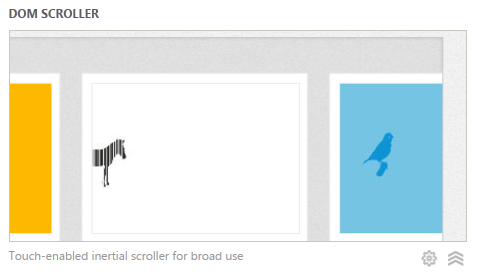
|  |
| --- |
| HTML5 Elements Cookbook |
| H5E Experiment: DOM Scroller |
| DRAFT  Published 12 July, 2011 |
|  |

|  |
| --- |
|  |
| H5E Scout Team  Windows Web Partners  Microsoft Corporation  Microsoft Confidential |

# Summary

A touch of inertia within scrollable content enables the navigator to peruse large blocks of content fast and intuitively. This effect is perfect for quickly moving through images or alphabetical lists.

**

## Scope

This document describes an experiment conducted by the H5E scout team using Clarity Consulting. Our objective is to test the limits of HTML5 solving real-world partner questions. This document assumes an existing knowledge of JavaScript and jQuery. This document does not supersede any requirements or instructions provided by the IE team.

## Keywords

Scroller, JavaScript, Touch, Hardware Acceleration, HTML5, Canvas, DOM, IE9

## Contact us

To contact us for questions or support, please email i-andods@microsoft.com. Feedback is welcome.

Contents

Summary 2

Scope 2

Keywords 2

Contact us 2

DOM Scroller Overview 4

Video 4

Experience Walkthrough 4

How do I build this? 6

Step 1: Listening for touch and mouse events 7

Step 2: Updating the sliders position within the events 7

Step 3: Adding Inertia 8

Extra 9

Complete code 10

Conclusions and Recommendations 12

Resources 12

Relevant Web sites and specifications 12

Microsoft Resources 12

Appendix A: About H5E 13

What is an HTML5 Elements Cookbook? 13

Contact us 13

Copyright and trademark information 13

Document Revision History 14

# DOM Scroller Overview

To better understand how HTML5-based features can be used to provide aspirational experiences, the H5E team has categorized these experiences into fundamental elements.

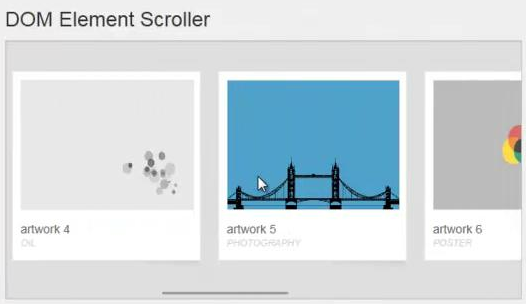
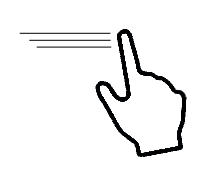
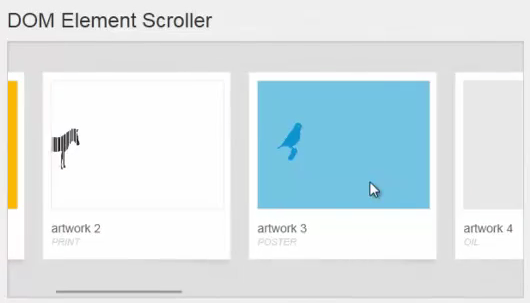
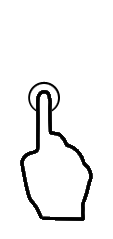
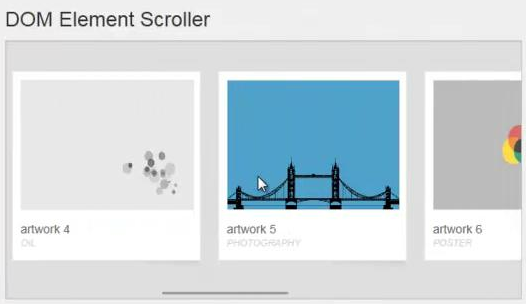
Using inertia with touch events is a core experience for current touch-based devices. Users expect the screen to behave in a way that mimics physical behavior. Using a few simple techniques, you can create the illusion of inertia-driven touch interaction that makes scrollable content feel intuitive and fast.

## Video

To see the DOM Element Scroller in action, see the following: <http://www.vimeo.com/26375685>

## Experience Walkthrough

The DOM Scroller sample creates a side-scrolling experience that is similar to native applications on touch-enabled devices. The demo application includes the following behavior:

1. The user opens the page, and the initial images are loaded into the viewport.   
   
2. The user swipes across the image using touch or holding the mouse button to move through the images. The images move in and out of the viewport at a speed and inertia that matches the touch motion.  
     
   
3. The user clicks an image to stop on a particular image.   
     
   

# How do I build this with HTML5?

There are multiple approaches to creating this experience using HTML5-related technologies. In this case, this HTML5 Experiment focuses on the following:

**Steps**

1. **Listening to events,** touch and mouse

Touch/Mouse start, Touch/Mouse move, Touch/Mouse end

1. **Changing the DOM in accordance to the change in the mouse position and the speed**

* CSS Left or marginLeft

1. **Adding inertia to give the illusion of potential energy**

**Extra**

* Removing text selection
* Adding CSS cursor grab icon
* Utilising accelerated

***NOTE****: Many of the HTML5 Experiments are still under development. Our initial target is to build prototypes that work on current HTML5-supported browsers and tablet devices, including the iPad. The experiments do not aim for full cross-browser support at this stage, but we will likely build in graceful degradation in future updates.*

To ensure that users have a similar cross-browser experience, the following table describes the compatibility of the solutions in this document:

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| HTML5 Logo**HTML5 Feature** | IE6.0 | IE7.0 | IE8.0 | IE9.0 | IEPP | Chrome11 | | Chrome12 | Safari4.x | Safari5.x | Firefox3.6 | Firefox4.x | Firefox5.x | Opera11 |
| CSS: Left, marginLeft |  |  |  |  |  |  | |  |  |  |  |  |  |  |
| CSS3: Transforms |  |  |  |  |  |  | |  |  |  |  |  |  |  |
|  | | | | | | | Full Support | | | | | | |  |
|  | | | | | | | Supported with Shim | | | | | | |  |
|  | | | | | | | No current support | | | | | | |  |

***NOTE****: In general, shims are not incorporated into the HTML5 Experiments at this stage. If a shim or polyfill is required for cross-browser support, see* [*http://browserexperiments.com*](http://browserexperiments.com) *for details on shim implementation.*

## Primary files in this solution

### Source Location

<https://github.com/molant/BrowserExperiments/tree/master/cookbook/4_DomScroller>

### Sample Location

<http://employees.claritycon.com/eklimczak/html5Cookbook/4_DomScroller/>

## Step 1: Listening for touch and mouse events

Devices which listen out for touch, i.e. touchmove, touchdown, etc., vary greatly between devices. You can see some of the compatibility tables at <http://www.quirksmode.org/mobile/tableTouch.html>.

Support for mouse events is important to make the same code work across desktop computers. Here is the basic structure of our event listeners.

(function(){

var startX, // Initial Global X coordinate of mouse/finger

$('.slider')

// Listen for the beginning of the swipe, and discover the initial position

.bind('mousedown touchstart',function(e){

…

})

// Listen out for during swipe events to update the sliders position

.bind('mousemove touchmove',function(e){

…

})

// Listen out for the release, the end of the swipe, and clear any values

.bind('mouseup touchend', function (e) {

…

});

})();

## Step 2: Updating the sliders position within the events

### Begin swipe

First record the position where the .slider is in and the position of the mouse in the global context.

$('.slider').bind('mousedown touchstart',function(e){

// Get initial offset

origX = this.offsetLeft;

// Grab the X coordinate of the event

startX = e.originalEvent.clientX;

})

### During swipe

Secondly when the swipe is occurring we need to update the position of the .slider.

$('.slider').bind('mousemove touchmove',function(e){

// If the swipe isn’t started then don’t respond

if(!startX) return;

// Change the sliders position

this.style.marginLeft = origX + (e.originalEvent - startX) + "px";

});

### Swipe End

Lastly when swipe ends we need to clear our flags which indicates that the swipe is in motion.

$('.slider').bind('mouseup touchend', function (e) {

// reset the startX

startX=null;

});

## Step 3: Adding Inertia

This continues to move our slider after the user events have stopped firing. This is to create our illusion of potential energy. To achieve this we need to determine the speed of the swipe and periodically update the position of the swipe using a timing function such as **setInterval**.

Define in the scope of our function:

var origX, // Initial X offset of the .slider

origT, // Initial rime of when the swipe

step = 50, // Number of ms to between steps

scrollInt, // Reference to the setInterval

Define on Swipe start.

// Get the coordinates

origT = (new Date).getTime();

// Get initial offset

origX = this.offsetLeft;

Bind to the Swipe end.

var dist = this.offsetLeft - origX,

time = ((new Date).getTime() - origT)/1000,

speed = dist/time,

el = this;

scrollInt = setInterval(function () {

// what distance has it travelled

dist = speed\*(step/1000)

// move the margin

el.style.marginLeft = (parseInt(el.style.marginLeft)+dist)+"px";

// decrease the speed

speed \*= 0.85;

// stop the slider when we get to a suitably slow speed

if(Math.abs(speed)<10){

clearInterval(scrollInt);

speed=0;

}

},step);

## Extra

### Selecting Text

Prevent the default behavior of selecting content whilst being dragged across the screen.

$('.slider').bind('selectstart', function () {

return false;

})

### Grab mouse cursor

Adding a grab hand to mouse cursor by attaching a CSS class to our .slider , on mousedown/touchstart. And removing it on mouseup/touchend . The CSS would be.

.slider.move,.slider.move \*{

cursor:grabbing;

cursor:-moz-grabbing;

cursor:-webkit-grabbing;

}

### Utilizing hardware acceleration

Enable webkit to use hardware acceleration. Adding the following code and calling scroll(x) to set the

offset position of the slider.

var $slider = $('.slider'), // jQuery reference to the slider

scroll = function(x){

$slider.css({marginLeft : x + "px"});

};

// Is this a webkit browser

if($.browser.webkit){

// use the CSS3 transform translate3d for hardware acceleration performance

scroll = function(x){

var v = "translate3d(" + x + ",0, 0px)";

$slider.css({

"-moz-transform": v,

"-webkit-transform": v,

"transform": v

});

}

}

## Complete code

(function(){

var startX, // Initial Global X coordinate of mouse/finger defined by touch/mouse start

origX, // Initial X offset of the .slider

lastX, // Last offset of X of .slider

origT, // Initial rime of when the swipe

step = 50, // Number of ms to between steps

scrollInt, // Reference to the setInterval

$slider = $('.slider'), // jQuery reference to the slider

coordX = function (e) {

// iOS devices expose "changedTouches"

if(e.changedTouches){

e = e.changedTouches[0];

}

return e.clientX;

},

scroll = function(x){

if(typeof(x)!=='number' || !(x<=0 || x>=0) ) return;

lastX = x;

$slider.css({marginLeft : x + "px"});

};

// Is this a webkit browser

if($.browser.webkit){

// use the CSS3 transform translate3d for hardware acceleration performance

scroll = function(x){

if(typeof(x)!=='number' || !(x<=0 || x>=0) ) return;

var v = "translate3d(" + x + "px,0, 0px)";

lastX = x;

$slider.css({

"-moz-transform": v,

"-webkit-transform": v,

"transform": v

});

}

}

$slider

// Lets prevent our mouse from selecting page elements whilst dragging over them

.bind('selectstart', function () {

return false;

})

// Initiate a touch start, and set the initial position. we shall use this to appoximate speed.

.bind('mousedown touchstart',function(e){

// Get the coordinates

origT = (new Date).getTime();

// Get initial offset

origX = lastX || this.offsetLeft;

// Add class to change the mouse cursor

$(this).addClass("move");

// prevent any further events getting fired

e.preventDefault();

// Grab the X coordinate of the event

startX = coordX(e.originalEvent);

// stop any previous scrolling

clearInterval(scrollInt);

})

// Listen out for mouse move events to store the new mouse position

.bind('mousemove touchmove', function (e) {

// has the swipe started

if(!startX) return;

e.preventDefault();

// Change the sliders position

scroll(origX + (coordX(e.originalEvent) - startX));

})

// Listen out for the release, the end of the swipe

.bind('mouseup touchend', function (e) {

// dont do anything if we dont have a startX in this scope

if (!startX) return;

// reset the startX

startX=null;

e.preventDefault();

// Remove the class which changed our cursor

$(this).removeClass("move");

/\*\*

\* Add inertia to the slider

\*/

var dist = lastX - origX,

time = ((new Date).getTime() - origT)/1000,

speed = dist/time,

el = this;

scrollInt = setInterval(function () {

// what distance has it travelled

dist = speed\*(step/1000)

// move the margin

scroll(lastX+dist);

// decrease the speed

speed \*= 0.85;

// stop the slider when we get to a suitably slow speed

if(Math.abs(speed)<10){

clearInterval(scrollInt);

speed=0;

}

},step);

});

})();

# Conclusions and Recommendations

This is a great solution for navigating though images or alphabetical data. It will work in all browsers.

# Resources

## Relevant Web sites and specifications

|  |  |
| --- | --- |
| Demo | <http://sandbox.knarly.com/js/scroll2.htm> |
| Demo | <http://employees.claritycon.com/eklimczak/html5Cookbook/4_DomScroller/> |

## Microsoft Resources

|  |  |
| --- | --- |
| H5E Primary Contact | Chewy Chong ([ChewyC](mailto:ChewyC?subject=H5E%20Experiments%20docs)) |
| H5E Development Contact | Anton Molleda Quintana ([v-anmoll](mailto:v-anmoll?subject=H5E%20Experiments%20Docs)) |

# Appendix A: About H5E

## What is an HTML5 Elements Cookbook?

Each HTML5 Elements Cookbook reflects a case study of an aspirational experience that is provided by a native or component-based application. The HTML5 Experiments that are conducted by the H5E team use HTML5 and related technologies to replicate these experiences. Our primary objective is to learn from these experiments to determine if an HTML5 alternative to component-based or native implementations is both possible, and practical. Each Cookbook provides a description of the element and technical details of the HTML5 replication of that feature. We also include recommendations on whether it makes sense to pursue this approach.

## Contact us

If you need assistance with technical solutions or have a best practice to share, please contact us by sending email to Chewy Chong ([ChewyC](mailto:chewyc?subject=H5E%20Cookbooks%20and%20Documentation)).

## Copyright and trademark information

This document is Microsoft Confidential.

Complying with all applicable copyright laws is the responsibility of the user. Without limiting the rights under copyright, no part of this document may be reproduced, stored in or introduced into a retrieval system, or transmitted in any form or by any means (electronic, mechanical, photocopying, recording, or otherwise), or for any purpose, without the express written permission of Microsoft Corporation.

 2011 Microsoft Corporation. All rights reserved.

# Document Revision History

|  |  |  |
| --- | --- | --- |
| **Reviser** | **Date** | **Revisions** |
| **i-andods** | 13 July 2011 | Initial draft |
| **v-jgeige** | 15 July 2011 | Edit |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |