link rel="stylesheet" href="style.css">
  <meta name="viewport" content="width=1200">
  <meta name="apple-mobile-web-app-capable" content="yes">
</head>
<body>
  <img draggable="true" id="snowman" src="images/body.svg"
alt="snowman" style="position: absolute; left: 50px; top: 120px; z-index:
5;">
  <img draggable="true" id="mouth03" src="images/mouth03.svg"
alt="mouth03" style="position: absolute; left: 367px; top: 150px; z-
index: 5;">
  <img draggable="true" id="neck01" src="images/neck05.svg"
alt="neck01" style="position: absolute; left: 333px; top: 201px; z-index:
5;">
  <img draggable="true" id="neck02" src="images/neck02.svg"
alt="neck02" style="position: absolute; left: 342px; top: 341px; z-index:
5;">
  <img draggable="true" id="neck03" src="images/neck03.svg"
alt="neck03" style="position: absolute; left: 349px; top: 484px; z-index:
5;">
  <img draggable="true" id="head03" src="images/head03.svg"
alt="head03" style="position: absolute; left: 515px; top: 417px; z-index:
5;">
  <img draggable="true" id="nose03" src="images/nose03.svg"
alt="nose03" style="position: absolute; left: 525px; top: 294px; z-index:
5;">

  <script src="script.js"></script>
</body>
</html>
```

```
var dragndrop = (function() {
  var myX = '';
  var myY = '';
```

```
var whichArt = '';

function resetZ() {
  var elements = document.querySelectorAll('img');
  for (var i = elements.length - 1; i >= 0; i--) {
    elements[i].style.zIndex = 5;
  };
}

function moveStart(e) {
  whichArt = e.target;
  myX = e.offsetX === undefined ? e.layerX : e.offsetX;
  myY = e.offsetY === undefined ? e.layerY : e.offsetY;
  resetZ();
  whichArt.style.zIndex = 10;
}

function moveDragOver(e) {
  e.preventDefault();
}

function moveDrop(e) {
  e.preventDefault();
  whichArt.style.left = e.pageX - myX + 'px';
  whichArt.style.top = e.pageY - myY + 'px';
}

function touchStart(e) {
  e.preventDefault();
  var whichArt = e.target;
  var touch = e.touches[0];
  var moveOffsetX = whichArt.offsetLeft - touch.pageX;
  var moveOffsetY = whichArt.offsetTop - touch.pageY;
  resetZ();
  whichArt.style.zIndex = 10;

  whichArt.addEventListener('touchmove', function() {
    var positionX = touch.pageX + moveOffsetX;
    var positionY = touch.pageY + moveOffsetY;
    whichArt.style.left = positionX + 'px';
```

```
    whichArt.style.top = positionY + 'px';
  }, false);
}

  document.querySelector('body').addEventListener('dragstart',
moveStart, false);
  document.querySelector('body').addEventListener('dragover',
moveDragOver, false);
  document.querySelector('body').addEventListener('drop', moveDrop,
false);

  document.querySelector('body').addEventListener('touchstart',
touchStart, false);


})();
```