

slide 1

hello everyone. i will be presenting on jQuery Mobile and Progressive Enhancement with HTML5. I plan to give a brief overview of the project and dive into the nuts and bolts of the jQuery Mobile framework and how it can be used to develop and deliver a mobile web application and then hopefully go into some tools and techniques to deploy a jQuery Mobile application as a native application to your device.

slide 2

Now, when i say that jQuery Mobile provides Progressive Enhancement with HTML5, truthfully i mean that it built on the foundation of semantic HTML5 and uses CSS3 for various things like media-queries, animations and shadows and it does this enhancement to the markup using JavaScript to define and enhance your web page, with graceful degradation for browsers that don't support the HTML5 doctype or JavaScript or are considered less-capable by the framework developers.

It does not however, utilize the new Dolphon-Duck-Knife, or DDK trifacta of the HTML5 specification. I mean, i shouldn't say never. Just like the jQuery Mobile framework doesn't provide any enhancement to the new media elements - there is no beautifying of controls or such - the framework doesn't address how to incorporate the DDK in an RIA. Which doesn't mean you can't, I don't want to box you in creatively. You'll just have to write your own scripts.

slide 3

Just a quick rundown of who I am. My name is Todd Anderson. I am a senior software engineer for Infrared5 up in boston. I am actually originally from New Jersey. Hightstown; exit 8. Even though i am up in Boston now, i still consider New York THE city. Love this town. And i'd like to thank Elad and Jose for having me. I also have some contact info up which i will show again at the end and have had the great pleasure of being a co-author on several books about the flash platform, specifically AIR and Flex, including the Cookbooks from O'Reilly.

So i would say my day job is basically architecting and developing rather large enterprise level Flex applications. I started looking into jQuery Mobile around november of last year. I was excited about the Burrito and Hero drops from Adobe and got inspired from a couple sessions at RIA Unleashed up in Boston and pretty much dove head first in in playing around with flex on mobile, even wrote a couple blog posts about it. But once i pushed it to a device, my heart sank. The performance was terrible. It is leaps and bounds above what it was, now - but at the time, late last year, there was no way I could justify to a client that Flash or Flex was a viable solution for their mobile application.

Like i said, now, it is a different story and i would definitely put it on the table. Adobe have done an amazing job since then, but at the time - last November - jQuery Mobile

had just released their second alpha and i decided to take a look. I think it is a great framework and immediately fell and love and wrote an 8 part series about using jQuery Mobile and CouchDB on my blog.

slide 4

So what is jQuery Mobile? jQuery Mobile is a navigation and UI enhancement framework being developed by the Filament Group. Now i should state that it is not necessarily an application framework per se. Meaning that it does not provide an MVC or MVVM architecture in which you are hooking up views to controllers and binding to data. You can add separate libraries like JavaScriptMVC, Joost, Backbone, KnockOut, to do that - but jQuery Mobile really just provides a navigational framework and enhancement API.

Its current state is at alpha 4.1 and is built off of jQuery and a part of jQuery UI - specifically the Widget plugin. We'll get into more depth about what a widget is, but it is basically an element decorator. Its cross platform, meaning that it targets the various browser flavors on your machine - tablet, desktop, smartphone, what have you - and they are developing against what they consider A-grade browsers.

slide 5

Much like Yahoo's YUI grade A browser chart for desktop, jQuery Mobile have a chart up for the mobile browsers on different platforms. The A stands for browser that at the very least support media-queries and the jQuery Mobile team considers to have the capabilities to support the framework. The B denotes a browser that does not have enough market share to devote a significant amount of time for testing, but could support the framework. The C denotes that the browser does not at the very least support media-queries and that the framework will fall back to plain old HTML and CSS. Which is a great approach to take and is basically the principal of Progressive Enhancement.

slide 6

In understanding Progressive Enhancement, the mark-up is the king. Meaning that if at the very least, your content or information should be valid and be seen by all browsers, indexable and searchable. And the mark-up itself should not contain any event handling or style and class declarations. Just good clean valid HTML markup. Once we are sure that everyone can at least access the same information in the same context, then we can spruce it up a little. jQuery Mobile utilizes the new data attributes of the HTML5 specification to store "roles" and assign decorating attributes and uses some CSS3, like media-queries, text-shadows, attribute selectors. And being built off of jQuery, it adheres to the unobtrusiveness of that library which basically means if the browser doesn't support JavaScript or the user has JavaScript turned off, they will still be able to access all information needed. It should not necessary to have JavaScript to view your content - its only there to enrich the experience.

Because i am a visual learner i thought i would give a quick visual explanation of Progressive Enhancement which is often considered the S-W-S approach to web application development.

slide 7

So first you start with your mark-up: the content. Here you have a waterfall, people have come to your site to find out about a waterfall. All the information is there. But say you, as a developer want to spruce the place up a bit, make it more inviting.

slide 8

We've added our CSS styling now. Thrown in a sandwich. Its welcoming, people will now stay a little longer on the page. And then to give it a little pizzazz and make sure they keep coming back...

slide 9

Throw in the Selleck or the JavaScript. And there you have your Selleck-Waterfall-Sandwich approach to web development using Progressive Enhancement. And that is the principal that the jQuery Mobile framework is built upon; with consideration that if your browser is not considered capable of grade-C, users will still be able to find information about the waterfall.

In all seriousness, a presentation architect by the name of Nicholas C. Zakas gave a presentation on Progressive Enhancement earlier this year and included a slide that had a quote by the comedian Mitch Hedberg and it said:

“An escalator can never break; it can only become stairs.”

Which i think fits the principal quite nicely. The content - the ability to go from A to B will always be available. An escalator just has enhanced the presentation with some rubber handles and grating for the steps, and a little motion or behaviour to get you where you need to go faster. But in the end, if it all breaks down, you still have the ability to get to the next level.

slide 10

How the framework works in bringing this Progressive Enhancement is through the user of what is considered “data roles”.

The HTML5 spec defines a new data attribute that is considered valid and will be passed over by browser parsing engines and serves as a storage area on an element. The make-up of a data attribute consists of the prefix declaration of data (which will denote a storage space) and dash and another value that hopefully denotes the

meaning of the data being stored in the attribute. The jQuery Mobile framework relies on this new data attribute and does have a couple different uses of it, but most often you will see the “data-role” assigned to elements.

slide 11

The first and most important of which is the “page”. So a jQuery Mobile application is basically, like most website and applications, made up of a series pages. You declare a page using the data-role. Because we are using the data attribute from the HTML5 spec, your page needs to declare the proper HTML Doctype. In this example we have the necessary pieces for a jQuery Mobile site loaded. There’s the css doc, and jQuery and jQuery mobile JavaScript libraries. And you can see from the before and after images how progressively enhanced your application becomes. C-grade browsers or ones with JavaScript turned off will basically see the example in top. Those that support the capabilities of the jQuery Mobile framework will see the bottom.

slide 12

And within the page you can have whatever you wish, but jQuery Mobile also provides a couple more context areas that are optional but available within a page. Those are header, content and footer - each denoted by their data-role value. Each are optional, meaning one does not rely on the other, you can have just content or just content and header etc. I’ll get more into the roles of the header in footer a little later, but at the very least, myself, i always have a content div.

slide 13

> demo1_page.html

So here we have declared the proper Doctype, loading up the css declared the jQuery and jQuery Mobile scripts and have defined the div as a page with has a header, footer and content. If we run that, we get this and we are done. send the bill. So what the framework has done, is upon load, it inspects the DOM and takes page-ifies all divs with data-roles of page, and hides all but the first page. In this case we only had one page so we see that.

If we inspect the DOM with tools on the browser, we can see how the frameowrk has enhanced the experience by adding classes, aria roles and such. Now it does so from the script that is loaded and used to parse the DOM.s

< end demo

slide 14

If our application just needed one page, we’d be all set. The jQuery Mobile framework does allow for multiple pages and there are two different types to consider. The first is

internal pages, meaning multiple divs declared in the same document with data-role of "page". Those are navigable to by using the # and id attribute value, usually defined as an href on a link but can be done programmatically.

Then there is what is considered external pages, those in which a page is declared in a separate html document. If the page is considered external and you have the data-ajax attribute as true (which it is by default), then the framework will go about loading that page, inspecting its DOM, removing the first data-role page and putting it on the current document DOM. As such, if you are AJAX loading an external page, you can only have one div with data-role page defined in the document.

You can have an external page that doesn't use AJAX to add it to the current document DOM. That will basically act as you regular external link. In such a case, since it is not being loaded and parsed within the current document, you can have any number of pages defined in the non-ajaxed external page. My personal preference is to lean toward external pages, mostly with AJAX on, but using no-ajax where deemed appropriate.

Keep in mind, that with current state of the jQuery Mobile framework - alpha 4.1 - that pages loaded using AJAX and added to the DOM are somewhat considered cached. MEaning that the framework does not then remove that page from the DOM when navigating away from that. Of course you could write your own script to do so, but I believe in future version of the jQuery Mobile framework there will also be data roles that denote whether to cache a page or not.

Just to get a quick look of multiple pages and the difference between external an internal:

slide 15

> demo2_page_int.html

In this document we have two pages declared and they each have their own unique id property value. I have added some links to each page's content which point to one another using the # id. So this is considered a document with 2 internal pages. And when we run it, you will see that the framework has parsed the DOM and only is showing initially the first declared page. If we click the links in the content we can navigate form one age to the other. We'll also see up in the location bar that the hash is being appended to the url. Which also means that we have deep-linking to pages form the framework. If we just refresh on this second page, we'll load up with that one.

< end demo

If we take a look at the context of external pages:

> demo3_ext_ajax.html

Here we have one that is going to navigate to an external page, but using AJAX to load the page, parse it and add it to the DOM to transition to it. We'll run that, and see pretty much what we just saw in the last > demo. Not much difference, If we look at the page it is loading we see that all that is declared is really just a div with data-role of page. In this external document we are not including jQuery Mobile or the CSS, because the framework is really just loading this document, stripping out the first page and dropping it on the current DOM.

If we open up and look at the current state of the DOM with some browser tools, we see that the other page now resides on the main documents DOM.

< end demo

Just a quick look at what a no-ajax external page looks like:

> demo_4_ext_noajax.html

All we've done here is set data-ajax to false on a link for the external page. And that will just do a straight new load of the document. As such, if we look at the external page it is navigating to, we see that we are required to declare the jQuery and jQuery Mobile scripts and CSS to be loaded into that documents DOM.

< end demo

I wanted to say a quick note about the Back button you guys had noticed while i was > demoing the pages that one was being added to the header of a page once navigated to. In alpha 4.1 and previous versions of jQuery Mobile, a back button was added to a page programmatically - either an internal page or an external page with ajax - once transitioned to. This gave the user a way to navigate back to the previous page. What is being proposed for later drops of the jQuery Mobile framework is to remove that addition of the back button programmatically on transition to pages, mainly because the application is already living in this browser which has those capabilities, so the button is a little extrenuous. So it will be gone, but still be available to be added to the header programmatically by you. There may be times when you need it, like when you deliver your application natively or is considered a native web-app, but hopefully there will be time and we can go over that later.

slide 16

While we are on the subject of pages, i wanted to quickly talk about dialogs and their context in the framework. Dialogs are essentially pages, but are widget-ized as a dialog and are shown with a different UI context. They are really transitioned over another page and by default the url is appended with a ui-state query. Just a quick > demo to see what i mean.

slide 17

> demo_6_dialog.html

Here you see that we now have two pages declared in our document, and the link on the first page has a data-rel attribute declaring that the page to be transitioned to should be considered a dialog. If we run that real quick, we see that the second page is actually transitioned in with a UI context of a dialog. The Back button we saw in a previous > demo is now a close button.

Now technically, you could remove the data-rel attribute on the link and declare the page we want to use as a dialog as having a data-role of dialog and it will work the same. And if we set it back to page, you can see that it still has the same content, however the back button is now there, where there used be a close button. Both those just have a data-rel of back, which means it will just navigate to the calling page if clicked.

So there are multiple ways to present a dialog using the data attributes and the framework will pick those up.

< end demo

slide 18

So at the heart of the framework is the \$.mobile plugin. Don't know if you guys are familiar with jQuery and the concept of plugins? In rough terms it is basically just an object that resides in the namespace and accessible using the \$. whatever you call your plugin. Upon load of the jQuery Mobile script, this \$.mobile object is created and basically is the access point for the application; parsing the DOM, pagify-ing divs and providing navigation to pages. And on that object is various properties and methods that we can see here. I wont go through them, but the more important ones - at least for when i am working with the framework - are 1) activePage: which is read-only and is the currently focused div 2) pageLoading: which takes a true or false value to show the loading indicator and 3) the changePage method which allows you to navigate to a page programmatically. Just as i showed in the > demo with the links pointing to internal and external pages. The framework, internally, uses changePage when navigating to pages and it is available for you to use as well.

slide 19

Also on that \$.mobile object are some pre-configurable properties that you can define globally for your application and a few test methods - such as gradeA() which returns a boolean of whether the current browser that the document is in is considered grade-A by the framework. There are a lot of properties so you should check them out on the site, but one that is of note is the namespace property - ns. By default the namespace of your jQuery Mobile applications is blank. Meaning there is nothing between the data

attribute and the dash attribute value type. We can set a custom namespace for our application using the ns property of \$.mobile like in this example, and that will basically change the data-role or which data type to data-custom-role. This is great, because the framework allows you to not clobber any other data attribute declarations that you may have loaded in from a third party or from legacy code. So in most cases, i would advocate using a custom namespace in a jQuery Mobile project, but for this presentation i am not going to be doing that.

slide 20

Because of this ability to add a custom namespace, accessing the data attributes in the jQuery Mobile application has a slightly different context. In jQuery, you can access and modify a data attribute using the data() method. In jQuery Mobile, you use the jqmData method; works the same but is resolved to the namespace defined on the \$.mobile object. So there is a simple example of finding the content role on a page.

slide 21

This is more of real world example, and one i use occasionally. It's just a quick way to reference all the 50 dollar bills in the Fratelli hideout. This script doesn't actually check if they are counterfeit or not. You have to use the underscore library for that.

slide 22

Alright, so we have addressed pages and how data-roles play a factor in how the framework parses the DOM and widget-tizes pages and dialogs. You can think of a widget as a decorator for an element. So the framework loads up, parse the DOM and decorates elements based on their data roles (and a few other data attributes at times), essentially providing style and behaviour enhancement; and at times exposes another level of an API to interact with the decorated or widget-ized element. Along with the page roles we've seen, there are a couple more data-roles for common page elements such as buttons and navbars, forms, lists and the headers and footers which we saw briefly.

slide 23

Buttons are probably the most common widget and are fairly easy to implement in jQuery Mobile. All you have to do is define a data-role of button on an element - typically a link - and it is automatically stylized for you. The framework also recognizes the button element, as well, and works the same. Along with the data-role of button, default icons are provided that can be applied to buttons and their position within the button.

We can see the default icons there, and what the framework does is load in this single bitmap and assigns styles based on position - just like a sprite sheet. Fairly common practice, but a great way to do it. And this approach doesn't limit you to only those icons, you can create your own and just include some more CSS.

slide 24

With regards to layout of buttons, the framework provides a couple more ways to easily organize the presentation of buttons. The implementations can at times take on another context of how the buttons are related as well. A real quick way to organize a list of buttons is using the data-inline attribute, which if set to true will layout each sibling link that has it defined horizontally to its previous sibling.

That won't however, stretch each button so they are a percentage of the area within which they reside. Each button will just resize itself based on the textual content and be placed next to each other. You will have to do some extra work for that, and i know there are some libraries out there that will help with that, like jquery-mobile-960 which will allow you to assign grid types to the elements. Cool stuff.

slide 25

One other button grouping layout context is the data-role of controlgroup. This is set on the container for your buttons and takes a data-type for the directional placement of the buttons. As well as stylizing the buttons a little more to provide a context of a grouping.

slide 26

Another way you can organize buttons using the grid classes directly, so there is no pre-defined role that will do this, but you can layout out buttons using a gridding system within the framework and those are assigned as classes. The ui-grid and ui-block classes are appended with an alphabet character that denote columns. For the ui-grid, you have a through d which means 2 through 5 columns, and the ui-block is assigned on the elements to denote which column they reside in. When there are more elements then columns they will stack.

slide 27

> demo7_buttons.html

Just a quick > demo of how the button widget works and these different layout and grouping contexts look. We can see that setting data-role of button on a link, stylizes the button and stretches it to the width of parent area. If we had multiple buttons declared in the same container, by default they are stack vertically. We can set the data-inline attribute to horizontal and the framework will do a sibling check for this attribute and align them in the direction defined - here it is horizontal.

Containers with a control group data-role will take its children and create an over-all stylizing to them which presents a conotation of them being related to each other, just as in a select from a list sort of context.

And then we can use the gridding classes to provide a more unified layout that will stretch each button to the parent container area and also do some truncation on label text if the allotted space for a button is smaller than its content.

< end demo

slide 28

There is also another widget that provides a context for a series of buttons and that is the navbar.

slide 29

The navbar widget is a fairly common element found in mobile applications and is essentially what we have come to know as the tabbed view navigator. Internally, it uses same gridding system as we just saw, just under the covers and by using the data-grid attribute. The value of data-grid relates to the number of columns just like in ui-grid class. You see in this example code, that you don't have to assign a data role of button to the links, and the content of a navbar is actually a list of link items. The framework will handle recognizing and styling those elements.

And this, i have to say is probably another good example of progressive enhancement, because this navigational UI elements will still be available if everything fails and just HTML is presented. A user is still going to have this available for them to navigate to other parts of your site or application.

slide 30

> demo8_navbar.html

If we just take a quick look at this and run it, we see that the navbar is pretty ugly sitting there in the middle of the page. Typically you would use the navbar in the context of a tabbed interface at the bottom or top of your application to denote that they lead to different parts of your application.

If we just drop that navbar into the header we can see that it gets a little more style treatment and looks more like what we are used to. It can also be placed in the footer rather easily.

If we add some other fancy attributes we can see that it takes on a more unified context that we are familiar seeing in mobile applications.

< end demo

slide 31

FORMS! I don't find forms that exciting, but the framework does some exciting things to the elements of a form, if that is your thing.

slide 32

So the jQuery Mobile framework also widget-izes forms, and you essentially declare them as you would normally - using the form element and having pairing labels with fields just as you would. The framework just provides a little more styling to the elements to make it feel more of a part of your application. You can also listen for the submit of a form and override any functionality you will need - like validation.

slide 33

I just wanted to show you quickly what is provided and really this > demo is straight from the > demo site for jQuery Mobile.

jqm_forms.html

So here is basically the gamut of what the framework does to field elements, and if we resize to a small screen, we can see that they are also separated by a horizontal rule, that happens by assigning a data-role of fieldcontain to the parent of the inputs.

By default on the select elements, it will use the native select of the browser which on the desktop as a > demo looks ugly, but if we run it in a simulator like the iOS simulator we can see that they actually use that nice casino reel.

If we go down to one of the selects, we see it use the default select menu of the browser on the device. Pretty cool, and that is by default. The framework does allow for a custom select menu if you prefer, and if we select the last select element, we can see that displayed and selectable.

< end demo

slide 34

So the framework presents all this stylization and functionality of form elements by widget-izing the html elements, which also provides you some other API on which you can invoke other operations. Here we see we can access the widget by using the widget method and call some part of the API by passing a string argument. It is a whole other discussion on what widgets are and how to create a custom widgets, but you can think of them as basically decorators on elements. So through Progressive Enhancement we have these elements, that get decorated and provide an extra API on top of them. And if the browser is not considered supported by the framework, users can still use the form as they would normally.

slide 35

Which brings me to what i want to touch on briefly as native elements. As we saw with the select menu using the default context on the device, you can also specify on elements what you don't want the framework to touch by using the data-role of none. So all those input fields that have the pretty colors and rounded corners, if you set data-role to none on those input elements, they will look like the default elements for that browser. These, however, as far as i have played with it, have to be declared in the mark-up so the framework knows to overlook them when it is parsing and widget-izing the dom. They can not be set at runtime.

slide 36

Now on to lists. I have this weird relation to lists. I secretly love them but i also feel like sometimes it is giving up. They make sense in most contexts right, i mean it is a list of information that has some type of recognition as to how it is presented - at least i hope. Chronologically, alphabetically. what have you. But in some ways, and this might just be me, it feels like giving up. Much like every designer i know hates scrollbars. Its like admitting defeat. I don't know how to present all this information to you in a meaningful context, so here you go. You do the leg work. I'm out.

But in all seriousness i love lists, and they are usually my litmus test on performance of rendering. That is one of the reasons I was so downtrodden when i first checked out Hero on my devices. The lists were horrible. And truth be told, it is not just the framework that needs to take the blame, you as a developer need to know best optimization practices to get a list to perform well. So in a sense, lists have this user and developer dichotomy where every one has given up.

I am kidding. Alright enough about what lists mean. You can create pretty lists in jQuery Mobile by assigning a data-role of listview. Pretty simple.

slide 37

So there is a simple way to enhance lists, and what's more, you get a couple item renderers right out of the box. You have the default, which is just textual content in a box, but you can also have 2 type of icon renderers and you don't need to assign any extra roles. The framework is smart enough in its parsing to recognize what type of renderer should be presented by inspecting its list children. You can also have what is considered a split button list which basically is two links within a renderer. Say one that tells you the name of the album and takes you to a description and one that just says Buy Now! and takes you to the store.

It's also fairly simple to create your own list item renderers using templates. I have used mustache and the jQuery template library to easily create custom item renderers.

slide 38

> demo9_lists.html

Just a quick > demo that shows you all these item renderers. We'll start off with no data-role assigned. Run that and see that we have our simple list. A couple UI enhancements as far as fonts and coloring to incorporate the look-and-feel of our application, but still available as a simple list.

If we add the data-role of listview, here is just a simple select list with your basic label item renderers. If the content is a link it adds the indicator on the right to denote that the item will take you to another location. You can also get a little fancy and add a data-inset attribute that will give it a nice look and feel. Pretty boring.

< end demo

If we also look at **one** of the icon renderers we get.

> demo_10_lists_icon_large.html

We can see that no extra data-roles are assigned and the framework will know how to parse an image within the link. By default it will use a large icon which is 80x80.

< end demo

> demo_11_lists_icon_small.html

You can define an item renderer to use a small icon by assigning the ui-li-icon class on the image element. That will make a 16x16 icon for the item renderer and will be positioned vertically along the content.

< end demo

> demo_12_lists_split.html

And just a quick > demo of the split lists. We can see by defining 2 links within the list item, the framework will consider the list item as a split item renderer and properly present that. And by using the data-split-icon and data-split-theme attributes on the list, we can apply some custom style.

This > demo actually also defines a data-filter attribute which will automatically drop a search bar above the list and allow for live filtering. This works great on the desktop, not so much on the device; but it is available.

< end demo

There is also support for list dividers which are common on contacts lists, split up by alphabet. So jQuery Mobile list widget is pretty powerful, but like i said you as developer also have to do as much optimization on presentation as you can to get it to perform well. Don't leave it up to just the framework.

slide 39

So that pretty much wraps up the widgets that are available to you. And the framework is pretty flexible and since it is built on jQuery you can create and define any other custom widget you want. I know they are working on a nice date picker and maps widgets for the framework and they are up in their github, so you should check them out.

So, I haven't really addressed about themes, but you may have noticed them in some of the > demos. jQuery Mobile provides some theme swatches to stylize your content and as we saw with the gridding, it uses alphabet characters to denote theme. By default, the framework provides 5 defined color swatches - a through e. You can define however many more you want up to z or override these, whatever. I advise against doubled-up wrapping of the alphabet, such as data-theme-aa, because then the styles tend to get a little introspective of themselves and your application start to become a little "deep". No one wants to think when they use your application. So keep that in mind.

slide 40

So these swatches are defined in what is considered the default theme for the framework which is really just the CSS file that comes with it. They also have another theme available called Valencia which styles elements like this. You can see that the swatches are different, but it also has enhancements on the other widgets as well, like sharper corners. Valencia is available but i think it is more of a guide as to how you can create your own theme, which as i mentioned is just taking the default CSS that comes with the framework and modifying it to your liking.

There is also talk of retrofitting the ThemeRoller from jQuery UI to be used for jQuery Mobile applications. Don't know if any one here has worked with jQuery UI, but the Theme Roller is just a form that allows you to easily apply styles and colors and see how they effect elements in real time. And then it will export that style sheet for you. Pretty neat, though i haven't played with it much.

So you have theming and the possibility to create custom themes in jQuery Mobile and it is really a great approach. The whole look and feel is defined in the CSS and then applied to elements using JavaScript. Meaning you will find very little type and no id style rule declarations in the CSS. Its mainly classes and media-queries.

slide 41

I also wanted to touch a little on accessibility. Unfortunately - and this is very bad of me - I don't know enough accessibility which is shameful. Thankfully, though, the Filament Group seem to be pretty adamant about having their framework be accessible and follow the WAI-ARIA (Web Accessibility Initiative - Accessible Rich Internet Applications) specification as much as possible. What the WAI-ARIA spec details is accessible roles, states and properties on elements as well as defining keyboard navigation using the tabIndex attribute and alerts for live regions or rather content that is updated dynamically, mostly with regards to AJAX. A guy by the name of Gez Lemon wrote a great article about it and you should go check it out.

slide 42

So i watched a presentation by Scott from the Filament Group (i wont butcher his last name by trying to pronounce it - he never says it in his presentations) and i thought it was rather cool and was a great feature of the framework so i thought i would do a quick screen reading > demo to show you all how your application would still be accessible.

> demo

CMD+F5 - start voice over.
navigate to browser.
CTRL+OP+SHIFT+DOWN go into content.
CTRL+OP+ARROW(L/R) navigate in application.

< end demo

So that's pretty cool. Like i said i shamefully do not know enough about accessibility to speak about it at length, but the framework does its best in assigning the aria roles, states and properties to the elements to assist in screen readers.

slide 43

So that is basically a nuts and bolts of what the framework offers and i what better way to wrap it up than to depress you with what else it offers... size.

In mobile applications, speed is king especially when delivered in a mobile browser. So just a quick look at what you are starting out with when you want to develop a jQuery Mobile application and the file size and load times you will be facing. Because the jQuery Mobile framework is built on jQuery, that library is necessary to be loaded first before the page loads the jQuery Mobile JS and CSS files. So we are looking at a close to 60k download of files - minimized and gzipped. Also through in the HTML document and how many pages you have defined and you are looking at possibly some hefty weight.

Now obviously one of my first rules of optimization and speed for your application is to only define pages absolutely necessary in a document. Use external pages. That will

get the main document presented faster, but you may see the content unstyled before the CSS and JS is loaded in. So at times, on slower browsers or slower connections there will be this flash (no pun intended) of raw HTML content - the content people will see in what is considered grade-C browsers.

I have been playing around with a couple ways to get over this hurdle and really haven't been overly emphatically excited about any of the solutions, but they are possible and i just wanted to touch on them. And these are just some solutions to think about to get away from flashing un-enhanced content to the user prior to the framework having loaded and available to enhance the application.

slide 44

There is the age old, not so good hiding of content until all resources are available. If we look at this we are setting the body to hidden and then upon ready, using jQuery here, we then flip the visibility back on. It works seems a little hack-ish but is possible. However, you don't get to present any content while everything is loading. There is no indication to the user that you page is actually doing anything. Which isn't that great usability wise.

slide 45

Another solution i have worked with is using LabJS - which stands for Load and Blocking JavaScript. Its basically a dynamic script loader which allows for you to load in script on demand and based on requirements. Fairly simple to use - just define the scripts you want to load and define the wait callback. In this example, I actually have my main page defined in another document and loaded into the body upon script load complete. So this allows for you to show some content as the page loads in our resources. You can show a splash screen or what have you. One drawback is that LabJS only loads javascript. So we had to define our CSS link to be loaded which will block our renderer slightly, but still a viable solution.

There is also another library out there called yepnope, which i don't know if any of you are familiar with. It basically is a poly-fill loader that loads resources based on the truthiness of a value. Most often then not, it is used in tandem with Modernizr.

I have tried to get a solution up and running with yepnope and Modernizr but have been unsuccessfully as of yet. Need to look more into it, but that may be a solution as well. However, yepnope is used more for defining what poly-fills or shims to load in based on the capabilities of the browser, so it may not hold weight as being a viable solution. Most likely you would use yepnope in tandem with your application instead of as a library to help in presentation of the initial load.

So just something to think about when developing your application. It is going to take time to load, and how are you going to handle that. I don't want you to get the wrong impression, on the devices i have tested with the load time is pretty neglegdable if the

application is structured right, but there are times when things will be seen in the raw before the framework has time to parse and beautify the DOM.

slide 46

So I have covered the framework and maybe talked about some things to consider with regards to loading your application, but I also wanted to touch on some ways on bringing your application out of the mobile browser. One thing to also consider when thinking of web application in this mobile space is having you application available outside of a user having to click on the browser each time to access your application - having the application reside on the home screen of your device and being considered what is called a “web-app”. I don’t know how many of you have done this but usually from a menu on your mobile browser you can select to add a web page to your Home Screen. That essentially makes you web application what is considered as a web app on the device. That will basically just launch your application in its own browser window and doesn’t rely on the user opening the browser on the device and pointing to where your application resides on the web.

slide 47

And there are a couple solutions and niceties you can add to your page to have ability. These can be found on the apple developer docs but are implemented by other browser vendors and i have tested it on iOS and android and they all work.

The first is viewport which is also for in browser but also when considered a web-app. And that just defines the visible area, the initial scale and if the user is allowed to scale the content using touch gestures.

There is also some web-app meta data that will define the web page as being able to be added to the home screen and considered its own web app as well as stylizing the status bar when your web-app is launched.

Apple also defines what are considered web-clips which are icons to be shown on the device, just like you have for native applications, you can define icons that will be shown on the device once the app is added to the home screen - which is pretty cool. There is also a start-up image, which is the image shown when you launch you app before it is shown in the UI web view. I have never gotten that to work, but maybe i am doing it wrong. Who knows.

Just a quick > demo in the simulator so you can see how it works.

slide 48

> demo_webapp.html

Here you can see we have all the meta and links defined and I have a little script here that will test if the browser supports the standalone property of the navigator on the window. When your web app is in a browser on mobile, hopefully it will be supported and that would allow for your application to be added to the home screen and serve as a web-app. If it is available, you can test for its value to determine whether a user is currently running in a web browser on the device or in a UI web view as a web-app does. Which is a great way to alert people to the fact that they can add your app to their Home Screen. A couple applications do this, I know Untappd does it, checks for this and shows a little alert telling you you can add it to your home screen.

So if we run that in the simulator you'll be able to see more clearly what I mean.

< end demo

slide 49

Some things to consider when having your jQuery Mobile application as a web-app, is that any links that do not use ajax to navigate to another page will actually launch the browser. As well, as I mentioned before, since the web-app is now residing in a UI web view, you lose the browser controls to navigate back and forward, so when developing the application keep that in mind and use the Back button available in the jQuery Mobile framework. And as we saw with the standalone property, we can test if we need to provide back buttons based on that.

As well, you might want to consider looking into caching pages for your web-app. It is something I intend to look more into and haven't played with much but you can define a cache manifest on your HTML document to cache pages which is something to consider because your web-app will go out to find pages on the network and if there is no internet connection, no one will be able to use your application. And with that come online-offline synchronization which is a whole other presentation.

slide 50

So web-apps are somewhat native or have the ability to appear native, but what if you want to take your current online jQuery Mobile application and have it available as a native application on the device. There are a handful of solutions out there, but the easiest in my testing were PhoneGap and Adobe AIR. And by easy I meant, just taking the files of an application I already created, and dropping it into another project and being able to deploy a native application in minutes without having to modify a whole bunch to get it to look and act just like it does in the browser.

slide 51

So PhoneGap is basically a template suite for deploying a native application. They have templates for various IDEs to deploy to various mobile platforms and is pretty straight forward to use. Now I haven't really testing incorporating native device access with

applications but it is provided through a JavaScript interface. In essence, all you do is in your main HTML file, include the JavaScript for PhoneGap, and you are good to go. It will take care of adding your application to a UI web view and you have access to some native APIs through the JavaScript proxy. Fairly straight forward and easy to get up in running in no time.

slide 52

Also dear to my heart is Adobe AIR. So AIR provides some compiler tools to compile your ActionScript and Flex applications a native installer. Within an ActionScript project it is fairly easy to get my jQuery Mobile application running in an instance of StageWebView. The StageWebView is basically a light-weight HTMLLoader, if you guys are familiar with the web API for Adobe AIR. The HTMLLoader is Adobe AIR's custom version of WebKit. StageWebView is the actual native browser on the device - just like your basic UI web view that PhoneGap drops your HTML content into. The downside of StageWebView being a light-weight HTMLLoader is that there is no communication bridge available for it. With HTMLLoader, you have access to the DOM. With StageWebView, not so much.

Sean Voisen wrote a good article on StageWebView that touch in part on the JavaScript to ActionScript communication, and i got it to work on the desktop, but it is pretty spotty on the device itself. I need to look more into it. But needless to say the communication between your web app and the native device API that Adobe AIR has may be a little limited.

slide 53

There is also a couple more software suites out there that provide the ability to take your HTML knowledge and deploy installer files for devices.

There is an XCode project template for a NimbleKit app, as such it only support iOS but is quite easy to use and actually provides a nice API for create native elements and navigation as well as provide support for native device API and database. It is free to test with but you have to buy a license to deploy to the store.

QuickConnectFamily which seems to be the less known of all these tools was rather quick to get up and running as well. However, it does not provide the nice templates that PhoneGap and NimbleKit do, and you have to set up your project manually to use the quick connect files. And as with PhoneGap and NimbleKit it provides the means to use native UI elements and device access.

Titanium from appcelerator is a bug name in this HTML technology stack to native race. This one was a little harder to get up and running with an existing jQuery Mobile application and i could never get it to fully work. In theory you should be able to just point the main file as the one to be loaded into a Titanium.UIWebView and have your application available - but it was a bear to try and get going and actually couldn't get it to

work. I don't want to count it out though. But it does make sense, because Titanium actually provides its own JS framework and API for building web applications. So you wouldn't typically have an existing application and drop it into Titanium and publish. You would build the application with their API which gives you native elements and device access.

There is also Rhodes which I haven't played with but in theory, for any Ruby guys out there might be a great solution. And what Rhodes provides is an MVC application framework built on Ruby with the ability to add some native UI elements and device access, and by default it actually uses jQuery Mobile as the default UI. In theory you can just switch this out with jQuery Mobile instead and you could be good to go. So check that out if you are into Ruby.

slide 54

I think that is it. Are there any questions?

slide 55

Thanks!