

wexarts.org iPhone Project: Developer Documentation

Alex Ford

Wexner Center for the Arts

Introduction

In the process of creating the iPhone version of wexarts.org, we learned and absorbed many concepts, tips, tricks, conventions, code snippets, and "gotchas" related to building such a site from scratch. Our initial design concepts didn't seem suited to something pre-built like iUI, and so the student assistant developer (me) began to explore just what a project like this entails.

To coincide with the official launch of our mobile view, we decided to release a kind of "brain dump" or overview of some specific things we learned that might help someone (be they a person, a company, or a non-profit arts organization) embarking on something similar, which is what you're reading now.

Most of these concepts and methods can be found elsewhere online in various forms, and I try to link out to resources as much as possible. Sometimes, though, our code was written in a "spur of the moment" type scenario during which I failed to make note of the source of the idea or process (or maybe I actually came up with it, you never know). There's no earth shattering revelations in this document, and seasoned iPhone web developers are almost certainly familiar with everything here. This isn't written for those people.

I wrote our iPhone site from the perspective of a designer (and design major) with a strong interest in web programming, not as a computer science major, so a lot of this code could probably be refined. We welcome those refinements and encourage you to share them with us and other like-minded groups. Feel free to use any code here you like for any of your projects, and distribute this document to anyone. Please try to keep this document intact, however, if you do redistribute it.

What this document is

This is a relatively in depth write-up of many of the things I discovered while writing our iPhone site. The document covers conceptual ideas, code snippets, development process tips, some coding "gotchas", and a look at what I consider to be our more unique features. All of this stuff is believed to be accurate as of iPhone OS 3.0, and consists solely of client-side JavaScript and some CSS. Most of what I talk about here will not work on any browser *except* for Mobile Safari or the most recent version of desktop Safari (4, at the time of this writing).

What this document is *not*

This doesn't contain any code that relates specifically to the Wexner Center's content or structure, or that of any structure/content, hypothetical or otherwise. To that end, this document contains no server-side code with the exception of an example in PHP in the section of device sniffing.

This document is also *not* the only set of methods you can use to build an iPhone (or generic mobile) friendly site. This is the angle I took when I built wexarts.org/iphone, and it works pretty well for us. In fact, I expect people who read this document to have refinements and new ideas for making what's presented here better and/or more efficient.

What you gain from reading this is not a complete solution for building a "web-app" style iPhone site. This is a ground floor overview of some things you might run into over the process of building your iPhone view from scratch.

If you want to quickly and easily create something that completely emulates an iPhone's interface and behaves like a native app, something like [iUI](http://code.google.com/p/iui/) (<http://code.google.com/p/iui/>) is definitely the way to go. Many of the things I discuss here are handled by this library beautifully. Something else to look at, which I discovered mid-way through the project but that you may already know about, is Apple's own Dashcode, which is part of their developer tools (<http://www.apple.com/developer/>). It also has some great starting points and frameworks for building these "native-ish" web-app style sites.

Who this document is *for*

This is for anyone trying to build an *iPhone specific* mobile view of their website, from scratch. You will need a good, working understanding of HTML, CSS, and JavaScript (and their relationship to each other) to build something like this. If you're already familiar with the jQuery framework (<http://jquery.com>), that's great, because it's essentially the lifeblood of our methods. Personally, my JavaScript (and jQuery) skills increased probably ten-fold over the course of doing this project.

I'm not going to cover things like HTML structure or JavaScript syntax, CSS, how to include JavaScript in your pages, or anything of that nature. If you need to learn these technologies first, here are a few tutorials:

CSS: <http://www.w3schools.com/css/>

HTML: <http://www.w3schools.com/html/>

JavaScript: <http://www.w3schools.com/js/>

jQuery: <http://docs.jquery.com/Tutorials>

If you are familiar with everything except jQuery, I suggest you read the jQuery "How jQuery Works" tutorial by John Resig at the least. Also, ideally you have a MySQL or similar database solution/backend for your content that you can tap into yourself using your server-side language of choice.

Why didn't I use something prebuilt like iUI? Isn't this more work?

Yes, it is more work. What we gain from building from scratch, though, is total control over design, behavior, and format. Being an arts organization, design is essential in any project the Wexner Center does. We also have the advantage of truly understanding what it is that makes Mobile Safari tick, and as an added bonus, we *learn more* as we develop. That was the biggest draw to me.

Alright, let's get to it.

Tools & Software

There aren't many tools required to get an iPhone site off the ground; a text editor and some web server space are all that's required. There are a few things that will help you immensely along the way, though:

Safari 4 (<http://www.apple.com/safari/>) *free*: This runs on Mac & PC, and allows you to preview your iPhone site using essentially the same rendering engine as on the device itself. It's developer tools are also fantastic, no plug-ins required.

iPhone SDK / iPhone Simulator (<http://developer.apple.com/iphone/>) *free*: If you haven't already, and you're using a Mac, go sign up for a free Developer account and download the iPhone SDK. It includes, among many, many other things, Apple's own iPhone Simulator, running the real Mobile Safari, but using your Mac's network connection. You'll need this (or a real iPhone) to test things like touch events and orientation changes. *Also, many of the Apple documents I reference or link to require a developer account to view, so just go get one!*

Coda (<http://panic.com/coda>) *\$99*: This is just an excellent (though not free) web development tool. I've used it for years and it was used exclusively to develop wexarts.org/mobile. It integrates a text editor with syntax coloring, FTP and local file editing, multiple tabs, free web technologies documentation, a Unix shell, CSS editor, and the list goes on. You can use whatever editor you want, but I can't say enough good things about this one. It's Mac only, unfortunately.

You'll also of course need whatever else you normally use to develop web sites or web graphics.

Code listing conventions used here:

The only thing that should need explaining is that I have marked line wraps with `»`. You shouldn't actually break here in your code.

iPhone Safari Basics

The iPhone browser is a kind of cross between a desktop browser and a mobile browser. It is WebKit based, like desktop Safari, Google Chrome, and several other browsers. *From now on, when I refer to "Safari", I mean iPhone Safari.* The use of WebKit propels Safari to be a more advanced browser than many desktop browsers in terms of standards support, and it consistently blows away other mobile browsers, though this is beginning to change with platforms like Android and Palm Pre (which also use variants of WebKit). Some of the interesting things Safari supports are several draft HTML5 tags (including `<video>`), CSS3 (including transforms and "hardware accelerated" animation), a very fast JavaScript implementation (especially for a mobile browser, and *especially* in OS 3.0), and PNG alpha transparency.

The Viewport

Read more at: **Safari Reference Library** document: "Configuring the Viewport"

The viewport on a desktop browser and the viewport on iPhone Safari behave slightly differently. This is best described at the Apple document referenced above, so I won't try to reinvent their description here. I highly recommend you read Apple's article.

You describe how you want your viewport to behave using a `<meta>` tag in the `<head>` of an HTML document. *You should always have a viewport `<meta>` tag!*

When in doubt, try this in your `<head>` tag:

```
<meta name = "viewport" content = "user-scalable=no,  
    » width=device-width, initial-scale=1, maximum-scale=1,  
    » minimum-scale=.66667">
```

The above example is what's running on `wexarts.org/iphone` and does a good job for websites that are designed specifically for the iPhone. The `minimum-scale` seems weird, but it fixed a bug I encountered when handling orientation changes and scaling. You might not need that part, but if you're having problems in landscape mode give it a try.

Other <meta> tags

Read more at: **Safari HTML Reference** article: “Supported Meta tags”

There are a few other things you can do at the start of the document that deserve to be mentioned here:

UTF-8

```
<meta http-equiv="content-type" content="text/html; charset=utf-8">
```

This isn't really iPhone specific, but recommended for everyone. Using UTF-8 helps special characters (accents, dashes, etc.) come across correctly in your pages. This tag ended some headaches for me.

Mobile Web App Capable

```
<meta name="apple-mobile-web-app-capable" content="yes" />
```

This is an interesting one – it affects the way your site behaves when added to a user's home screen. If it's enabled, when the user taps your icon (see below), your site is opened and loaded *outside of Safari's interface*. This has a few effects. First, Safari's “chrome” (toolbars, addressbar, etc). is gone, giving you more space to work. Second, if your site isn't contained entirely in one HTML document, clicking a link will drop you out of your new “web app” and open up the full Safari browser, where it opens the new page. Third, orientation-detection is apparently broken, and JavaScript touch events don't work. The latter two effects are real bummer.

Read more at: <http://nrbd.tumblr.com/post/55066018/apple-mobile-web-app-capable-is-really-broken>

Format Detection

```
<meta name="format-detection" content="telephone=no">
```

This simply stops Safari from automatically turning phone numbers into clickable links, if for some reason you don't want that. This tag is left out on wexarts.org/iphone, which allows our “Visit” tab to be populated with “callable” phone number links.

Handling Orientation Change

Read more at: **Safari Reference Library** document: “Handling Events”

The iPhone has an accelerometer that detects the angle your user is holding the device (either in landscape or portrait). When that changes, Safari conveniently sends your <body> element an event. Set up a JavaScript event handler like this:

```
<body onorientationchange="updateOrientation();">
```

For wexarts.org/iphone, I took Apple's example JavaScript function for handling orientation events (see article referenced above) and had it add a class to my `<body>` element to "inform" my CSS of the change. I have special styles set up for many key elements on the page (mostly `width` attributes, as you'd imagine) that get automatically applied by simply adding or removing the `horizontal` class:

```
function updateOrientation()
{
    var portrait;
    var landscape;

    switch(window.orientation)
    {
        case 90:
            landscape=true;
            break;

        case -90:
            landscape=true;
            break;

        case 180:
            portrait=true;
            break;

        case 0:
            portrait=true;
            break;
    }

    //portrait
    if (portrait)
    {
        $('body').removeClass('horizontal');
    }

    //landscape
    if (landscape)
    {
        $('body').addClass('horizontal');
    }
}
```

You can also see the use of jQuery here to target the `<body>` element with `$('body').addClass()` is provided by jQuery and does just what it says. The CSS rule below is a good example of something applied by this code to bump up the text size just a bit when we go to horizontal mode in order to keep a nice line length:

```
body.horizontal p
{
    font-size:15px;
    line-height:20px;
}
```

CSS Tricks

CSS Transforms and Animation

Read more at: <http://webkit.org/blog/138/css-animation/> & <http://www.ferretarmy.com/css-animation-examples/>


Hands down, this is (to me) the coolest feature in CSS3 and the Safari browser. This technology is at the core of the way wexarts.org/iphone handles the “sliding” or “swinging” panels effect, and will become even more important as we explore new ways to provide our visually-heavy content. There are some great, concrete examples in the links above of this basic concept, so I won’t give any of these examples here. I recommend you read those posts and become familiar with how to set these up on your own. Note that these links won’t work unless you’re running Safari (either mobile or desktop). There will be more examples of this when I start talking about the panels model.

CSS Text Shadows

Read more at: <http://fredericiana.com/2009/06/10/stylish-text-with-the-css-text-shadow-property/>

Another neat (and under-supported in other browsers) CSS trick is `text-shadow`. It can be used for very gratuitous effects like huge drop shadows, but it comes in real handy when you want to do a nice, Apple-esque “bezel” or “chiseled” text effect, like wexarts.org/iphone uses on some of its controls and elsewhere:

```
text-shadow: #ddd 0px -1px 0px;
```



All upcoming exhibition events...

It’s subtle, but it’s there. The `text-shadow` property accepts first a color for the shadow, followed by the horizontal offset (in pixels), the vertical offset (in pixels) and an optional blur radius. For the chiseled effect, I just used a light grey shadow moved 1 pixel down using the vertical offset. This effectively duplicates our text behind the original, one pixel down, and in a lighter color. This is great because we *don’t have to use images* to get the effect, and it’s a good reference to the iPhone’s native UI.

1 pixel super easy gradient bars

In order to minimize bandwidth, whenever I want to use a gradient effect like the one in the button above, I create the appropriate gradient at the correct height, use Photoshop to make my image 1 pixel wide, and save it as PNG24. This almost always results in a file that’s less than a kilobyte, and using CSS we can simply repeat it across our element to give the appearance of one solid background:

```
background:url('../graphics/button_gradient.png') top left repeat-x;
```

We can take this idea even further by combining all the gradients needed into *one* PNG file, stacked on top of one another, using CSS to specify a vertical-position for the background of our element. This minimizes HTTP requests. In the words of Martha Stewart, “it’s a good thing”. I use this for the colored event area bars on wexarts.org/iphone, but it could be expanded to handle every gradient on the site (and probably should be).



```


Today ▾


div#today .sectionheader
{
  background:white url('../graphics/gradients.png') repeat-x 0px 0px;
}



Exhibitions >


div#ex .sectionheader
{
  background:white url('../graphics/gradients.png') repeat-x 0px -39px;
}



Film/Video >


div#fv .sectionheader
{
  background:white url('../graphics/gradients.png') repeat-x 0px -78px;
}



Performing Arts >


div#pa .sectionheader
{
  background:white url('../graphics/gradients.png') repeat-x 0px -117px;
}



Public Programs >

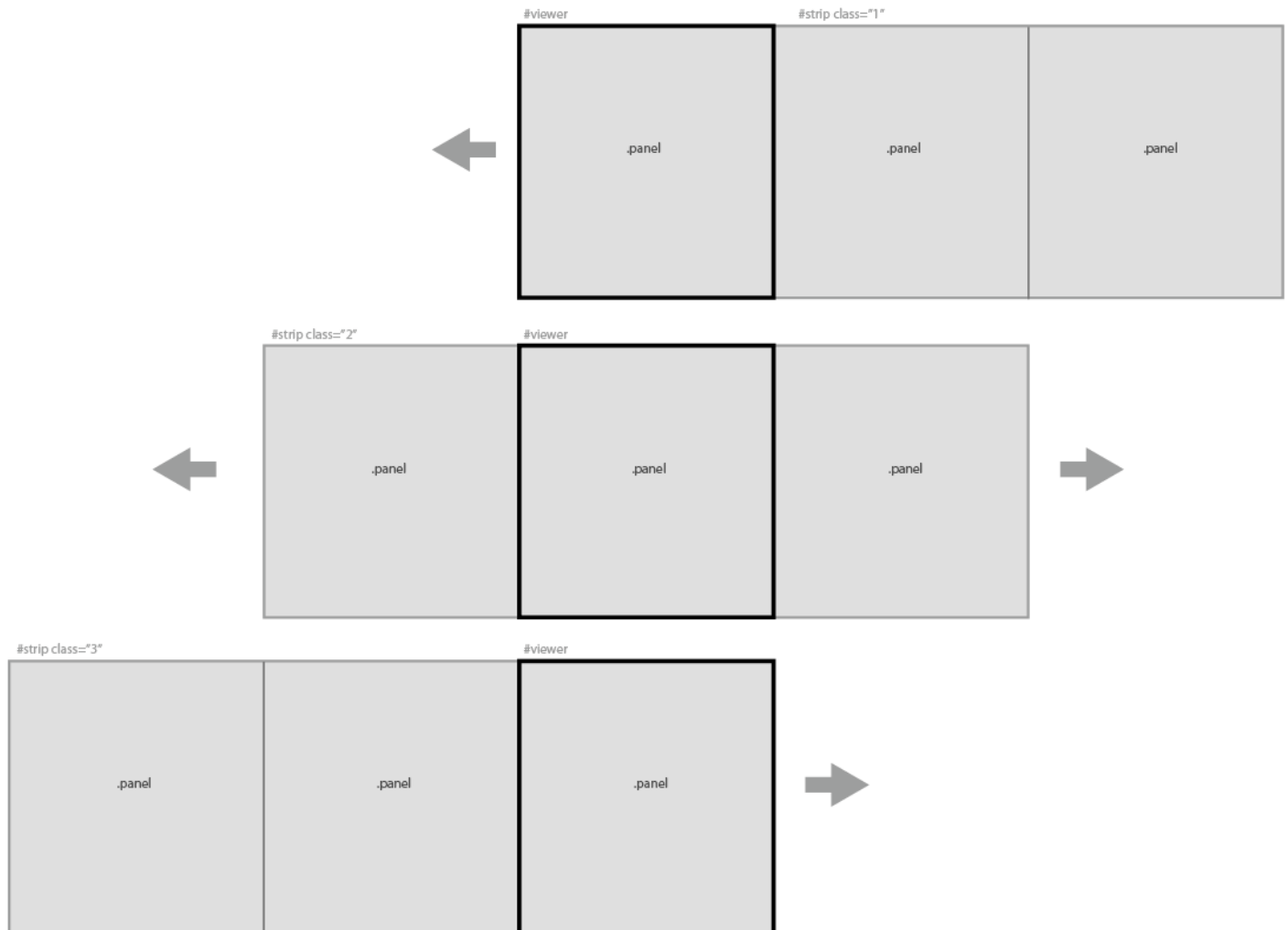

div#ed .sectionheader
{
  background:white url('../graphics/gradients.png') repeat-x 0px -156px;
}

```

This is a very good way to help load time, especially on 3G networks where latency can be a big issue, but this isn’t limited to mobile sites. *Do this!*

The Panels Model

Many iPhone native apps (Mail is a good example) utilizes what I and others call sliding “panels.” It gives a nice visual representation of “drilling down” into different layers of content or levels of detail. If you have ever used an iPhone, you know what I’m talking about. It works like this:



On the web, the basic way to do this is with two constant elements (`#viewer` and `#strip`, here), and several panel elements (`.panel` here). The `#viewer` element is specified with `position: absolute;` and `overflow-x: hidden;`, making it easy for `#strip` to be moved left and right to show and reveal the panels within it. Only the panel you’re interested in can be seen “through” the `#viewer` element. All of these elements are usually `<div>`. A strong article that helped me implement and refine this concept for use in wexarts.org/iphone: <http://css-tricks.com/create-a-slick-iphonemobile-interface-from-any-rss-feed/> (scroll down to “step” 4 to get to the slider interface stuff). Also, iUI has some good code for doing this if you peak into their files (it’s totally open source, so steal to your hearts content!).

Wexarts.org/iphone Panels Code

I've included a simplified version of the wexarts.org/iphone panels code below. I implemented CSS animation/transforms for “hardware accelerated” animation, and it really does look really smooth, especially on a 3GS. This is something that the latest iUI has as an “experimental” feature, but it works great for us, with one caveat (explained below). I'll explain some specific points after the code dump:

HTML

```
<div id='wrapper'>
  <div id='viewer'>
    <div id='strip' class='panel1'>

      <div class='panel 1 shown'>
        <!-- default calendar page view -->
      </div>

    </div>
  </div>
</div>
```

CSS

```
div#viewer
{
  overflow-x:hidden;
  width:320px;
}

div#strip
{
  position:relative;
  width:1500px;

  -webkit-transform:translateX(0px);

  -webkit-transition-property: -webkit-transform;
  -webkit-transition-duration: .3s;
  -webkit-transition-timing-function: ease-out;
}

div#strip.panel1
{
  -webkit-transform:translateX(0px);
}

div#strip.panel2
{
  -webkit-transform:translateX(-320px);
}

div#strip.panel3
{
  -webkit-transform:translateX(-640px);
}

div.panel.shown
```

```

{
    display:block;
    visibility:visible;

    height:auto;
}
div.panel
{
    width:320px;
    float:left;
    position:relative;
    height:1px;
    overflow:hidden;
}

```

JavaScript

```

var currentPanel=1;

function loadPanel(panelURL)
{
    $.ajax({
        url: panelURL,
        cache: false,

        success: function(data)
        {
            $("#strip").append(data);    //append the wrapper
        },
    });

    var oldPanel = currentPanel;    //keep track of where we are
    currentPanel = currentPanel + 1;

    scrollTo(0,1); //make sure we're at the top

    $("#strip").removeClass("panel" + String(oldPanel)); //remove #strip class
    $("#strip").addClass("panel" + String(currentPanel)); //add #strip class

    $(".panel:last").addClass(String(currentPanel));

    $(".panel." + String(currentPanel)).addClass("shown"); //show it

    setTimeout('$(".panel." + '+ String(oldPanel)+').removeClass("shown")',500);
    //hide previous panel so it's height doesn't interfere after we've slid
}

function prevPanel()
{
    if (currentPanel > 1)    //dont swing into the abyss!
    {
        var oldPanel = currentPanel;
        currentPanel = currentPanel - 1;
    }
}

```

```

scrollTo(0,1);

$("#strip").removeClass("panel" + String(oldPanel));
$("#strip").addClass("panel" + String(currentPanel));

$(".panel." + currentPanel).addClass("shown");

setTimeout('$($(".panel." + ' + oldPanel + ').remove() ',500);
//remove panel we just came from from DOM

}

```

Several things are happening here at once. First, we set up the structure I described earlier in HTML (inside a wrapper `<div>` just to keep things tidy). The classnames are important here. The class “panel1” on our `#strip` tells the CSS which position it should be in. The two classes on our only `div.panel` indicate which panel it is, and whether or not it is shown.

The CSS uses the class of `#strip` to determine it’s position and consequentially which panel is in front of `#viewer`. The `shown` class on the panel serves a special purpose. You can see in the CSS that any panels without this class have a height of 1px, this is so that longer panels that are not being viewed don’t extend `#viewer` past the viewable content, allowing the user to scroll into nothingness. The other important thing that’s happening in the CSS is the use of `-webkit-transition` and `-webkit-transform`. Hopefully you’ve read the links earlier that explain how these work —these properties are what make the animation happen. You’ll notice in our JavaScript that there is no mention of animation; just changing the `translateX()` value on an element triggers Safari to animate to the new value. The reason we don’t just use the `left` property here is because `-webkit-transform` is performance optimized on the iPhone for the smoothest possible animation.

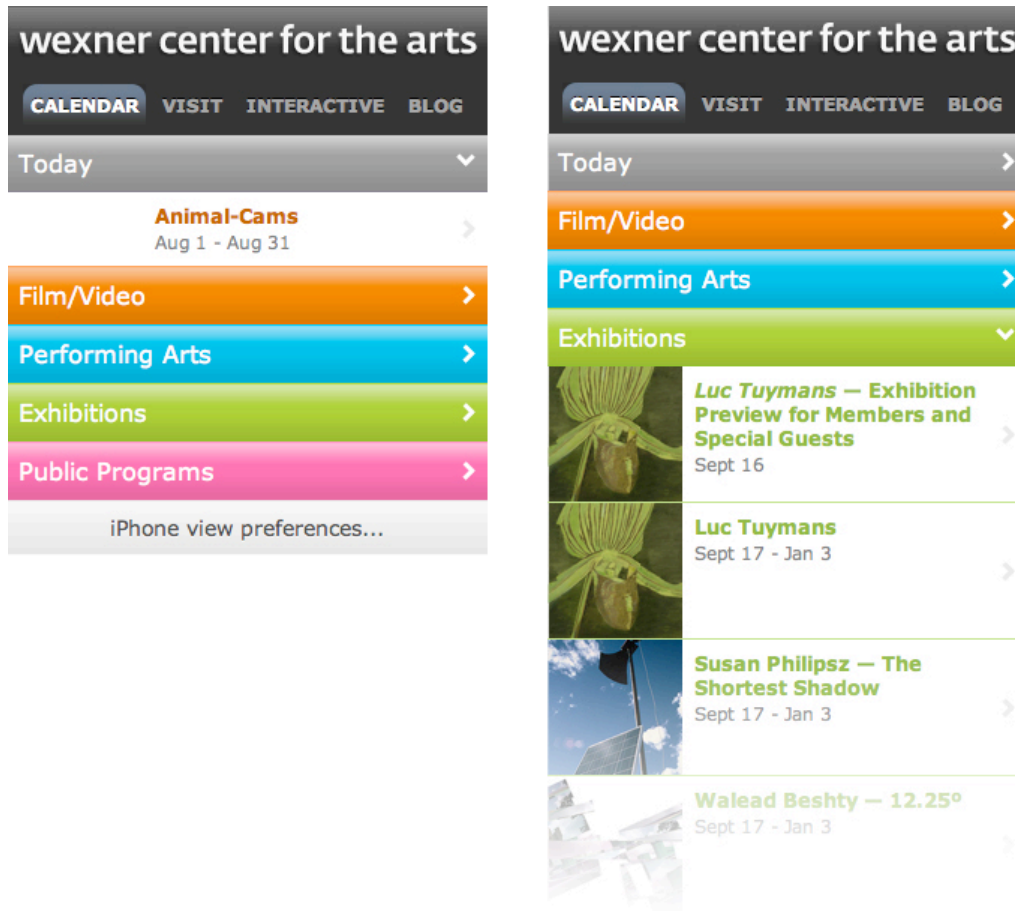
The JavaScript does all the nifty manipulation of these classes (made especially easy through jQuery). It keeps track of the current panel through a global variable, and adds and removes classes to the various panels and the `#strip` element to make all this work. There’s also an example of using jQuery for AJAX. If you’re not familiar with it, AJAX is a method for making requests to a server from within JavaScript. This allows us to dynamically load in content (say, an “Event Page” panel) without having to reload the entire page. So, if you called `loadPanel('/panels/event.php?eventId=1029')`; for example, this function is going to load that file and inject it into the DOM. The `‘/panels/event.php’` file (remember, this is just an example; that file could be anything) is where the hook to your own CMS or database goes. In this case, the file you are loading returns one element, a `<div>` with class `panel`. `scrollTo(0,1)`; attempts to keep Safari scrolled to the top of the page, as well as hide the address bar, as you go through panels. However, it’s a little unpredictable as to how often the address bar disappears with this method.

Implementing the panels model is a hairy process. You are going to run into issues, especially when it comes to things like `<video>` tags and Safari's built in functionality for automatically turning YouTube videos into links to the iPhone's own YouTube app. When using a CSS transform on their container element, these elements simply disappear on mobile Safari, and sometimes, desktop Safari as well. I haven't found a solution.

wexarts.org/iphone uses a workaround: a special site-wide embed code for videos that generates a YouTube link on iPhone using an image, and an embed on desktop browsers. This embed code is going to be improved upon in the future as we begin to support our upcoming video portal and the `<video>` tag on iPhone.

Another problem I came across was the way my implementation of panels worked when the device's orientation (and therefore, page width) changed. The way I ended up handling this was using hard coded pixel values for the width of the relevant `<div>`s and their CSS transform values, which get updated via CSS when the orientation changes (see the section on that earlier in this document). There are a number of options for getting this done in a cleaner way, mostly involving setting the CSS transform values directly from JavaScript. Hopefully, I will make progress in this area in future versions of wexarts.org/iphone, in which case I'll update this document.

Expanding/Contracting Categories



To keep iPhone users from scrolling through a stupidly long page on our calendar and about pages, we decided on using expanding and contracting “sections” or “categories” that drop open or closed when a user touches them. This is pretty easy to implement with jQuery, so let’s get straight to the code.

HTML

```
<!--today section-->
<div id='today' class='section'>
  <div id='today_header' section='today' class='sectionheader clickable'>Today
    <span class='hideshow'><img src='graphics/open_arrow.png' /></span>
  </div>

  <div id='today_content' class='sectioncontent sectionopen'>
    <?php ...load content ...?>
  </div>
</div>
<!-- end today section-->
```

...continued on next page

```

<!--film video section-->
<div id='fv' class='section'>
  <div id='fv_header' section='fv' class='sectionheader clickable'>Film/Video
    <span class='hideshow'><img src='graphics/closed_arrow.png' /></span>
  </div>
  <div id='fv_content' class='sectioncontent'>
    <?php ...load content ...?>
  </div>
</div>
<!--end film video section-->

```

(There is one of these blocks of code for each category.)

CSS

```

div.sectioncontent
{
    display:none;
}

div.sectioncontent.sectionopen
{
    display:block;
}

div.sectionheader
{
    height:29px;
    padding-left:7px;
    padding-top:10px;
    /* these rules don't really apply to this example
       in general, they are just for appearance */
}

```

JavaScript

```

function switchSectionHeader(section_id)
{
    if (!$('#' + section_id + " .sectioncontent").hasClass('sectionopen'))
        //currently closed?
        {
            //close others
            $('.sectioncontent').slideUp(250).removeClass('sectionopen');
            $('span.hideshow img').attr("src", "graphics/closed_arrow.png");

            //open this one
            $('#' + section_id + "
            .sectioncontent").slideDown(250).addClass('sectionopen');

            $('#' + section_id + " span.hideshow img").attr("src", »
            "graphics/open_arrow.png");

            //adjust view to currently open
            setTimeout(function(){$('body, html').scrollTop($('#' + section_id + "
            » .sectioncontent").scrollTop());},300);
        }
}

```

```

else //currently open?
{
    //close it
    $('#' + section_id + " »
    .sectioncontent").slideUp(250).removeClass('sectionopen');

    $('#' + section_id + " span.hideshow img").attr("src", »
    "graphics/closed_arrow.png");
}
}

$(document).ready(function ()
{
    $(".sectionheader.clickable").click(function(){
        switchSectionHeader($(this).attr('section'))
    });
}

```

The CSS/HTML is pretty minimal, all it does is make sure panels are hidden by default, except for the one we specify by giving it the class `.sectionopen`. I also showed some example rules for setting up the “button” element.

The JavaScript is where it gets more fun. In `$(document).ready()`, I set up an event handler for any `.sectionheader` element which I specify as `.clickable` (sometimes these headers are used for other things). The function it calls checks to see if the section that was “clicked” (really, touched), is open (by tracking the `sectionopen` class) and handles it appropriately. If the section in question is closed, we first close any other open sections on the page before opening the one we want. I’m using jQuery’s built in `slideUp()` and `slideDown()` functions to handle the animation. There is an opportunity to use CSS transitions here, but it’s too buggy for me to release for now. I also attempt to scroll the user’s view up to the one they just opened, because sometimes with longer sections you can end up in the middle of one after the switch. iPhone’s support of `scrollTop` is still a little buggy (you can’t animate it, for example), so this can be erratic at times. The other thing that happens here is switching the little arrow graphic on the headers to help reflect their state to the user.

Device Sniffing

For wexarts.org, we wanted to redirect iPhone users visiting an area, series, event, or the homepage of wexarts.org to our new mobile site, unless they had specified in the past that they prefer the desktop version. Similarly, we needed to redirect users of anything **but** an iPhone *away* from wexarts.org/iphone and into wexarts.org/mobile (the generic mobile view). Note that the example below does not redirect users of say, Opera mini, to the mobile view by default from wexarts.org. This is the only example in this document that uses server side scripting, because we want this redirect to be totally transparent to the end user. In the end, there's a combination of PHP, a little JavaScript, and cookies used to get this done.

PHP for <http://wexarts.org/>

This code is in our header files and decides if and how an iPhone or iPod touch user should be redirected:

```
$iphoneURL = "http://wexarts.org/iphone/";
$iphone = strpos($_SERVER['HTTP_USER_AGENT'], "iPhone");
$itouch = strpos($_SERVER['HTTP_USER_AGENT'], "iPod");

if ($iphone !=false || $itouch!=false)
    $iphoneVisitor=true;

if ($iphoneVisitor)
{    //using an iPhone?

    if (isset($_COOKIE['iphonechoice']))
    {    //we've been here before?

        if ($_COOKIE['iphonechoice']=='iphoneview')
            $iphoneRedirect=true;    //they want iPhone view
        else
            $iphoneRedirect=false;    //keep us here
    }
    else
    {    //or, first time iPhone visitor

        $iphoneRedirect=true;    //let them see the iPhone site

        setcookie("iphonechoice","iphoneview",time()+60*60*24*365,"/",".wexarts.org");

    }

    ///////iphone redirects
    //are we on an event page?
    if($_GET['eventid'])
        $iphoneRedirectURL = "$iphoneURL?eventid=".$_GET['eventid'];

    //what about a series page?
    if($_GET['seriesid'])
        $iphoneRedirectURL = "$iphoneURL?seriesid=".$_GET['seriesid'];
```

```

//are we on an area page?
if ($_SERVER['REQUEST_URI']=='/ex/')
    $iphoneRedirectURL = "$iphoneURL?area=ex";

if ($_SERVER['REQUEST_URI']=='/fv/')
    $iphoneRedirectURL = "$iphoneURL?area=fv";

if ($_SERVER['REQUEST_URI']=='/ed/')
    $iphoneRedirectURL = "$iphoneURL?area=ed";

if ($_SERVER['REQUEST_URI']=='/pa/')
    $iphoneRedirectURL = "$iphoneURL?area=pa";

//cal or index page?
if ($_SERVER['REQUEST_URI']=='/cal/' || $_SERVER['REQUEST_URI']=='/' ||
$_SERVER['REQUEST_URI']=='/index.php')
    $iphoneRedirectURL = $iphoneURL;

//if we're go for redirect, do it.
if ($iphoneRedirect && $iphoneRedirectURL) header("Location: ".$iphoneRedirectURL);

/////decide if we should alert the user to the existence of an iPhone version/////
$iphoneHidePrompt = $_COOKIE['iphonehideprompt'];

if ($iphoneRedirectURL && $iphoneHidePrompt!='yes')
    $iphoneAlert=true;
}

```

It's pretty straightforward. I won't get into PHP syntax, but it's similar enough to JavaScript ("C-Style") that you should be able to follow if you're used to something else. The first thing we do is check for the existence of the string "iPhone" or "iPod" in the user agent string. Every browser sends information about itself and the environment it's running in with each request to a web server. If either of these words exists in this string, it's safe enough for our purposes to assume this is a request from an iPhone (or iPod touch).

Once we've established that we're dealing with an iPhone, we check to see if our "iphonechoice" cookie is set. If it's not (first time iPhone visitor), or if it's set to "iphoneview", we set the variable `$iphoneRedirect` as `true` to be used in our logic further down. If it's set to anything else, this variable gets set to `false`.

The next block of code decides if this page has an iPhone equivalent, and if so sets `$iphoneRedirectURL` accordingly. Then, PHP checks to see if A) This is a request from an iPhone and the user has not chosen to see the desktop view (`$iphoneRedirect` set to `true`) and B) This page has an iPhone equivalent (`$iphoneRedirectURL` is not `null`). If both of these conditions are met, we use `header()` to redirect to the equivalent page. All of this happens before the iPhone user even knew what was coming to them.

If the user has specified that they don't want to see iPhone versions of applicable pages, but this opportunity does exist for this page, we set a flag variable to show an alert that they can see this in an iPhone view. Clicking no once on this sets a cookie (that expires at the end of the session) to hide this prompt as well. This is done in JavaScript:

```
<?php if ($iphoneAlert) { ?>
<script type="text/javascript">
    $(document).ready(function(){
        var redirect = confirm("View iPhone version of this page?")

        if(redirect)
        {
            window.location="<?=$iphoneRedirectURL?>";
        }
        else
        {
            var options = { domain:'.wexarts.org', path: '/', expires: 0 };
            $.cookie('iphonehideprompt', 'yes', options);
        }

    });
</script>
<?php } ?>
```

Note that this alert is only shown to iPhone users who have opted out of automatic redirects, are on a page that has an iPhone equivalent, and haven't said no to this yet this session. This makes use of the excellent jQuery cookie plug in, found here: <http://plugins.jquery.com/project/cookie>. That takes care of the desktop site, but what happens behind the scenes at wexarts.org/iphone/?

PHP in <http://wexarts.org/iphone/>

There's some user agent and cookie reading going on here, too:

```
$genericURL="http://wexarts.org/mobile/";

$iphone = strpos($_SERVER['HTTP_USER_AGENT'], "iPhone");
$itouch = strpos($_SERVER['HTTP_USER_AGENT'], "iPod");

if ($iphone==false && $itouch==false)
    header("Location:$genericURL");

if (!isset($_COOKIE['iphonechoice'])) //dont have a choice for iphone?
    //default to show iphone view
    setcookie("iphonechoice", "iphoneview", time()+60*60*24*365, "/", ".wexarts.org");
```

If we're not dealing with an iPhone, we need to redirect to the generic mobile site so these users can still access content. Also, if the user doesn't have an iPhone preference yet, (they may have typed <http://wexarts.org/iphone/> into Safari directly after hearing about it, for example) we go ahead and set that

cookie. By the way, you can't really set cookies to *never* expire as far as I know, so I just set them for a year as an effectively "permanent" preference.

For the "iPhone view preferences" on wexarts.org/iphone, I implemented a quick "toggle" switch in jQuery that just sets the "iphonechoice" cookie when activated. I won't go into that here, consider it homework!

Conclusion

Hopefully you came away from this document with a few tricks up your sleeve for iPhone development. I tried to be as clear as possible, but I'm not a tech writer, so if you have any questions or need some clarification, email me and I'll see what I can do. That being said, Google and Apple's Developer Connection website taught me everything I know about iPhone development. Chances are you can find the answer to your question somewhere. I tried to focus this document on things I couldn't find explained clearly to me. Thanks for reading, and feel free to share your feedback and suggestions.

Alex Ford

Student Assistant, Wexner Center for the Arts

Columbus, OH

aford@wexarts.org

This is the first version of this file, released publicly August 24, 2009.

Thanks to the many internal test group iPhone users at the Wexner Center for the Arts, and also to Robert Duffy, Jerry Dannemiller, and Chris Jones for guidance throughout the project. When in Columbus, Ohio, visit the Wexner Center for the Arts, 1871 N. High St.