Introduction to Touch events in JavaScript

Date created: Aug 1st, 2013

In this tutorial lets get touchy feely with JavaScript, by examining its touch related events and how they are used to detect and respond to touch and swipe events. With touch based devices ever growing in numbers, grasping these events is as essential as understanding the age old mouse events. Examples in this tutorial can be appreciated in both touch and non-touch enabled devices, with the later falling back to your trusted mouse instead. Ready to give those fingers a bit of a workout? Lets go!

JavaScript Touch Events

So lets dive right into it. The following lists the supported touch events in JavaScript:

JavaScript Touch Events Event Name	Description
touchstart	Triggers when the user makes contact with the touch surface and creates a touch point inside the element the event is bound to.
touchmove	Triggers when the user moves the touch point across the touch surface.
touchend	Triggers when the user removes a touch point from the surface. It fires regardless of whether the touch point is removed while inside the bound-to element, or outside, such as if the user's finger slides out of the element first or even off the edge of the screen.
touchenter	Triggers when the touch point enters the bound-to element. This event does not bubble.
touchleave	Triggers when the touch point leaves the bound-to element. This event does not bubble.
touchcancel	Triggers when the touch point no longer registers on the touch surface. This can occur if the user has moved the touch point outside the browser UI or into a plugin, for example, or if an alert modal pops up.

These events can be attached to any element on the page, and is passed an event object containing details about the touch point, such as its coordinates on the page. Use element.addEventListener() to attach the event(s), for example to the BODY of the page:

```
1    window.addEventListener('load', function(){ // on page load
2
3        document.body.addEventListener('touchstart', function(e){
4            alert(e.changedTouches[0].pageX) // alert pageX coordinate of touch point
5        }, false)
6
7    }, false)
```

Here I've attached the "touchstart" event to document.body once the page has loaded (you may want to do this on DOMContentLoaded instead). Inside the anonymous function for touchstart, we look at the changedTouches object of the Event object, which contains information on each touch point initiated by that touch event on the touch surface. Here we're only interested in the first touch point (ie: finger) that has made contact, specifically, its pageX coordinate on the page when the touch is made.

The Event object whenever a touch event is fired holds a wealth of information about the touch action; you already saw its changedTouches object, which contains information on touch points changed since the last touch event . Lets take the above example a bit further now, by bringing in the touchmove and touchend events to show the distance traveled by a touch action from beginning to end on a DIV, from a finger touching down on an object to lifting up.

Example (mouse simulation added for non touch devices):

Touch Me!

Status: touchend
Resting x coordinate: 712px

Touch then move your finger to see the current state of the touch and the distance traveled. The HTML markup for the DIV consists simply of:

```
1   <div class="box" id="box1">
2   <h3> Touch Me! </h3>
3   </div>
4
5   <h3 id="statusdiv">Status</h3>
```

The script looks like this:

```
1   <script>
2
3   window.addEventListener('load', function(){
4
5      var box1 = document.getElementById('box1')
6      var statusdiv = document.getElementById('statusdiv')
7      var startx = 0
8      var dist = 0
9
10     box1.addEventListener('touchstart', function(e){
11        var touchobj = e.changedTouches[0] // reference first touch point (ie: first finger)
12        startx = parseInt(touchobj.clientX) // get x position of touch point relative to left edge of browser
13        statusdiv.innerHTML = 'Status: touchstart<br> ClientX: ' + startx + 'px'
14        e.preventDefault()
15     }, false)
16
17     box1.addEventListener('touchmove', function(e){
18        var touchobj = e.changedTouches[0] // reference first touch point for this event
19        var dist = parseInt(touchobj.clientX) - startx
20        statusdiv.innerHTML = 'Status: touchmove<br> Horizontal distance traveled: ' + dist + 'px'
21        e.preventDefault()
22     }, false)
23
24     box1.addEventListener('touchend', function(e){
25        var touchobj = e.changedTouches[0] // reference first touch point for this event
26        statusdiv.innerHTML = 'Status: touchend<br> Resting x coordinate: ' + touchobj.clientX + 'px'
27        e.preventDefault()
28     }, false)
29
30  }, false)
31
32  </script>
```

A few points worth mentioning here:

- We call event.preventDefault() to prevent the default action associated with each event from occurring. In the case of touchstart and touchend for instance, if the bound-to element was a link, not suppressing the default action would cause the browser to navigate to the link, cutting short our custom sequence of actions. In the case of touchmove, calling event.preventDefault() stops the browser from scrolling the page when the user is moving his finger inside the bound-to element.
-
- Once again, we access the first element inside event.changedTouches[] for each of the touch events to reference the first touch point made to the element (there could be multiple fingers used), and examine the clientX property to get the horizontal coordinate of the touch point relative to the left edge of the browser (not including any scroll offsets). This property is adequate for what we're trying to do here, which is simply to get the relative distance traveled while a touch is maintained on the element.
-
- To get the distance traveled between touchstart and touchend events, we define a startx variable at the touchstart phase that gets the starting clientX position of the touch. Then throughout the touchmove event, we get the clientX position of the touch and subtract from it the startx value to get the distance traveled while the touch point is maintained.
-
- Notice how the touchend event is still fired and displays the final resting x coordinates even if your finger is outside the bound-to element at the time of lifting up your finger.
-

The object event.changedTouches[] is just one of numerous properties of the Event object that gets populated during touch related events. It's high time to look at this object in detail now.

Event object during Touch

The Event object is this mystical unicorn in JavaScript that contains information on an event when it occurs, whether it's the URL of a link in an onclick event, the keyCode of the key pressed in an onkeypress event etc. With touch related events, the Event object is populated with a slew of unique properties that give us insight into all aspects of the touch point, from how many fingers (or toes for that matter etc) touched down on the touch surface to their precise coordinates on the screen.

Event Object during Touch Property	Description
altkey	Boolean indicating whether the alt key was pressed at time of touch event.
changedTouches	A list of Touch objects representing **each touch point directly involved in this event**. Specifically: • In touchstart, it contains a list of fingers that have made contact with the touch surface during this touchstart event. • In touchmove, it contains a list of fingers that have moved during this touchmove event. • In touchend, it contains a list of fingers that have just been removed from the touch surface during this touchend event. • In touchenter, it contains a list of fingers that have entered the touch surface during this touchenter event. • In touchleave, it contains a list of fingers that have exited the touch surface during this touchleave event. You can use the length property to get the number of Touch objects inside changedTouches[].
ctrlKey	Boolean indicating whether the crtrl key was pressed at time of touch event.
metaKey	Boolean indicating whether the meta key was pressed at time of touch event.
shiftKey	Boolean indicating whether the shift key was pressed at time of touch event.
targetTouches	A list of touch points currently making contact with the touch surface **AND** started out from the same element that is the target of this event. For example, lets say you bind the touchstart event to a DIV and place two fingers down on the surface. targetTouches will only contain information on the finger(s) placed inside the DIV, and not any outside. You can use the length property to get the number of Touch objects inside targetTouches[].
touches	A list of Touch objects representing all touch points currently in contact with the touch surface, regardless of which element a touch point is on at the moment.
type	The type of event that triggered the Event object, such as touchstart, touchmove, etc.
target	The target element of the touches associated with this event.

So for example, during a touchstart event, the Event object's touches property lets us access all touch points currently in contact with touch surface in general

```
1  document.body.addEventListener('touchstart', function(e){
2      var touchlist = e.touches
3      for (var i=0; i<touchlist.length; i++){ // loop through all touch points currently in contact with surface
4          //do something with each Touch object (point)
5      }
6  }, false)
```

The Event object's three properties evt.changedTouches, evt.targetTouches, and evt.touches are all list objects containing a list of Touch objects, one Touch object for each touch point made. It is through a Touch object you get details about a specific touch point, such as its coordinates on the screen, its unique identifier to help you identify which touch point is which, and so on. You saw in the beginning some code that accesses a Touch object contained inside evt.changedTouches:

```
1  box1.addEventListener('touchstart', function(e){
2      var touchobj = e.changedTouches[0] // reference first touch point (ie: first finger)
3      startx = parseInt(touchobj.clientX) // get x position of touch point relative to left edge of browser
4      e.preventDefault()
5  }, false)
```

Here e.changedTouches[0] is a Touch object, with clientX being one property of the Touch object. Lets take a formal look at the Touch object now:

Touch object Property	Description
identifier	An value to help uniquely identify each touch point currently in contact with the touch surface. The value starts at 0 for the first unique touch point on the surface, 1 for the second etc. This value is maintained for each touch point until the user's finger is lifted off the surface. Lets say the user puts two fingers down on an element. Each finger at this point is assigned a unique identifier. When you move the fingers, you can use each touch point's identifier to identify which touch point is which.
screenX	The x coordinate of the touch point relative to the left edge of the user's screen.
screenY	The y coordinate of the touch point relative to the top edge of the user's screen.
clientX	The x coordinate of the touch point relative to the left edge of the viewport, **not** including scroll offsets.
clientY	The y coordinate of the touch point relative to the top edge of the viewport, **not** including scroll offsets.
pageX	The x coordinate of the touch point relative to the left edge of the viewport, including scroll offsets.
pageY	The y coordinate of the touch point relative to the top edge of the viewport, including scroll offsets.
radiusX	The radius of the ellipse which most closely defines the touching area (e.g. finger, stylus) along the x-axis.
radiusY	The radius of the ellipse which most closely defines the touching area (e.g. finger, stylus) along the y-axis.
rotationAngle	The angle (in degrees) that the ellipse described by **radiusX** and **radiusY** is rotated clockwise about its center.
force	Returns the force of the touch point in the form of an integer between 0 and 1, where 0 is no force as detected by the device, and 1, the highest.

target	The target element of the touch point; in other words, the element the touch point landed on, which may be different from the element its corresponding touch event was originally bounded to. In the following, this always returns the BODY element, while Touch.target returns the element the finger actually touched down on, which could be a DIV, a SPAN etc:
	document.body.addEventListener('touchstart', function(e){
	var touchobj = e.changedTouches[0]
	console.log(this.tagName) // returns BODY
	console.log(touchobj.target) // returns element touch point landed on
	}, false)

The properties of the Touch object you'll most frequently be accessing are those relating to coordinates, to help you determine where, and with a little Math, in what direction and how fast a touch action is performed.
Lets rewind now back to the Event object and talk a bit more about the Touches, changedTouches, and targetTouches properties, to help more clearly explain their differences:

- **Touches**: A list of all touch points currently making contact with the touch surface.
-
- **changedTouches**: A list of touch points involved in this event. For example, in a touchmove event, changedTouches contains only a list of touch points that are currently moving, whereas Touches would contain all touch points currently on the surface.
-
- **targetTouches**: A list of touch points currently making contact with the touch surface **AND** started out from the same element that is the target of this event. For example, lets say you bind the touchstart event to a DIV and place two fingers down on the surface. targetTouches will only contain information on the finger(s) placed inside the DIV, and not any outside.
-

Andrei at Stackoverflow gave a very illuminating example that clarifies the subtle differences between these three properties:

- When I put a finger down, all three lists will have the same information. It will be in **changedTouches** because putting the finger down is what caused the event
- When I put a second finger down, **touches** will have two items, one for each finger. **targetTouches** will have two items only if the finger was placed in the same node as the first finger. **changedTouches** will have the information related to the second finger, because it's what caused the event
- If I put two fingers down at exactly the same time, it's possible to have two items in **changedTouches**, one for each finger
- If I move my fingers, the only list that will change is **changedTouches** and will contain information related to as many fingers as have moved (at least one).
- When I lift a finger, it will be removed from **touches**, **targetTouches** and will appear in **changedTouches** since it's what caused the event
- Removing my last finger will leave **touches** and **targetTouches** empty, and **changedTouches** will contain information for the last finger

Moving an object using touch

Using touch to move a DIV horizontally or vertically across the screen is very simple. Take a look at the below, which moves a DIV horizontally across a track when touched and dragged:
Example (mouse simulation added for non touch devices):

Drag

```
<script>

window.addEventListener('load', function(){

    var box2 = document.getElementById('box2'),
    boxleft, // left position of moving box
    startx, // starting x coordinate of touch point
    dist = 0, // distance traveled by touch point
    touchobj = null // Touch object holder

    box2.addEventListener('touchstart', function(e){
        touchobj = e.changedTouches[0] // reference first touch point
        boxleft = parseInt(box2.style.left) // get left position of box
        startx = parseInt(touchobj.clientX) // get x coord of touch point
        e.preventDefault() // prevent default click behavior
    }, false)

    box2.addEventListener('touchmove', function(e){
        touchobj = e.changedTouches[0] // reference first touch point for this event
        var dist = parseInt(touchobj.clientX) - startx // calculate dist traveled by touch point
        // move box according to starting pos plus dist
        // with lower limit 0 and upper limit 380 so it doesn't move outside track:
        box2.style.left = ( (boxleft + dist > 380)? 380 : (boxleft + dist < 0)? 0 : boxleft + dist ) + 'px'
        e.preventDefault()
    }, false)

}, false)

</script>

<div id="track" class="track">
<div id="box2" style="left:0; top:0">Drag Me</div>
</div>
```

The outer #track DIV is a relatively positioned element, while the #box2 DIV contained inside is absolutely positioned. We get #box2 DIV's initial left position and the x coordinate of the touch point at the touchstart event. Note I'm using touchobj.clientX here; we could have easily used touchobj.pageX instead, it doesn't matter, since we're only using this property to help ascertain the relative distance traveled by touch point.
During the touchmove event, we calculate the distance traveled by the moving touch point, by getting its current x coordinate and subtracting from that the initial x coordinate. Then, to move the #box2 DIV, we add that distance to the DIV's initial left position, throwing in a lower and upper limit of 0 and 380px, so to prevent the DIV from moving outside the parent DIV. And with that our DIV box now moves with our finger!

Detecting a swipe (left, right, top or down) using touch

Swiping in touch is the act of quickly moving your finger across the touch surface in a certain direction. There is currently no "onswipe" event in JavaScript, which means it's up to us to implement one using the available touch events, plus define just when a swipe is a, well, "swipe".

Lets first define when a movement across the touch surface should be considered a swipe. There are two variables at play here- the distance traveled by the user's finger on the x or y-axis from touchstart to touchend, and, the time it took. Based on these two factors, we can decide whether that action qualifies as a swipe and in what direction.

With that said, lets put ideas into action and see how to go about detecting a swipe right (from left to right). Once we can do that, detecting swipe in the other 3 directions is pretty much identical. For this exercise we'll stipulate that a right swipe has occurred when the user has moved his finger across the touch surface a minimum of 150px **horizontally** in 200 ms or less from left to right. Furthermore, there should be no more than 100px traveled vertically, to avoid "false positives" whereby the user swipes diagonally across, which we don't want to qualify as a swipe right.

Example (mouse simulation added for non touch devices):

Swipe Me

```
<script>

window.addEventListener('load', function(){

  var touchsurface = document.getElementById('touchsurface'),
    startX,
    startY,
    dist,
    threshold = 150, //required min distance traveled to be considered swipe
    allowedTime = 200, // maximum time allowed to travel that distance
    elapsedTime,
    startTime

  function handleswipe(isrightswipe){
    if (isrightswipe)
      touchsurface.innerHTML = 'Congrats, you\'ve made a <span style="color:red">right swipe!</span>'
    else{
      touchsurface.innerHTML = 'Condition for right swipe not met yet'
    }
  }

  touchsurface.addEventListener('touchstart', function(e){
    touchsurface.innerHTML = ''
    var touchobj = e.changedTouches[0]
    dist = 0
    startX = touchobj.pageX
    startY = touchobj.pageY
    startTime = new Date().getTime() // record time when finger first makes contact with surface
    e.preventDefault()
  }, false)

  touchsurface.addEventListener('touchmove', function(e){
    e.preventDefault() // prevent scrolling when inside DIV
  }, false)

  touchsurface.addEventListener('touchend', function(e){
    var touchobj = e.changedTouches[0]
    dist = touchobj.pageX - startX // get total dist traveled by finger while in contact with surface
    elapsedTime = new Date().getTime() - startTime // get time elapsed
    // check that elapsed time is within specified, horizontal dist traveled >= threshold, and vertical dist traveled <=
100
    var swiperightBol = (elapsedTime <= allowedTime && dist >= threshold && Math.abs(touchobj.pageY - star-
tY) <= 100)
    handleswipe(swiperightBol)
```

```
45        e.preventDefault()
46    }, false)
47
48 }, false) // end window.onload
49 </script>

    <div id="touchsurface">Swipe Me</div>
```

Inside touchend, we check that the dist traveled from touchstart to touchend is a positive number above the specified threshold value (ie: 150), since in a right swipe, that dist should always be positive based on the equation used (versus negative for a left swipe). At the same time, we make sure any vertical lateral movement traveled is less than 100px to weed out diagonal swipes. Since the vertical movement can occur either above the starting touch point or below, we use Math.abs() when getting the absolute vertical dist traveled so both scenarios are covered when comparing it to our vertical threshold value of 100.

A generic swipe detecting function

Now that we got right swipe down, lets create a more generic function that detects swiping in either of the 4 directions (left, right, up, or down):

```
1   function swipedetect(el, callback){
2
3       var touchsurface = el,
4       swipedir,
5       startX,
6       startY,
7       distX,
8       distY,
9       threshold = 150, //required min distance traveled to be considered swipe
10      restraint = 100, // maximum distance allowed at the same time in perpendicular direction
11      allowedTime = 300, // maximum time allowed to travel that distance
12      elapsedTime,
13      startTime,
14      handleswipe = callback || function(swipedir){}
15
16      touchsurface.addEventListener('touchstart', function(e){
17          var touchobj = e.changedTouches[0]
18          swipedir = 'none'
19          dist = 0
20          startX = touchobj.pageX
21          startY = touchobj.pageY
22          startTime = new Date().getTime() // record time when finger first makes contact with surface
23          e.preventDefault()
24      }, false)
25
26      touchsurface.addEventListener('touchmove', function(e){
27          e.preventDefault() // prevent scrolling when inside DIV
28      }, false)
29
30      touchsurface.addEventListener('touchend', function(e){
31          var touchobj = e.changedTouches[0]
32          distX = touchobj.pageX - startX // get horizontal dist traveled by finger while in contact with surface
33          distY = touchobj.pageY - startY // get vertical dist traveled by finger while in contact with surface
34          elapsedTime = new Date().getTime() - startTime // get time elapsed
35          if (elapsedTime <= allowedTime){ // first condition for awipe met
36              if (Math.abs(distX) >= threshold && Math.abs(distY) <= restraint){ // 2nd condition for horizontal swipe
37  met
38                  swipedir = (distX < 0)? 'left' : 'right' // if dist traveled is negative, it indicates left swipe
39              }
40              else if (Math.abs(distY) >= threshold && Math.abs(distX) <= restraint){ // 2nd condition for vertical swipe
41  met
42                  swipedir = (distY < 0)? 'up' : 'down' // if dist traveled is negative, it indicates up swipe
```

```
            }
        }
        handleswipe(swipedir)
        e.preventDefault()
    }, false)
}

//USAGE:
/*
var el = document.getElementById('someel')
swipedetect(el, function(swipedir){
    swipedir contains either "none", "left", "right", "top", or "down"
    if (swipedir =='left')
        alert('You just swiped left!')
})
*/
```

swipedetect() accepts two parameters, the element to bind the touch events to, plus a function to execute when a swipe has occurred. The function parameter "swipedir" tells you the type of swipe that was just made with five possible values: "**none**", "**left**", "**right**", "**top**", or "**down**".

The below uses the swipedetect() function to show a "left", "right", "top", or "down" background image (overlaid on top of a default background image) depending on the swipe that has just occurred:

Example (mouse simulation added for non touch devices):

Swipe Me

The code used is:

```
window.addEventListener('load', function(){
    var el = document.getElementById('touchsurface2')
    var inner = document.getElementById('inner')
    var hidetimer = null
    swipedetect(el, function(swipedir){
        if (swipedir != 'none'){
            clearTimeout(hidetimer)
            var bgimage = swipedir + 'arrow.png' // naming convention is "leftarrow.png", "rightarrow.png" etc
            inner.style.background = 'transparent url(' + bgimage + ') center center no-repeat'
            hidetimer = setTimeout(function(){ // reset background image after 1 second
                inner.style.background = ''
            }, 1000)
        }
    })
}, false)
```

The HTML markup is:

```
<div id="touchsurface2">
    <div id="inner">
        Swipe Me
    </div>
</div>
```

We bind swipedetect() to "#touchsurface2", and whenever a valid swipe has occurred inside it, we change the "#inner" DIV's background image accordingly to reflect the type of swipe that has just occurred.

Monitoring touch actions at every stage, swipe image gallery

On the previous page, you saw how to detect swipes on a touch surface, and packaged that knowledge into a generic swipedetect() function:

```
swipedetect(el, function(swipedir){
    // swipedir contains either "none", "left", "right", "top", or "down"
    if (swipedir =='left')
        alert('You just swiped left!')
})
```

That's all fine and dandy, but swipedetect() is limited in that it only lets us react to after a swipe has been made, and not during, or as the user is moving his finger across the touch surface. The later is useful in applications that need to react in tandem to a touch movement across the touch surface, such as an image gallery whereby the user can drag his finger to get a preview of the next or previous slide.

A generic ontouch function

Lets set out to create a generic ontouch() function that can be used to execute custom code at every step of a touch action, from the finger's initial contact with the surface, movement across, to lifting the finger up to end it. Surprisingly it's not much different from our swipedetect() function, only that we'll be scrutinizing the touchmove event more closely this time:

```
1  function ontouch(el, callback){
2
3     var touchsurface = el,
4     dir,
5     swipeType,
6     startX,
7     startY,
8     distX,
9     distY,
10    threshold = 150, //required min distance traveled to be considered swipe
11    restraint = 100, // maximum distance allowed at the same time in perpendicular direction
12    allowedTime = 500, // maximum time allowed to travel that distance
13    elapsedTime,
14    startTime,
15    handletouch = callback || function(evt, dir, phase, swipetype, distance){}
16
17    touchsurface.addEventListener('touchstart', function(e){
18       var touchobj = e.changedTouches[0]
19       dir = 'none'
20       swipeType = 'none'
21       dist = 0
22       startX = touchobj.pageX
23       startY = touchobj.pageY
24       startTime = new Date().getTime() // record time when finger first makes contact with surface
25       handletouch(e, 'none', 'start', swipeType, 0) // fire callback function with params dir="none", phase="start", swi-
26 petype="none" etc
27       e.preventDefault()
28
29    }, false)
30
31    touchsurface.addEventListener('touchmove', function(e){
32       var touchobj = e.changedTouches[0]
33       distX = touchobj.pageX - startX // get horizontal dist traveled by finger while in contact with surface
34       distY = touchobj.pageY - startY // get vertical dist traveled by finger while in contact with surface
35       if (Math.abs(distX) > Math.abs(distY)){ // if distance traveled horizontally is greater than vertically, consider
36 this a horizontal movement
37          dir = (distX < 0)? 'left' : 'right'
38          handletouch(e, dir, 'move', swipeType, distX) // fire callback function with params dir="left|right", pha-
39 se="move", swipetype="none" etc
40       }
41       else{ // else consider this a vertical movement
42          dir = (distY < 0)? 'up' : 'down'
43          handletouch(e, dir, 'move', swipeType, distY) // fire callback function with params dir="up|down", pha-
44 se="move", swipetype="none" etc
45       }
46       e.preventDefault() // prevent scrolling when inside DIV
47    }, false)
48
49    touchsurface.addEventListener('touchend', function(e){
50       var touchobj = e.changedTouches[0]
51       elapsedTime = new Date().getTime() - startTime // get time elapsed
52       if (elapsedTime <= allowedTime){ // first condition for awipe met
```

```
53          if (Math.abs(distX) >= threshold && Math.abs(distY) <= restraint){ // 2nd condition for horizontal swipe met
54              swipeType = dir // set swipeType to either "left" or "right"
55          }
56          else if (Math.abs(distY) >= threshold && Math.abs(distX) <= restraint){ // 2nd condition for vertical swipe
57 met
58              swipeType = dir // set swipeType to either "top" or "down"
59          }
60      }
61      // Fire callback function with params dir="left|right|up|down", phase="end", swipetype=dir etc:
62      handletouch(e, dir, 'end', swipeType, (dir =='left' || dir =='right')? distX : distY)
63      e.preventDefault()
64  }, false)
65 }
66
67 // USAGE:
68 /*
69 ontouch(el, function(evt, dir, phase, swipetype, distance){
70  // evt: contains original Event object
71  // dir: contains "none", "left", "right", "top", or "down"
72  // phase: contains "start", "move", or "end"
73  // swipetype: contains "none", "left", "right", "top", or "down"
74  // distance: distance traveled either horizontally or vertically, depending on dir value

    if ( phase == 'move' && (dir =='left' || dir == 'right') )
     console.log('You are moving the finger horizontally by ' + distance)
   })
   */
```

During each phase of a touch action- "**start**", "**move**", and "**end**", we get things like the direction of the touch point, distance traveled, and at the "end" phase, whether the touch movement constituted a swipe in one of the four directions, based on the same rules as that defined inside swipedetect() we saw earlier. The following illustrates ontouch in action to show various info about a touch action over a DIV:

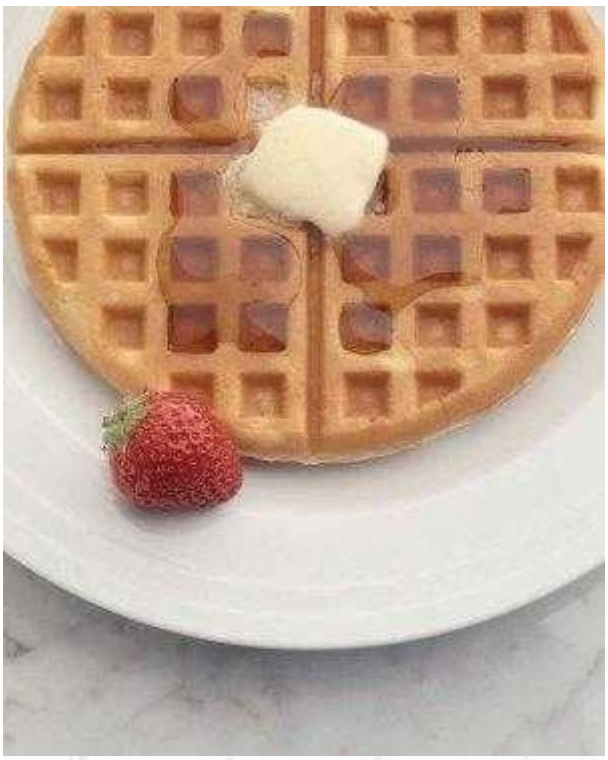
Example (mouse simulation added for non touch devices):

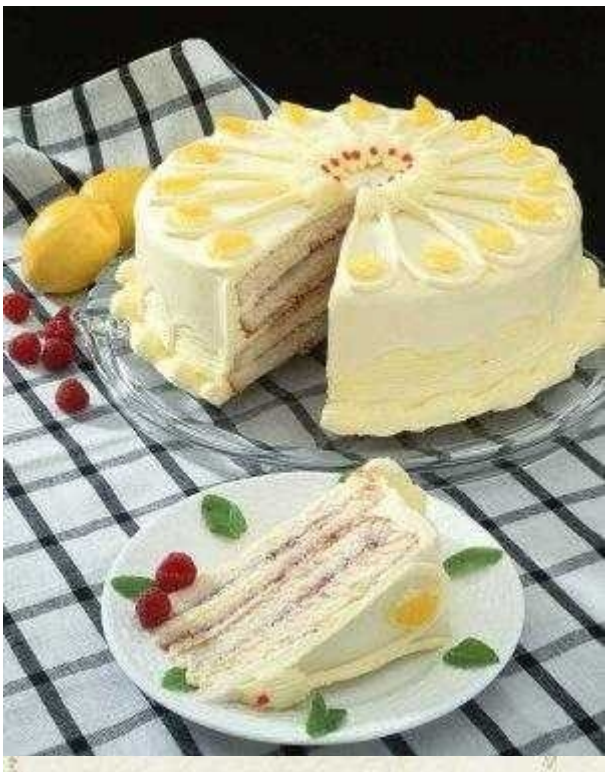
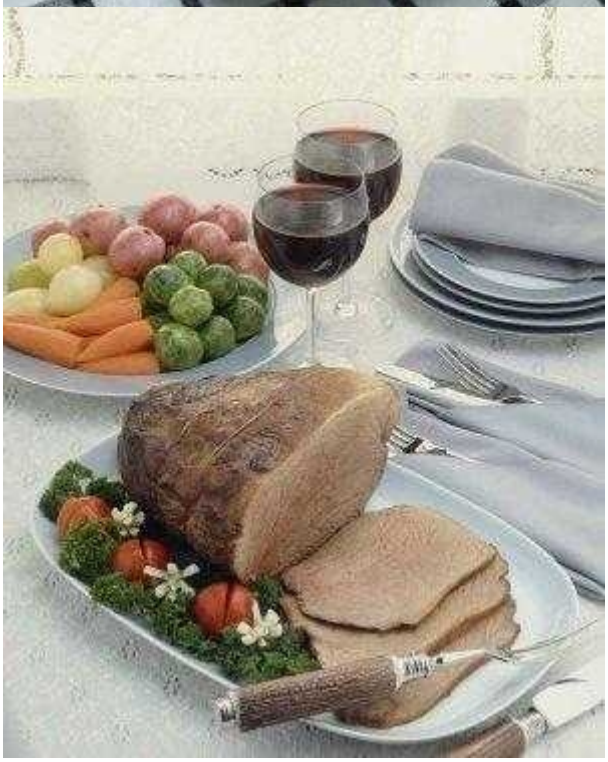
```
1  <script>
2
3  window.addEventListener('load', function(){
4      var el = document.getElementById('touchsurface2')
5      ontouch(el, function(evt, dir, phase, swipetype, distance){
6          var touchreport = ''
7          touchreport += '<b>Dir:</b> ' + dir + '<br />'
8          touchreport += '<b>Phase:</b> ' + phase + '<br />'
9          touchreport += '<b>Swipe Type:</b> ' + swipetype + '<br />'
10         touchreport += '<b>Distance:</b> ' + distance + '<br />'
11         el.innerHTML = touchreport
12     })
13 }, false)
14
15 </script>
16
17 <div id="touchsurface2">
18 </div>
```

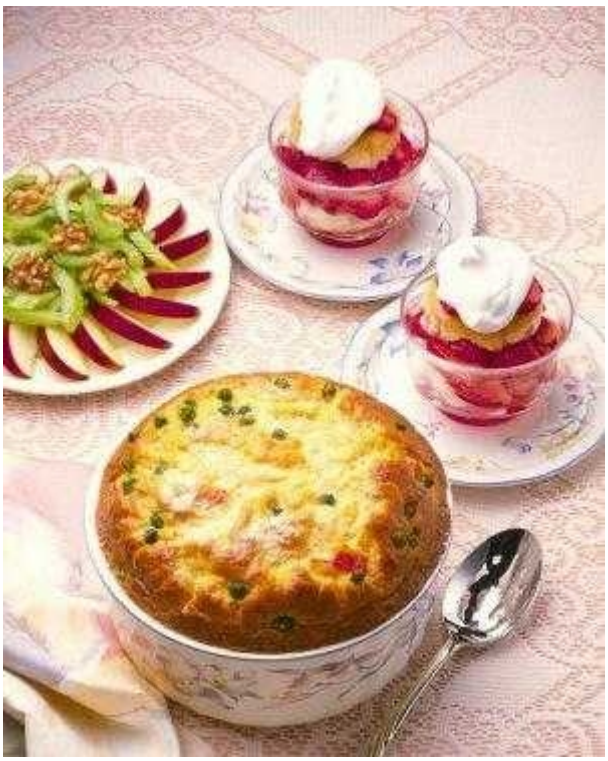
A swipe image gallery

With the ontouch function, we can use it to create a swipe image gallery that responds to not just swiping for changing a slide, but dragging to get a preview of the next slide. Take a look at the below:

Example (drag or swipe to move gallery, mouse simulation added for non touch device):

- 
- 

- 
- 

- 

```
<style>

.touchgallery{
position: relative;
overflow: hidden;
width: 350px; /* default gallery width */
height: 270px; /* default gallery height */
background: #eee;
}

.touchgallery ul{
list-style: none;
margin: 0;
padding: 0;
left: 0;
position: absolute;
-moz-transition: all 100ms ease-in-out; /* image transition. Change 100ms to desired transition duration */
-webkit-transition: all 100ms ease-in-out;
transition: all 100ms ease-in-out;
}

.touchgallery ul li{
float: left;
display: block;
width: 350px;
text-align: center;
}

.touchgallery ul li img{ /* CSS for images within gallery */
max-width: 100%; /* make each image responsive, so its native width can occupy up to 100% of gallery's width, but
not beyond */
height: auto;
}

</style>

<script>

```

```
39 window.addEventListener('load', function(){
40     var el = document.getElementById('swipegallery') // reference gallery's main DIV container
41     var gallerywidth = el.offsetWidth
42     var ul = el.getElementsByTagName('ul')[0]
43     var liscount = ul.getElementsByTagName('li').length, curindex = 0, ulLeft = 0
44     ul.style.width = gallerywidth * liscount + 'px' // set width of gallery to parent container's width * total images
45
46     ontouch(el, function(evt, dir, phase, swipetype, distance){
47       if (phase == 'start'){ // on touchstart
48         ulLeft = parseInt(ul.style.left) || 0 // initialize ulLeft var with left position of UL
49       }
50       else if (phase == 'move' && (dir =='left' || dir =='right')){ // on touchmove and if moving left or right
51         var totaldist = distance + ulLeft // calculate new left position of UL based on movement of finger
52         ul.style.left = Math.min(totaldist, (curindex+1) * gallerywidth) + 'px' // set gallery to new left position
53       }
54       else if (phase == 'end'){ // on touchend
55         if (swipetype == 'left' || swipetype == 'right'){ // if a successful left or right swipe is made
56           curindex = (swipetype == 'left')? Math.min(curindex+1, liscount-1) : Math.max(curindex-1, 0) // get new
57 index of image to show
58         }
59         ul.style.left = -curindex * gallerywidth + 'px' // move UL to show the new image
60       }
61     }) // end ontouch
62 }, false)
63
64 </script>
65
66 <div id="swipegallery" class="touchgallery">
67 <ul>
68 <li><img src="../script/script2/food1.jpg" /></li>
69 <li><img src="../script/script2/food2.jpg" /></li>
70 <li><img src="../script/script2/food3.jpg" /></li>
71 <li><img src="../script/script2/food4.jpg" /></li>
72 <li><img src="../script/script2/food5.jpg" /></li>
    </ul>
    </div>
```

Markup wise our gallery consists of a relatively positioned main DIV with an absolutely positioned UL element inside it. Each of the LI elements (images) are floated left so they appear side by side horizontally. We set the total width of the UL element to the width of the gallery DIV multiplied by the number of LI elements inside our script. With such a set up, to show the 2nd image for example, we'd simply set the left position of the UL element to (nth image -1) * width of gallery DIV (ie: 1 * 350).

Now to our ontouch function, which like a magic wand is what transforms our otherwise static series of images into a touch and swipe image gallery! We call ontouch() and pass in a reference to the gallery's DIV container as the first parameter, or the element to monitor touch actions on. The anonymous function that follows lets us respond to all touch actions taking place inside on a granular level. When the user first touches down on the gallery, we get the current UL element's left position (0 when page first loads). Then as the user moves his finger across the touch surface, if the dir is left or right we access the distance parameter to get the distance traveled horizontally. That value when added to the UL's initial left position gives us the new left position of the UL, which we then set the UL to, causing the gallery to move left or right in tandem with the user's finger movement.

When the user lifts his finger off the gallery (phase equals "end"), we use the swipetype parameter to determine if a legitimate left or right swipe has occurred (versus just a slow drag for example). If so, we increment or decrement the current index (curindex) of the gallery before moving the gallery to show the new image.
And there you have it. As you can see, implementing touch friendly UIs in JavaScript is relatively straightforward thanks to the Touch API