Extracted from:

# Web Development Recipes Second Edition

The Pragmatic Bookshelf

Dallas, Texas • Raleigh, North Carolina

The Pragmatic Programmers

2nd Edition

# Web Development Recipes

Brian P. Hogan,
Chris Warren,
Mike Weber, and
Chris Johnson

*edited by Rebecca Gulick*

# Web Development Recipes
# Second Edition

Brian P. Hogan
Chris Warren
Mike Weber
Chris Johnson

The Pragmatic Bookshelf
Dallas, Texas • Raleigh, North Carolina

The team that produced this book includes:

Rebecca Gulick (editor)
Potomac Indexing, LLC (index)
Eileen Cohen; Cathleen Small (copyedit)
Dave Thomas (layout)
Janet Furlow (producer)
Ellie Callahan (support)

For international rights, please contact *rights@pragprog.com*.

Printed in the United States of America.
ISBN-13: 978-1-68050-056-1
Encoded using the finest acid-free high-entropy binary digits.
Book version: P1.0—July 2015

Recipe 25

# Mobile Drag and Drop

## Problem

We have a pop-up window on our website that we use to display product details. This pop-up is draggable so users can move the detail window to the side of the screen, allowing them to browse the site while the pop-up is visible. Unfortunately, we've received some feedback from users with iPads that they can't move the pop-up windows.

Drag-and-drop functionality has been an easy feature to add to websites for a while now. Various plug-ins are available that can add it with little effort, and it's not even that difficult to write from scratch. The problem with these plug-ins is that most of them don't work on mobile devices, because they respond only to events triggered by the user's mouse. We need to make our interface work for our mobile users by using some new, mobile-specific events.

## Ingredients

- jQuery
- QEDServer (for our test server)[4]

## Solution

Browsers on mobile devices like the iPad and other touch interfaces have a new set of events they listen for instead of the normal mousedown and mouseup events. Two of these new events, touchstart and touchend, are perfect substitutes.

### Layout and Style

We'll use JavaScript to handle these events, but first we need to create our markup. The page is an unordered list of products and a hidden <div> for the draggable window. We put this file in QED's public directory as drag.html:

**dragndrop/index.html**

```
<header>
  <h1>Products list</h1>
</header>
<div id='content'>
```

4. A version for this book is available at http://webdevelopmentrecipes.com/.

```
  <ul>
    <li><a href="product1.html" class="popup">
      AirPort Express Base Station
    </a></li>
    <li><a href="product1.html" class="popup">
      DVI to VGA Adapter
    </a></li>
  </ul>
</div>
<div class="popup_window draggable" style="display: none;">
  <div class="header handle">
    <div class="header_text">Product description</div>
    <div class="close">X</div>
    <div class="clear"></div>
  </div>
  <div class="body"></div>
</div>
```

We also need to make sure that the pop-up window is absolutely positioned. Here are some basic styles that we'll need:

dragndrop/style.css

```
.clear {
  clear: both;
  display: block;
  overflow: hidden;
  visibility: hidden;
  height: 0;
  width: 0;
}
.popup_window {
  border: 1px solid #000;
  width: 500px;
  height: 300px;
  box-shadow: 1px 1px 2px #555;
  position: absolute;
  top: 50px;
  left: 50px;
  background: #EEE;
  transition: box-shadow 0.5s ease;
}

ul {
  list-style: none;
  padding-left: 0;
}

.popup_window.dragging {
  box-shadow: 4px 4px 4px #555;
}
.popup_window .header {
  background: green;
  width: 100%;
  display: block;
}

.draggable .handle { cursor: move; }

.popup_window .header .header_text {
  margin: 5px;
  display: inline;
  color: #FFF;
}

.popup_window .header .close {
  float: right;
  padding: 2px 5px;
  border: 1px solid #999;
  background: red;
  color: #FFF;
  cursor: pointer;
  margin: 0;
}
.popup_window .header:after { clear: both; }
```

Additionally, we need to create an individual product page that the links will point to. Normally this page would be built on the server, but for demonstration purposes we create a single page for all of the product links. This page, which we name product1.html, also goes in QED's public directory:

**dragndrop/product1.html**

```
<h3>Product Name</h3>
<div class='product_details'>
  <missing>Need a real product page</missing>
  <p>This is a product description. Below is a list of features:</p>
  <ul>
    <li>Durable</li>
    <li>Fireproof</li>
    <li>Impenetrable</li>
    <li>Fuzzy</li>
  </ul>
</div>
```

## Basic Drag and Drop

So far our links work fine, but we want them to load the pages that they reference into the pop-up window, rather than redirecting the browser. We add the popup classes to our product links so we know which links should be loaded into the pop-up when clicked:

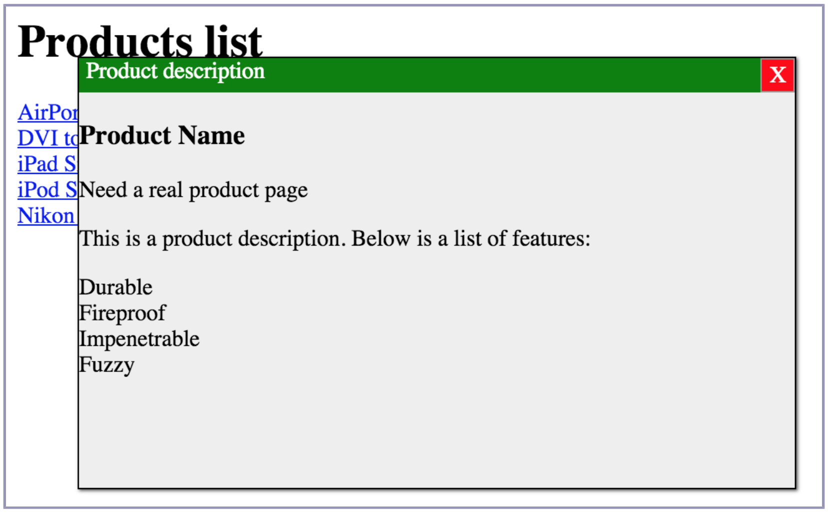
dragndrop/dragndrop.js
```
$('.popup').on('click', updatePopup);

function updatePopup(event) {
  $.get($(event.target).attr('href'), [], updatePopupContent);
  return false;
}

function updatePopupContent(data) {
  var popupWindow = $('div.popup_window');
  popupWindow.find('.body').html($(data));
  popupWindow.fadeIn();
}

$('.popup_window .close').on('click', hidePopup);
function hidePopup() {
  $(this).parents('.popup_window').fadeOut();
  return false;
}
```

These functions give us a way to hide and show the pop-up window. Everything looks great; we can load it dynamically with new data and still see most of the page. The problem is that it's in the way, as shown in the following figure:


Products list

Product description X

AirPor
DVI to
iPad S
iPod S
Nikon

Product Name

Need a real product page

This is a product description. Below is a list of features:

Durable
Fireproof
Impenetrable
Fuzzy

We currently have no way of moving this pop-up, so let's fix that by making it draggable. We'll start by making it work in desktop browsers and then apply the same logic to the touch events.

dragndrop/dragndrop.js
```
$('.draggable .handle').on('mousedown', dragPopup);
function dragPopup(event) {
  event.preventDefault();
  var handle = $(event.target);
  var draggableWindow = $(handle.parents('.draggable')[0]);
  draggableWindow.addClass('dragging');
  var cursor = event;
  var cursorOffset = {
    pageX: cursor.pageX - parseInt(draggableWindow.css('left')),
    pageY: cursor.pageY - parseInt(draggableWindow.css('top'))
  };
    $(document).mousemove(function(moveEvent) {
      observeMove(moveEvent, cursorOffset,
        moveEvent, draggableWindow);
    });
    $(document).mouseup(function(up_event) {
      unbindMovePopup(up_event, draggableWindow);
    });
}
function observeMove(event, cursorOffset, cursorPosition, draggableWindow) {
  event.preventDefault();
  var left = cursorPosition.pageX - cursorOffset.pageX;
  var top  = cursorPosition.pageY - cursorOffset.pageY;
  draggableWindow.css('left', left).css('top', top);
}
function unbindMovePopup(event, draggableWindow) {
    $(document).unbind('mousemove');
  draggableWindow.removeClass('dragging');
}
```

We start by watching for any <div> elements with a handle class in a draggable element. When the mouse is clicked and held, we call dragPopup(). This adds another observer for the mousemove event. Every time the mouse is moved, we update the position of the draggable_window. The event gives us the position of the mouse, but we need to set the position of the draggable <div>'s upper-left corner. To calculate this, we capture the offset between the initial position of the window and the position of the first click. That way, we can subtract those extra pixels from the mouse's position when moving the window in the observeMove() function.

Then, so we can finish the move event, we add an event handler for the mouseup event. When this event is triggered, we clean up the changes that we made since the mousedown event. This means we stop observing the mousemove event and remove an extra style class we added to the draggable_window.

## Adding Mobile Functionality

Thankfully, with the hard part out of the way, it is easy to adapt this approach for mobile devices. Other than the use of mouse-related events, the dragPopup() function does most of what we want. So, it should be a matter of mimicking that mouse-related code and making it act on the touch events.

First we need a way to check that the touch events are supported. If we were to call a touch-related function on a desktop, our code would break. To prevent that, we wrap our touch code in isTouchSupported() if statements:

dragndrop/dragndrop.js

```
function isTouchSupported() {
  return 'ontouchmove' in document.documentElement;
}
```

Then we add an event handler for the touchstart event alongside our handler for the mousedown event. These both trigger the dragPopup() function. Then we trigger the dragPopup function from the touchstart event:

dragndrop/dragndrop.js

```
$('.draggable .handle').on('mousedown', dragPopup);
if (isTouchSupported()) {
  $('.draggable .handle').on('touchstart', dragPopup);
}
```

Since a user can touch multiple spots, the touch event returns an array of touches. But we're focused on only one-finger movements for now, so we use the first touch in the array to determine the position of the user's finger. We then pass in this location as the cursorPosition:

dragndrop/dragndrop.js

```
function dragPopup(event) {
  event.preventDefault();
  var handle = $(event.target);
  var draggableWindow = $(handle.parents('.draggable')[0]);
  draggableWindow.addClass('dragging');
  var cursor = event;
  if (isTouchSupported()) {
    cursor = event.originalEvent.touches[0];
  }
  var cursorOffset = {
    pageX: cursor.pageX - parseInt(draggableWindow.css('left')),
    pageY: cursor.pageY - parseInt(draggableWindow.css('top'))
  };

  if (isTouchSupported()) {
    $(document).bind('touchmove', function(moveEvent) {
      var currentPosition = moveEvent.originalEvent.touches[0];
      observeMove(moveEvent, cursorOffset,
        currentPosition, draggableWindow);
    });
    $(document).bind('touchend', function(upEvent) {
      unbindMovePopup(upEvent, draggableWindow);
    });
  } else {
    $(document).mousemove(function(moveEvent) {
      observeMove(moveEvent, cursorOffset,
        moveEvent, draggableWindow);
    });
    $(document).mouseup(function(up_event) {
      unbindMovePopup(up_event, draggableWindow);
    });
  }
}
function unbindMovePopup(event, draggableWindow) {
  if (isTouchSupported()) {
    $(document).unbind('touchmove');
  } else {
    $(document).unbind('mousemove');
  }
  draggableWindow.removeClass('dragging');
}
```

Unfortunately, jQuery doesn't fully support observing touch events using the on() function, so we can't access the touches array from the jQuery event. Instead, we have to get the position of the user's finger from the original event. Now we can also mimic the mousemove behavior with the touchmove event by calling observeMove(), which remains the same. The final difference is that on the touchend event, we unbind the touchmove event, just as we did with the mouseup and mousemove events, respectively.

## Further Exploration

Now that we've seen how a single-touch event can be handled, it should be easy to figure out how to start handling multifinger gesture commands. Since the touch events return an array of touch positions, we can determine when a user has multiple fingers on the screen and where each finger is. This means we can know when users are pinching the screen, swiping side to side, or using a gesture that we invent. For more about what we can do with this API, check out HTML5 Rocks.[5]

## Also See

- Recipe 24, *Touch-Responsive Drop-Down Menus* on page ?

5. http://www.html5rocks.com/en/mobile/touch.html