

jQuery Mobile

JavaScript & Events

Lesson 1, Activity 2: JavaScript & Events

jQuery Basics

jQuery is the JavaScript framework upon which jQuery Mobile is built. Core [jQuery](#) "simplifies HTML document traversing, event handling, animating, and Ajax interactions for rapid web development" - offering you, as a developer, both great power and a much simplified API to work with (as compared to plain old JavaScript).

The most basic concept of jQuery is to "select some elements and do something with them". jQuery supports most CSS3 selectors, as well as some nonstandard selectors. (For a complete selector reference, visit the [jQuery docs](#).)

One might, for example, want to find all `div`s on a page with a given class:

```
$('div.myClass');
```

Or to find "list elements within the unordered list element with class `people` within the element with id `contents`":

```
$('#contents ul.people li');
```

Each of the examples above returns a collection - a set, with zero, one, or more DOM elements - which can be traversed using jQuery.

`$('.p.highlight').length` returns the number of paragraphs of class `highlight`. This is a good way to test for the existence of a given element, since a length of 0 means the searched-for element does not exist.

We can traverse these collections of elements and manipulate them - remove elements, set displayable-on-the-screen content, update a class or style, etc.

Here's a quick example from the [jQuery docs](#). In a page containing the following markup:

```
<ul>
<li>list item 1</li>
<li>list item 2</li>
<li class="third-item">list item 3</li>
<li>list item 4</li>
<li>list item 5</li>
</ul>
```

We can perform the following using jQuery's `next` method:

```
$('li.third-item').next().css('background-color', 'red');
```

This code says "find the immediately-following sibling [`.next`] of the list item with class `third-item` [`$('.li.third-item')`] and set its CSS `background-color` to the value `red` [`.css('background-color', 'red')`]. The result would be a red background for the list with display text "list item 4".

Element-manipulation methods like [append](#), [prepend](#), and [remove](#) offer an easy way to dynamically edit content in the DOM with jQuery.

Since the markup we use with jQuery Mobile depends heavily on `data-x` attributes, the framework makes available an easy way to reference elements like `<div data-role="footer">`: `$("div:jqmData(role='footer') ")` returns a collection of all footers in your document.

jQuery is a great tool - check out the online [documentation](#) and [tutorials](#) for more information.

Events

JavaScript is very much an event-driven language. An event fires (i.e., the user clicks on a link, the page finishes loading, the user tilts the phone, etc.) and our code answers, usually with the relevant JavaScript event handler calling some code we have written to respond to the given event.

jQuery Mobile abstracts a variety of mobile-appropriate event handlers so that we need not worry about device- or browser-specific, low-level JavaScript implementation.

Calling Event Handlers

Developers familiar with jQuery - the core jQuery library, that is, as opposed to jQuery Mobile - are used to writing code so that it runs when the page loads. Typical jQuery code might look as below - a simple example of popping up a JavaScript alert when the user clicks on a link (an `a` tag) of class `example`:

```
$(document).ready(function() {
  $('a.example').click(function() {
    alert('hello world');
  });
});
```

The `$(document).ready` portion of the code above ensures that the document (the DOM) has fully loaded.

In jQuery Mobile, we use a slightly different "wait until the page has loaded" strategy:

```
$(document).on("pageinit", function() {
  alert('hello mobile world');
});
```

Since jQuery Mobile uses Ajax to load pages into the DOM as users navigate, the DOM-ready handler (`$(document).ready()`) only executes for the first page. To execute code whenever a new page is loaded and created in jQuery Mobile, we instead bind to the `pageinit` event using `on`. (jQuery now recommends `on` in place of `bind`; as the [jQuery docs](#) state: "As of jQuery 1.7, the `.on()` method is the preferred method for attaching event handlers to a document.")

Recently, jQuery Mobile has recommended that developers use `pageinit`, rather than other events, as the state of the document to test for to ensure the DOM has fully loaded. Note that you'll still find lots of example code on the web with other code. As the [jQuery Mobile docs](#) note:

We recommend binding to this `[pageinit]` event instead of `DOM ready ()` because this will work regardless of whether the page is loaded directly or if the content is pulled into another page as part of the Ajax navigation system.

Touch

Gesture events like tapping and swiping don't play a big, if any, part on desktops - but are essential aspects of user interaction with mobile and tablet websites and apps:

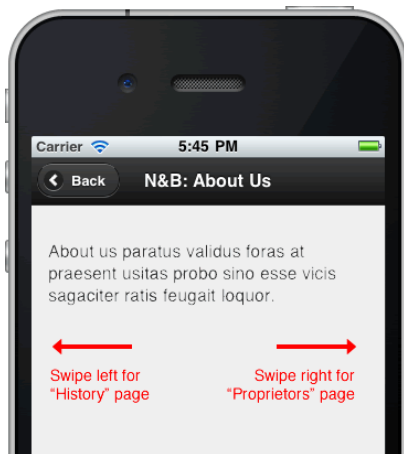
Touch Events

Event Handler	Description
tap	Triggers after a quick, complete touch event.
taphold	Triggers after a held complete touch event (close to one second).
swipe	Triggers when a horizontal drag of 30px or more (and less than 20px vertically) occurs within a 1-second duration but these can be configured: <ul style="list-style-type: none"> • <code>scrollSuppressionThreshold</code> (default: 10px) - More than this horizontal displacement, and we will suppress scrolling. • <code>durationThreshold</code> (default: 1000ms) - More time than this, and it isn't a swipe. • <code>horizontalDistanceThreshold</code> (default: 30px) - Swipe horizontal displacement must be more than this. • <code>verticalDistanceThreshold</code> (default: 75px) - Swipe vertical displacement must be less than this.
swipeleft	Triggers when a swipe event occurred moving in the left direction.
swiperight	Triggers when a swipe event occurred moving in the right direction.

See the [jQuery Mobile docs](#) for more information on the above events.

When handling events, we'll often make use of `$.mobile.changePage` - the preferred way to programmatically change to a new page. Feeding the `id` of a given page or an appropriate jQuery DOM object as the argument of the `changePage` function allows us to, for instance, redirect to a specific page when the user swipes. Let's take a look at an example.

Open [Events/Demos/touch.html](#) in a mobile browser, and in a file editor to check the code. This example offers users four pages: the home page and three interior pages - "About Us", "Proprietors", and "History". When not on the home page, users can swipe left or right to reach the next (or previous) page, "wrapping around" to reach the first or last page as appropriate:



Code Sample:

[Events/Demos/touch.html](#)

```

---- CODE OMITTED ----

<script>
  $(document).on('swipeleft', '.swipeable', function() {
    if ($(this).next('.swipeable').length == 0) {
      $.mobile.changePage($('#about'));
    } else {
      $.mobile.changePage($(this).next('.swipeable'));
    }
  });
  $(document).on('swiperight', '.swipeable', function() {
    if ($(this).prev('.swipeable').length == 0) {
      $.mobile.changePage($('#history'));
    } else {
      $.mobile.changePage($(this).prev('.swipeable'));
    }
  });
</script>
---- CODE OMITTED ----

```

```

<div data-role="page" id="about" data-add-back-btn="true" >
  <div data-role="header">
    <h2>N&B: About Us</h2>
  </div>
  <div data-role="content">
    <p>About us paratus validus foras at praesent usitas probo sino esse vicis sagaciter ratis feugait loquor.</p>
  </div>
---- C O D E   O M I T T E D ----

  <div data-role="page" id="proprietors" data-add-back-btn="true" class="swipeable">
    <div data-role="header">
      <h2>N&B: Proprietors</h2>
    </div>
    <div data-role="content">
      <p>Proprietors defui uxor tristique sudo, ex luctus aptent esse tego.</p>
    </div>
---- C O D E   O M I T T E D ----

    <div data-role="page" id="history" data-add-back-btn="true" class="swipeable">
      <div data-role="header">
        <h2>N&B: History</h2>
      </div>
      <div data-role="content">
        <p>History decet molior, saepius antehabeo comis, pala ea, ratis, duis. Tristique epulae eu in feugait, quis ad pertineo et.</p>
      </div>
---- C O D E   O M I T T E D ----

```

As with handling any event - clicking a button, mousing over a div, etc. - with jQuery, we will add some code that says "when this event [a swipe left or right] happens, perform this action". In this case, we'll change to a new page when the user swipes left or right.

We add a `<script>` block to the head of the document. We use jQuery's `on` method to handle the `swipeleft` and `swiperight` events.

We have added a class (`swipeable`) to the non-home pages, to filter the set of pages we wish to handle swiping for.

We call the jQuery `on` method (`$(document).on('swipeleft', '.swipeable', function() {})`: "for any swipe-left event performed in this document on an element of class 'swipeable', do the following".

For the `swipeleft` event, we query the presence of a next `.swipeable` DOM element - `$(this).next('.swipeable').length` - asking, in effect, if this is the last `.swipeable` element on the page. If so - that is, if the element being swiped is the last element of class `swipeable` on the page - then we change to the first (`#about`) page. If not, then we change to the next page of class `swipeable`: `$.mobile.changePage($(this).next('.swipeable'))`

We do the reverse for the `swiperight` event. Check to see if this is the first page (with class `swipeable`): if so, change to the last page; if not, change to the previous page.

In the next exercise you will add some swiping functionality to the Nan & Bob's Online site.

Lesson 1, Activity 3: Touch Events

Duration: 15 to 20 minutes.

In this exercise, you will allow users to cycle through book-detail pages by swiping left or right.

1. Open [Events/Exercises/touch.html](#) in a file editor.
2. Add class `bookdetail` to each book-detail page.
3. Write JavaScript code in the head of the document to handle `swiperight` and `swipeleft` events on elements of class `bookdetail`.
4. In the example code above, we hard coded the first and last pages into our code, changing to the first page (by referencing its `id`) when the user swiped left on the last page and changing to the last page (by referencing its `id`) when the user swiped right on the first page. Use jQuery's `first` and `last` methods to handle these cases.
5. Test your code in a mobile browser.

Solution:

Events/Solutions/touch.html

```

---- C O D E   O M I T T E D ----

<script>
$(document).on('swipeleft', '.bookdetail', function() {
    // handle the swipeleft event
    if($(this).next('.bookdetail').length == 0) {
        // if there are no more books left after this one, go to the first page
        $.mobile.changePage($('.bookdetail').first());
    } else {
        $.mobile.changePage($(this).next('.bookdetail'));
        // otherwise, go to the next page
    }
});
$(document).on('swiperight', '.bookdetail', function() {
    // handle the swipe right event
    if($(this).prev('.bookdetail').length == 0) {
        // if there are no previous pages, go to the last page
        $.mobile.changePage($('.bookdetail').last());
    } else {
        // otherwise, go to the previous page
        $.mobile.changePage($(this).prev('.bookdetail'));
    }
});
</script>
---- C O D E   O M I T T E D ----

```

We add class `bookdetail` to each book-detail page, allowing us to limit the pages that will offer users swiping interaction.

`$(document).on('swipeleft', '.bookdetail')` handles swipe-left events on any div of class `bookdetail`. We check to see if there is no next `bookdetail` with `if($(this).next('.bookdetail').length == 0)`. If yes (that is, if none exists), then we `changePage` to the first `bookdetail`; otherwise, we `changePage` to the next `bookdetail`.

Lesson 1, Activity 5: **Scroll**

We can capture the start and finish of user scrolling with the scrollstart and scrollstop events; see the [jQuery Mobile docs](#) for more information.

Scroll Events

Event Handler	Description
scrollstart	Triggers when a scroll begins. Note that iOS devices freeze DOM manipulation during scroll, queuing them to apply when the scroll finishes. We're currently investigating ways to allow DOM manipulations to apply before a scroll starts.
scrollstop	Triggers when a scroll finishes.

In this example, we use jQuery to add a class to the paragraph tags in the body copy when scrolling starts - making the text italic - and remove the class when scrolling stops:



Open [Events/Demos/scroll.html](#) in a mobile browser to check this out.

Code Sample:

[Events/Demos/scroll.html](#)

```

---- CODE OMITTED ----

        .scrolling {
            font-style:italic;
        }

---- CODE OMITTED ----

```

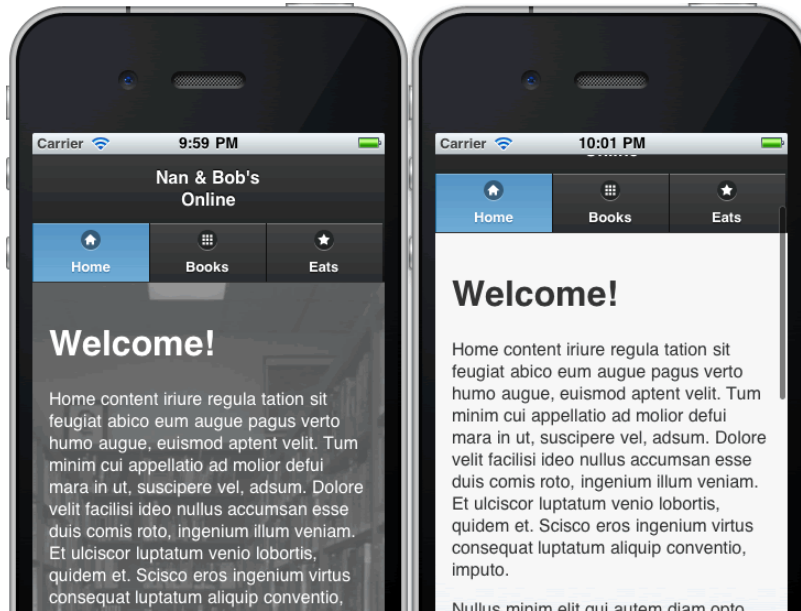
We use `$(document).on('scrollstart')` and `$(document).on('scrollstop')` to capture the scrolling start and stop events. We use jQuery's `addClass` and `removeClass` methods to add and remove class `scrolling` (which we created) to the paragraphs of greeking text.

Lesson 1, Activity 6: Scroll Events

Duration: 10 to 15 minutes.

Nan & Bob have asked us to spice up the home page of their new site with a photo background - but they think the text is hard to read when scrolling, so the photo background needs to go away when the user is scrolling the longish content.

1. Open [Events/Exercises/scroll.html](#) in a file editor.
2. Add a class and an id to the div data-role="content" in the #home page - you'll use the class for formatting (see the photo-background CSS, already written for you) and the id to reference the div when handling scroll events.
3. Write jQuery code to remove the background image when the user scrolls and replace it when the user stops scrolling.
4. The result should look as below, with the left screenshot showing the home page just after page load, and the right screenshot showing the view when scrolling:



5. Test your work from a mobile device.

Solution:

[Events/Solutions/scroll.html](#)

```

---- CODE OMITTED ----

<style type="text/css">
  .bookstorebg {
    background: #6A6A6A url('images/bg_home.jpg') no-repeat center center fixed;
    -webkit-background-size: cover;
    -moz-background-size: cover;
    -o-background-size: cover;
    background-size: cover;
    color: #fff;
    text-shadow: none;
    font-size: 16px;
  }
  h2 {
    white-space: normal !important;
  }
  ul li a h3 {
    white-space: normal !important;
  }
</style>
<script>
  $(document).on('swipeleft', '.bookdetail', function() {
    if ($(this).next('.bookdetail').length == 0) {
      $.mobile.changePage($('.bookdetail').first());
    } else {
      $.mobile.changePage($(this).next('.bookdetail'));
    }
  });
  $(document).on('swiperight', '.bookdetail', function() {
    if ($(this).prev('.bookdetail').length == 0) {
      $.mobile.changePage($('.bookdetail').last(0));
    } else {
      $.mobile.changePage($(this).prev('.bookdetail'));
    }
  });
  $(document).on('scrollstart', function() {
    $('#homecontent').removeClass('bookstorebg');
  });
  $(document).on('scrollstop', function() {
    $('#homecontent').addClass('bookstorebg');
  });
</script>
---- CODE OMITTED ----

<div data-role="page" id="home">
  <div data-role="header">
    <h2>Nan & Bob's Online</h2>

```

```

    <div data-role="navbar">
      <ul>
        <li><a href="#home" class="ui-btn-active" data-icon="home">Home</a></li>
        <li><a href="#books" data-icon="grid">Books</a></li>
        <li><a href="#eats" data-icon="star">Eats</a></li>
      </ul>
    </div>
  </div>
  <div data-role="content" class="bookstorebg" id="homecontent">
    <h1>Welcome!</h1>

---- C O D E   O M I T T E D ----

    </div>
  </div>
  <div data-role="content" class="bookstorebg" id="homecontent">
    <h1>Welcome!</h1>

---- C O D E   O M I T T E D ----

```

See Chris Coyier's [excellent article](#) (from which we fashioned the CSS for the bookstore background image in this exercise) for more info on stretching background images.

Adding class `bookstorebg` to the home page content `div` allow us to style it, initially, with the CSS class already written. We add `id=homecontent` to the home page content, too, so that we can reference the element in the event handler.

We add JavaScript rules - using `scrollstart` and `scrollstop` - to add and remove the class from the `div` in response to user scrolling.

Lesson 1, Activity 8: **Orientation**

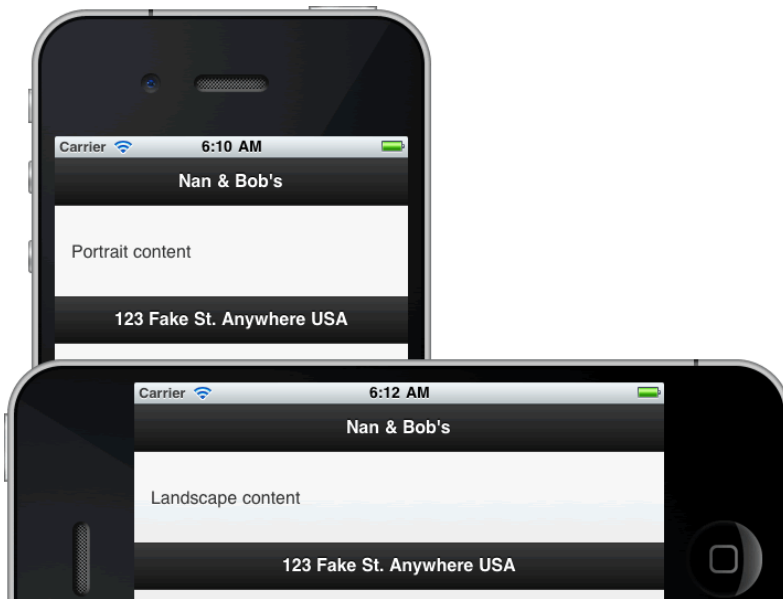
Most mobile-device screens aren't square - they're rectangular. As such, and unlike desktop or laptop devices, there's a marked difference when we tilt our phones or tablets. We can provide our users with a richer experience when we present a different view or different interactions with content presented in landscape versus portrait mode. Showing/hiding content, changing a side-by-side layout into a stacked vertical layout, and other orientation-state reactions are enhancements to your pages that your mobile users will appreciate.

The jQuery Mobile orientationchange event handler allows us to capture changes in the orientation of the device:

Orientation Events

Event Handler	Description
orientationchange	Triggers when a device orientation changes (by turning it vertically or horizontally). When bound to this event, your callback function can leverage a second argument, which contains an orientation property equal to either "portrait" or "landscape". These values are also added as classes to the HTML element, allowing you to leverage them in your CSS selectors. Note that we currently bind to the <code>resize</code> event when <code>orientationchange</code> is not natively supported, or when <code>\$.mobile.orientationChangeEnabled</code> is set to false.

In this next example, we selectively show content based on the orientation of the device. Two paragraphs live in the main content div; the paragraph with class `landscape` shows when the phone has landscape orientation, and the paragraph with class `portrait` shows when the phone has portrait orientation:



Open [Events/Demos/orientation.html](#) to test it out and to view the code.

Code Sample:

[Events/Demos/orientation.html](#)

```

---- CODE OMITTED ----

<script>
  $(window).on('orientationchange', function(event) {
    setContent(event.orientation);
  });

  $(document).on('pageinit', '#home', function() {
    setContent(window.orientation);
  });

  function setContent(orien) {
    // desktops will return undefined for window.orientation, so handle that:
    if (orien != undefined) {
      // on page load, window.orientation returns 0, 90, -90, 180, so translate to 'portrait' or 'landscape'
      if (orien == '0' || orien == '180') {
        orien = 'portrait';
      } else if (orien == '90' || orien == '-90') {
        orien = 'landscape';
      }

      //set content appropriately
      if (orien == 'portrait') {
        $('p.landscape').hide();
        $('p.portrait').show();
      } else if (orien == 'landscape') {
        $('p.landscape').show();
        $('p.portrait').hide();
      }
    }
  }
</script>
---- CODE OMITTED ----

```

We create a function, `setContent`, which when executed will check to see if the parameter passed to it (`orien`) is undefined; if not, then we must be dealing with a device for which orientation makes sense - a phone rather than a desktop.

`setContent` can be invoked in two different ways. When the page initially loads (`$(document).on('pageinit', '#home', function())`), we pass the value of `window.orientation` to `setContent` - this has a value of 90 or -90 (for landscape orientations) or 0 or 180 (for portrait orientations).

We also check for orientation-state changes once the page is live (`$(window).on('orientationchange', function(event))`), passing the value of `event.orientation` to `setContent`. This parameter has a value of "landscape" or "portrait".

If its parameter has one of the numeric values, function `setContent` translates the value into "landscape" or "portrait". It then uses jQuery's `hide()` and `show()` functions to hide or show the correct paragraphs.

Refreshing the DOM

When manipulating jQuery Mobile widgets programmatically - adding list items to an existing `listview ul`, for instance, or adding a new collapsible to the main content `div` of a page - after the `pageinit` event has fired (or later, in response to some event), we need to call `.trigger('create')` or `.trigger('refresh')` to handle the initialization of the widgets. This ensures that the widgets transform into the enhanced version. Use `.trigger('create')` to enhance raw markup that contains one or more widgets, and use `.trigger('refresh')` to update the UI for already-existing widgets that have been manipulated programmatically. See the jQuery Mobile docs for more information.

In the next exercise, you will add some orientation effects to the Nan & Bob's site.

Further Reading on Events

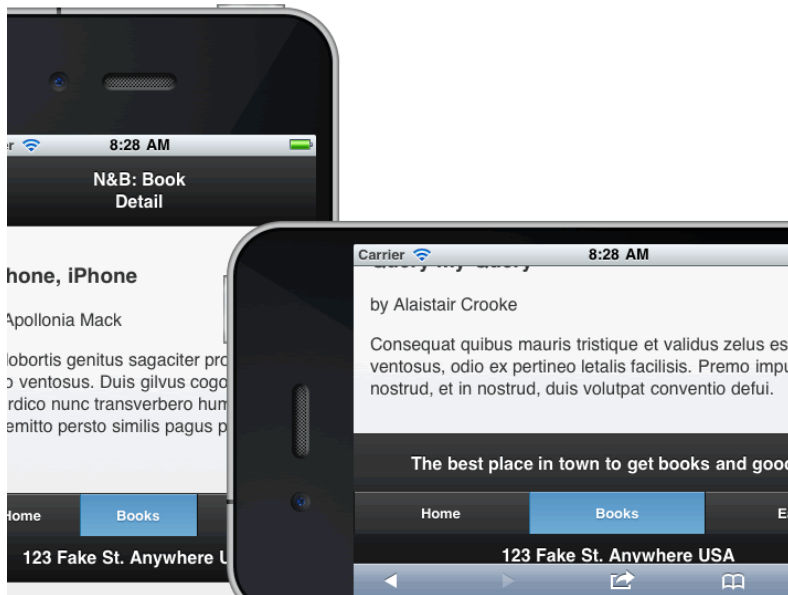
jQuery Mobile exposes many more user-action and page events; see the [jQuery Mobile docs](#) for a full list.

Lesson 1, Activity 9: Adding Orientation Events

Duration: 15 to 20 minutes.

Nan & Bob have been looking for a way to include extra marketing copy - a tagline - in their new website, but felt that the additional text didn't fit well when viewed on a phone. We suggested showing that extra copy in the footer, but only when the device is in landscape orientation.

The screenshot below shows the result, in both portrait and landscape orientation - note that the landscape orientation shows the tagline "The best place in town to get books and goodies!" on the footer, above the icons:



1. Open [Events/Exercises/orientation.html](#) in a file editor.
2. Add scripting to add the tagline to the footer on every page of the site when in landscape orientation and to remove the tagline when in portrait orientation.
3. Call `.trigger('refresh')` on the footer element after each data manipulation to update the UI appropriately.
4. Be sure to handle the case when the user changes orientation while viewing a page as well as the case when the user opens a page for the first time.
5. Check your work from a mobile device.

Solution:

[Events/Solutions/orientation.html](#)

```

---- CODE OMITTED ----

<script>

---- CODE OMITTED ----

    });
    $(window).on('orientationchange', function(event) {
        setContent(event.orientation);
    });
    $(document).on('pageinit', function() {
        setContent(window.orientation);
    });
    function setContent(orien) {
        // desktops will return undefined for window.orientation, so handle that:
        if (orien != undefined) {
            // on page load, window.orientation returns 0, 90, -90, 180, so translate to 'portrait' or 'landscape'
            if (orien == '0' || orien == '180') {
                orien = 'portrait';
            } else if (orien == '90' || orien == '-90') {
                orien = 'landscape';
            }
            if (orien == 'portrait') {
                $('p.tagline').remove();
                $("[div:jqmData(role='footer']").trigger('refresh');
            } else if (orien == 'landscape') {
                if ($('p.tagline').length == 0) {
                    $("[div:jqmData(role='footer']").prepend('<p class="tagline">The best place in town to get books and goodies!</p>').trigger('refresh');
                }
            }
        }
    }
}
</script>
---- CODE OMITTED ----

```

We detect the orientation of the device with `$(window).on('orientationchange')` (when the device changes orientation after the page has loaded) and `$(document).on('pageinit')` (when each page first loads).

Each handler calls method `setContent` which, as in our example above, translates the on-load (numeric) and orientation-change ("portrait", "landscape") orientation values into a consistent string.

setContent uses `$('.p.tagline').remove()` to remove the tagline paragraph from all footers when in portrait orientation and uses `$("#div:jqmData(role='footer')").prepend` to add the tagline paragraph to all footers when in landscape orientation, first checking to see if the tagline paragraph already exists. The method calls `.trigger('refresh')` each time.

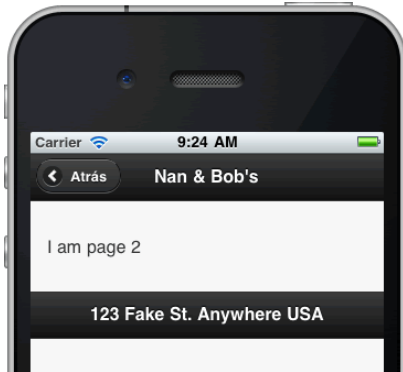
Lesson 1, Activity 11: **Configuring Defaults with JavaScript**

We can override jQuery Mobile's default values - for page transitions, for button labels, for positioning of elements, and many other aspects of our sites - via the `$.mobile` object, an object specific to jQuery Mobile. Setting a value for this object happens after the call to jQuery but before the call to the jQuery Mobile:

```
<script src="http://ajax.googleapis.com/ajax/libs/jquery/1.7.1/jquery.min.js"></script>
<script>
  $(document).on('pageinit', function() {
    $.mobile.defaultPageTransition = 'slidefade';
  });
</script>
<script src="http://code.jquery.com/mobile/1.1.0/jquery.mobile-1.1.0.min.js"></script>
```

The code above would set the default transition between all pages to `slidefade`.

Consider an example: Nan & Bob wish to support a local Spanish-language community agency's "Kids Read" event, promoting the joy of books for kids who live near the bookstore. For one week, they wish to change the display text on the back button on all pages of their site to "Atras" ("back" in Spanish). On a large site, changing the text on all buttons might be burdensome - we can instead do this once, globally, for the entire site.



Open [Events/Demos/config.html](#) to see the code and test it out.

Code Sample:

[Events/Demos/config.html](#)

```
---- C O D E   O M I T T E D ----

<script src="http://ajax.googleapis.com/ajax/libs/jquery/1.7.1/jquery.min.js"></script>
<script>
  $(document).on('pageinit', function() {
    $.mobile.page.prototype.options.backBtnText = "Atrás";
  });
</script>
<script src="http://code.jquery.com/mobile/1.1.0/jquery.mobile-1.1.0.min.js"></script>
---- C O D E   O M I T T E D ----
```

We set the back-button text for all pages with `$.mobile.page.prototype.options.backBtnText = "Atras"` - note that we place this code after the `script` tag linking jQuery but before the `script` tag linking jQuery Mobile.

You can view more details about and a list of configurable options on the [jQuery Mobile website](#).