How much interface can you fit into a single icon?

This article shows how a web app can present a navigation icon to the user at a fixed point on the screen. The second part discusses how to expand it into a feature-rich panel, with two levels of navigation options sliding out into an accordion-style interface.

26 Jun 2011

Introduction

The popularity in recent years of touch-based mobile devices has boosted the screen area available to mobile browsers, but available space is still tiny compared with desktop browsers. Mobile designers must often figure out ways to pack a great deal of functionality onto a small screen, all of which needs to be accessed with big fingertips rather than small mouse-driven pointers. We've become accustomed to a set of mobile interfaces that are patterned around this small-screen, big-finger limitation.

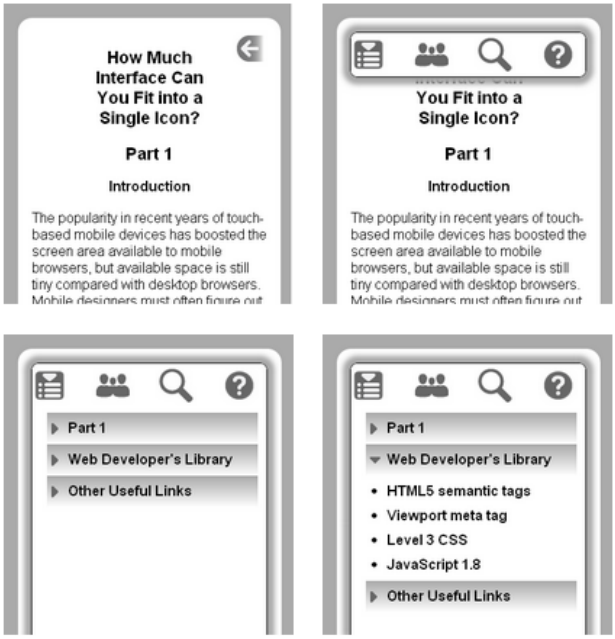
Mobile web interfaces typically feature a small number of site-wide navigation options available at the top of the page, with more detailed options available by drilling down to different screens. Users may then need *breadcrumb* hints to remind them where they are within the interface. If a page overflows with a lot of text or a long list of items, the top-level navigation options might be available only after scrolling back to the top of the page, or by getting there via a "Top of Page" link.

Now that mobile browsers support the latest set of HTML5 and CSS3 web standards, however, designers have an impressive toolkit with which to fashion complex and engaging interfaces. This article focuses on how to use CSS animations to fashion such an interface, with a goal to provide users the sense that a site's full set of functionality is available at all times, much like in a desktop interface such as the one you're currently viewing.

We'll step through an example that makes common navigation, search, and sharing options all available within a single icon, one that gently inserts itself wherever users happen to navigate. When users scroll up and down within a page, the navigation icon disappears, then slides back in after a brief delay once users stop scrolling. We'll build the CSS interface from scratch, and for the sake of clarity implement it with no jQuery or any other front-end library.

This article is divided into two parts. The first part discusses a way to use animations to present the icon to the user at a fixed point on the screen. The second part (to be published later) discusses how to expand it into a feature-rich panel, with two levels of navigation options sliding out into an accordion-style interface.

The accompanying mobile-optimized sample page shows how you might use such an interface to navigate to headings within an article—namely, the one you're reading. You can view it with WebKit-based browsers such as Google Chrome and Safari 5 on your desktop, or iOS 4.3 and Nokia devices with Browser 7.3. Here is how the interface flow appears:



The techniques discussed here can make navigation on your mobile site not only more accessible, but more engaging. The overall goal for mobile web designers is to break the habit of working around low-capability browsers.

How to Display a Navigation Icon at a Fixed point

A Basic Mobile Layout

The first thing to do is structure the page and format it flexibly for mobile browsers. The structure shown here relies on semantic HTML5 block tags, but can just as easily rely on more familiar and generic `div` tags:

```
<body>
   <section id="page">
```


(C) Copyright Nokia 2013. All rights reserved.

```
      <nav id="icon" class="scrolling">...</nav>
      <article>...</article>
    </section>
  </body>
```

Read more about HTML5 semantic tags[1]

Adding a viewport to the page's head region signals to the mobile browser that you want to adapt the content to the dimensions of the mobile screen, rather than letting text wrap to the very wide default window dimensions of desktop browsers:

```
<meta name='viewport' content='width=device-width, initial-scale=1, user-scalable=no' />
```

Read more about the viewport meta tag[2]

The `section` tag provides a common background element. It uses absolute positioning, with its `top`, `left`, and `right` edges snapped to the dimensions of the window:

```
body > section {
   position   : absolute;
   left       : 0em;
   top        : 0em;
   right      : 0em;
   background : #aaaaaa;
}
```

Its bottom edge is allowed to expand along with its nested contents. Unlike the `body` element, it also provides a reliable edge against which the enclosed `article` can offset a generous bottom margin to clear any browser chrome items:

```
body > section > article {
   padding       : 1em;
   border-radius : 1em;
   min-height    : 80%;
   margin        : 1em 1em 10em 1em;
   background    : #ffffff;
}
```

The `article` contains most of the page's content. Even when relatively empty, its `min-height` property makes it fill most of the screen.

A Fixed-Screen Icon?

The page represents the icon using HTML5's `nav` tag, which is appropriate for regions filled with navigation options. Its CSS uses absolute positioning to place the icon in the top right corner of the screen:

```
#icon {
   position : absolute;
   top      : 32px;
   right    : 32px;
   overflow : hidden;
}
```

This approach is adequate when there's not so much content on the page that the user might need to scroll downwards. In that case, the icon would scroll off the top of the screen, and users would have to return to the top of the screen to get to it. Ideally, though, you want the site's navigation elements to be available from every point within the interface, remaining stationary even as other content on the page scrolls away. Users shouldn't have to scroll or jump back to the top of the page.

CSS's `position:fixed` property should work in this case, by freezing the element to a set of screen coordinates rather than page coordinates. However, `position:fixed` is disabled on mobile browsers, due to the extra processing power it would require to scroll pages smoothly while making exceptions for individual elements.

Some JavaScript libraries offer workarounds. Cubiq's iScroll 4 and Sencha Touch address the problem by placing static elements on the screen along with elements that are made to appear scrollable in ways familiar for mobile browser users. Quick "flick" touch gestures make the element keep scrolling and eventually slow to a stop after the finger lifts from the screen. There's even a familiar bouncing effect when users reach the end of the scrolling area, which would ordinarily seem like hitting a hard wall. In this case, you would make the scrolling area fill the entire screen, then position the static icon as an overlay, but that's quite a bit of overhead for what should be a simple interface feature.

Read more about Cubiq: iScroll 4[3] and Sencha Touch: Mobile JavaScript Framework[4].

As a simple alternative, the application described here maintains the single-column page layout, and repositions the icon based on how far down the page is scrolled. Manipulating the icon's `top` position would ordinarily appear very choppy as the JavaScript struggles to catch up with what the user is doing, but as we'll see later, CSS animations can smooth those transitions. The application described here uses a simple `scroll` listener to reposition the icon based on the value of `window.scrollY`.

Building the Application

The code implementing the interface is simple. First, set up a *ui* namespace to encapsulate the application. An initialization phase makes the navigation element easily accessible:

```
var ui = new function();
ui.delay = 2000; // ...2 seconds before returning from "scrolling" state
ui.offset = 32;  // icon's initial top offset

ui.init = function() {
```


(C) Copyright Nokia 2013. All rights reserved.

```
    ui.nav = document.getElementById('icon');
};

window.onload = function() { ui.init() };
```

For now, the application assigns the icon one of two interface states, either static by default, or identified with a class named *scrolling*. To draw attention to the icon's special purpose, it slides in when the page initially loads. The initial markup defines it as *scrolling*, after which a delayed function returns it to its default state:

```
ui.isScrolling = setTimeout( ui.defaultNav, ui.delay );

ui.defaultNav = function() {
    ui.nav.className = '';
    ui.isScrolling = false;
};
```

Any `scroll` event that occurs thereafter alters the icon's `top` property directly, reapplies the *scrolling* class, and dispatches the same timer to return it to its default state:

```
ui.scrollNav = function() {
    ui.nav.style.top = ( window.scrollY + ui.offset ) + 'px';
    if ( ui.isScrolling ) return false;
    ui.isScrolling = setTimeout( ui.defaultNav, ui.delay );
    ui.nav.className = 'scrolling';
}
```

Note that the function features a test to keep from inefficiently setting off new timers while one is already in progress. After the timer expires, and while the icon slides back into the screen, the animations described in the following section smooth out transitions resulting from any subsequent scrolling.

Crafting the Animation

Once the basic class-toggling mechanism is in place, the rest of the interface can be implemented using CSS transitions. While the application mainly changes the icon's vertical position, the transitions obscure this by first sliding the icon off the screen to the right. You can do this by specifying a sequence of two transitions, with the second delayed after the first. Various `-webkit-transition` properties accept more than one comma-separated value:

```
#icon.scrolling {
   -webkit-transform                     : translateX(100px);
   -webkit-transition-property           : -webkit-transform, top;
   -webkit-transition-duration           : 0.25s, 0.0s;
   -webkit-transition-delay              : 0.0s, 0.25s;
   -webkit-transition-timing-function    : ease-in-out, linear;
}
```

The first transition, which manipulates CSS3's `translateX` transform function, moves the icon to the side out of view, quite rapidly over a quarter second. The second animation, delayed until the first one ends, shifts its `top` position to whatever new value the *scrollNav* function has set it to based on the user's scroll input. Because by now the icon has been moved out of view, its duration can be safely set to zero so that the transition occurs abruptly. If the `top` position were not specified as part of a transition, the icon would jump just as abruptly to its new location, but before moving off-screen.

NOTE: Transitions can be flexibly applied to CSS properties (such as `top`) that are assigned dynamically via JavaScript, not just those that are specified within style sheets, making them easier to integrate with the application's functionality. Also note that since the horizontal positioning is prescribed and the vertical positioning is not, and because we want to manipulate them independently, they can't both be implemented as `translateX` and `translateY` transforms. You can't transition CSS's various transform functions independently of each other, so in this case you need to use the `top` property to move the icon vertically.

Once the icon is in its *scrolling* state, it returns to its default stationary state after a two-second delay plus an additional one-second transition:

```
#icon {
   -webkit-transform                     : translateX(0);
   -webkit-transition-property           : all;
   -webkit-transition-duration           : 1s;
   -webkit-transition-delay              : 0.0s;
   -webkit-transition-timing-function    : ease-in-out;
}
```

The transition back to the stationary state is far gentler than the quicker transition to the *scrolling* state. Setting the `transition-property` to `all` manipulates the value of any remaining property that varies from the previously applied `#icon.scrolling` style sheet, in this case the `transform` property. One of the benefits of working with CSS transitions is the ability to customize their behavior depending on whether you're moving towards or away from a particular interface state.

Read more about CSS3 Transitions[5] and 2D Transforms[6].

Conclusion

So far we've seen how convenient CSS transitions are when woven into a mobile web interface. In this case, it allows the navigation icon to move to where it's needed without being too distracting. The transition back onto the screen could be made even more subtle, for example by extending its duration or by manipulating the `opacity` property to fade it in.

Continue reading the second part of the article.

References

(C) Copyright Nokia 2013. All rights reserved.

1. ↑ HTML5 semantic tags
2. ↑ Viewport meta tag
3. ↑ Cubiq: iScroll 4
4. ↑ Sencha Touch: Mobile JavaScript Framework
5. ↑ Transitions
6. ↑ 2D Transforms

(C) Copyright Nokia 2013. All rights reserved.

(C) Copyright Nokia 2013. All rights reserved.