jQuery and jQuery Mobile in Detail

jQuery and, increasingly jQuery Mobile, are becoming standard libraries for use in developing web applications and mobile web applications. They are not the only libraries available to provide this support – there are also:

- Dojo – an alternative to jQuery that incorporates 2D and 3D drawing, dynamic user-interface components and drag and drop features. Dojo is more accurately described as a toolkit, since there are parts of it that are there to generate sections of an app – i.e. it does more than act as a code library
- Sencha touch – this is an alternative to jQuery Mobile that provides better graphics support (incorporating Raphael – a SVG graphics library for HTML apps) and is oriented more towards Javascript coding than HTML mark-up. However, Sencha is limited to WebKit based browsers (iPhone, Blackberry and Android) and provides no support for WinMobile
- XDK – this is a full web-based HTML5 development environment that provides access to everything you need from inside a browser. It incorporates emulators for a range of devices (including iPhones, Androids and Win Mobiles) and an App Framework that defines components for the most common Mobile Web App tasks. It is not limited to Mobile web apps, but incorporates a neat UI prototype so you can create user-interfaces using drag-and-drop
- PhoneGap – this is the most long established mobile web app development kit and is distinguished by its capability of creating native mobile apps by packing the HTML 5 mark-up and scripts into a package compiled for any of several platforms. PhoneGap provides an online Build service that lets you upload a web app for compilation – the result is a Hybrid web-app that can be uploaded to the various platform app stores (iTunes, Google Play, Windows App Store etc.). PhoneGap inter-operability is built into XDK

These are just the mainstream tools for building mobile web apps (and hybrid mobile apps) – many more exist. Our reason for selecting jQuery and jQuery mobile is that the development processes are simpler. In Sencha Touch, everything is done in Javascript code, which means a lot of learning before you can get started, XDK and PhoneGap have similarly steep learning curves. However all of the above and various other free to use libraries and toolkits are available for creating mobile web apps, ad once you've gained some experience with one framework, it is usually a good idea to check out the alternatives in case they work better for you.

In essence, none of the above frameworks are necessary. A web app that is suitable for a mobile can be built in plain HTML, CSS and Javascript. However, that means creating a lot of fairly boring definitions for button styles, lists and other on-screen widgets before your app would be usable.

jQuery and jQuery Mobile are currently the most widely used and documented libraries for mobile web apps. There are loads of websites and books available to make learning to use the libraries easier and quicker (beware, there are among these, a fair proportion of dross, 1-skill websites and badly written books [I know - I've bought most of them]).

jQuery

jQuery puts a subtitle on its main website – "write less, do more", and this is a good explanation of the philosophy embodied in the library. The 'killer' features of jQuery are:

- Cross-Browser: jQuery works the same in all current browsers and quite a lot of older ones. This saves developers from having to create alternative solutions for different browsers – until recently a major problem for web developers
- Lightweight: jQuery is a very small (by current standards) library of code. It is fast, and so will almost always help to produce a faster, more efficient site than HTML, CSS and Javascript written by the majority of developers. It is possible to produce faster code without jQuery, but it isn't easy
- Is widely supported: jQuery provides add-ins for user-interfaces, mobile development (jQuery Mobile) and unit testing. In addition, it is compatible with a large number of 3rd party Javascript toolkits (app frameworks like Anchor, the AMD modular development standard, PhoneGao, Codiqa [rapid prototyping], various user-interface themes etc.)

jQuery provides three core features that are widely applicable:

1. DOM Traversal and manipulation: this makes it quicker and easier to select specific items of groups of items from a live webpage and make changes to them. Selectors return jQuery objects which are simple wrappers for DOM objects (e.g. <h1> elements, <input> elements etc.) and that can be specified very flexibly
2. Event handling: jQuery adds many events to the ones already defined in the HTML DOM, and makes it easy to attach Javascript code to them
3. AJAX: Asynchronous Javascript and XML is now a core feature of Javascript applications, and refers to the ability to update parts of a web-page on the fly (rather than refreshing the whole page). It is responsible for the ability of web pages to have sophisticated and snappy user-interfaces. jQuery makes AJAX a lot simpler to do

jQuery Selectors

Using Javascript with the Document Object Model, you can select any part of an HTML document using one of four different selection functions:

`document.getElementById():`	this returns a single HTML element with the given ID
`document.getElementsByName():`	this returns an array of HTML elements with the given value for the name attribute. Note that any number of elements can be given the same value for the name attribute, whereas only one element can have a given value for its id attribute
`document.getElementsByTagName():`	this returns an array of elements with a particular HTML tag – e.g. it could be used to return all of the <H1> elements
`document.getElementsByClassName():`	this returns an array of elements that have a particular class value. Typically, an element is given a class to associate it with a specific CSS style-sheet. This selector therefore lets us select groups of elements according to their display properties

jQuery replaces all of these with a single function, which also happens to be the default function of the jQuery object (the whole jQuery library is defined as a single object with functions and properties). A jQuery selector is formed like this:

```
    jQuery(<selector expression>)
```

The selector expression is a string value that indicates the id, name, tag or class of the element(s) you wish to select, and can contain further information (predicates) to narrow the selection. What type of element is selected depends on the format of the selector expression:

`jQuery("p")`	Selects all <p> elements in the document
`jQuery("#id")`	Selects the single element with the given ID
`jQuery(".class_name")`	Selects all elements which have the specified CSS class

In addition, jQuery provides a shed-load of other useful selectors for selecting multiple types of element (e.g. `jQuery("p, h1, li")` selects all <p>, <h1> and <li> elements in a document), making more complex selections (e.g. `jQuery("#list li")` selects all <li> elements that are inside a <ul> with the ID "list"), the first, last or nth element in a selection, elements with specific attribute values etc. The w3Schools website has a page explaining the full range of selectors available (http://www.w3schools.com/jquery/jquery_ref_selectors.asp) and includes a link to a jQuery Selector Tester (http://www.w3schools.com/jquery/trysel.asp?filename=trysel_basic&jqsel=p.intro,%23choose) that lets you try them out on a demo page or your own mark-up.

There are a few distinctive features about jQuery selectors you need to be aware of:

- jQuery selectors always return an array of elements – even if the selector picks only a single element. This is most noticeable with an id selection (e.g. `jQuery("#page_1")`), since the standard DOM `getElementById()` function is the only one that returns a single value (HTML rules that only one element can have a given ID). jQuery will ALWAYS return an array of elements, so to access the returned single element, it still has to be treated as the first (and only) element in an array. To get an element with the id "page_1", you need to use the expression `jQuery("#page_1")[0]` to pick out the first array element
- jQuery selectors return jQuery objects – these objects that *contain* the DOM elements you are selecting. Effectively, that applies additional mark-up to each of the returned elements so that they now contain some additional jQuery functionality. Everything returned from a jQuery selector has a length property (to indicate how many elements are returned), and is compatible with a range of jQuery utility functions. There is a good explanation of this at http://www.learningjquery.com/2008/12/peeling-away-the-jquery-wrapper
- jQuery selector expressions can be chained to combine the results of multiple selectors. For example, `jQuery("#myForm, .boldHeading, h3")` will return an array containing the specified form, all the elements with the CSS class boldHeading and all <h3> elements
- jQuery selector expressions can be combined to narrow a selection down. For example `jQuery("li")` will select all the `<li>` elements in a document, but `jQuery("#mainlist li")` will only select all the `<li>` elements inside an element with the id `"mainlist"`. Note that this is different from chained selectors (above), which are used to combine different selections
- You can add predicates to jQuery selectors to further narrow down a selection. For example, :first, :last, :even, :odd and :eq() can be used to select specific elements or

groups – `jQuery("li:even")` will return only the even numbered elements from the list. See http://www.netmagazine.com/tutorials/getting-most-out-jquery-selectors for a good tutorial on this

There is one further trick up jQuery selectors' sleeves – the keyword `jQuery` has a shorter alias, `$`. Because of this, any expression you've already seen using `jQuery()` is also viable as `$()`. This can save a lot of typing.

Exercises

1. Using a web browser, go to the jQuery Selector Tester page at http://www.w3schools.com/jquery/trysel.asp?filename=trysel_basic and enter jQuery selector expressions in the box to select the following from the HTML fragment shown on the page (e.g. entering "body" selects the whole document):

    a. The main heading in the document

    b. Every <p> element

    c. Every <li> element

    d. Every paragraph with a class of "intro"

    e. The complete <ul>

    f. The first and third list elements (Goofy & Pluto)

    g. Only the first element in the list

    h. Only the last element in the list

2. Copy the webpage shown in the box below and paste it into the left hand pane (the HTML pane) of the jQuery Selector Tester used in exercise 1 *in place of the existing HTML mark-up*:

```
<!DOCTYPE html>
<html>
<head>
    <title></title>
</head>
<body>
    <h1>First heading</h1>
    <h2>First sub-heading</h2>
    <h2>Second sub-heading</h2>
    <h1>Second heading</h1>
    <ul id="first_list">
        <li>First item</li>
        <li>Second item</li>
        <li>Third item</li>
    </ul>
    <ul>
        <li>Red</li>
        <li>Green</li>
        <li>Blue</li>
        <li>Purple</li>
    </ul>
    <ul id="food_list">
```

```
            <li type="v">Egg</li>
            <li type="v">Cheese</li>
            <li>Bacon</li>
            <li type="v">Onion</li>
            <li type="v">Chips</li>
            <li>Steak</li>
            <li type="v">Tofu</li>
        </ul>
        <h1>Third heading</h1>
        <form action="#">
            <input id="name" type="text" size="20"/>
            <input id="age" type="number">
            <input id="email"  type="email">
            <input id="go_button" type="button" value="Go"/>
        </form>
    </body>
    </html>
```

Enter selector expressions to select the following:

a. Every <li> item on the page

b. Every <li> item in the first list

c. The last item on the first list

d. Every <li> item in the second list

e. The even numbered items in the second list

f. The first item from each of the lists

g. Every item in the list of colours

h. The first item in the list of colours

i. Every item in the list of foods

j. Every item in the food list suitable for vegetarians (type="v")

k. Every item in the food list NOT suitable for vegetarians

l. The email box on the form

jQuery Effects

jQuery includes a lot of functions that can manipulate the mark-up on a page by attaching CSS classes and animations. CSS style-sheets can always be used to alter the colour, visibility, position or even movement of an element on a page, but some of these facilities are so common that jQuery has included special functions for them:

Function	Description	Example
.hide()	Makes an element invisible	$("p").hide(); // hide all <p> elements
	Makes an invisible element visible (does not affect a visible one)	$("#title").show(1000); // Show the title over 1 second

.toggle()	Make visible elements invisible and vice-versa	$(".menu").toggle(); // show/hide all .menu items
.fadeIn(), .fadeOut()	Slowly fades an element in or out	$("canvas").fadeIn("slow"); // Make canvas elements fade
.fadeToggle()	Fades visible elements out, invisible elements in	
.fadeTo()	Fade an element to a given opacity (0 = invisible, 1 = fully visible)	$("#icon").fadeTo(0.5); // Make element 50% visible
.slideDown(), .slideUp(), .slideToggle()	Slide (reveal or display elements vertically). Can specify "slow", "fast" or a time in mS	$("button").slideDown(2000);
.animate()	Move an element horizontally (only elements with a position property of relative, fixed or absolute)	

Table 9.1: jQuery effects

Using the methods in table 9.1, document elements can be hidden, revealed, faded in or out and moved around the screen at various speeds. While none of this is essential for a web-app, it comes under the heading of "pizzaz"; making an app more interesting and attractive.

jQuery HTML Manipulation

Four main functions provide facilities for accessing document mark-up and manipulating it:

Function	Purpose	Examples
.text()	Retrieves the text in an element or updates it	var s = $(“#heading”).text(); $(“#heading”).text(“Hello”);
.html()	Retrieves the html mark-up enclosed by an element or updates it	var items = $(“#list”).html(); $(“#list”).html(“<li>Item</li>”);
.val()	Gets the content from an input element or sets it to a new value	var n = $(“#name”).val(); $(“#email”).val(“joe@bloggs.com”);
.attr()	Retrieves or updates an element’s attribute value	var url = $(“#url”).attr(“href”); $(“#url”).attr(“href”,“www.uws.ac.uk”);

Table 9.2: jQuery’s document manipulation functions

Using the four functions described in table 9.2, you can access or change any mark-up in a given document.

jQuery Document Traversal

Document traversal describes the process of locating parts of a document based on their relation to other parts. For example, I might want to change the CSS style of all of the <li> elements within the first <ul> element to occur after a particular <h2> element. If I already have a selector for the <h2> element, I can use this to traverse (move across) the document to locate the <li> elements.

Document traversal requires that you know the relationship one element has to another. For example, the <li> elements within a <ul> element are its children, or child elements; conversely, the <ul> is the parent element of the <li> elements it contains. From the field of genealogy, a number of terms have been appropriated because they make sense in stating the relationships between elements:

- Parent – an element that directly contains other elements (the child elements). For example, the <li> elements in a <ul>
- Child – an element that is directly contained by another (its parent). For example, an <input> element inside a <form>
- Ancestor – an element that contains a given element at any level of nesting. e.g. <html> is the ancestor element of everything else in the document
- Descendant – an element that is contained directly or indirectly by another (e.g. all <li> elements are descendants of the <html> element even though they need to be within a <ul> or <ol> element
- Sibling – an element that shares a parent with other elements. e.g. all the <li> elements inside a specific <ul> are siblings.

The jQuery object contains a rich set of functions that allow you to move through a document from any initial selector. The page at http://www.w3schools.com/jquery/jquery_ref_traversing.asp gives a comprehensive list of these and there are too many to discuss here, but important examples are:

Function	Purpose	Examples
.children()	Returns all direct child elements of a given element	$(“#list”).children().hide(); // Hide all <li> elements inside the list
.parent()	Returns the direct parent of an element	$(“li”).parent()children().length; // How many elements in the list I’m in

.each()	Execute a function for each element that matches a selector	$(“p”).each(function(){
		  console.log(this.text());
		});
		// Print each <p> element on console

.closest()	Returns the first ancestor of the selected element that matches a selector	$(“#email”).closest(“div”).css(“color”:”red”}); // Colours the nearest div ancestor’s text red
.find()	Returns all descendant elements of the selected element (children, grandchildren etc.)	$(“#home”).find(“h1”).css({“color”:”black”}); // Turns all <h1> elements in the home page black

Table 9.3: Document traversal functions

jQuery AJAX

AJAX is the basis of most web-apps that are worthy of the name. A web-app is (at least by my definition) an application that just happens to run within a web browser. This distinguishes it from a web page in a number of respects; web-apps have a user-interface instead of just document mark-up, web-apps can update page elements individually and coherently rather than just being displayed wholesale.

The technology that gives web-apps these capabilities is AJAX (Asynchronous Javascript and XML).

The term was first used by Jesse James Garrett to describe the way that web apps like Google Docs and Google Maps worked. Essentially, AJAX allows web pages to be updated without updating the whole document. Page refreshes are expensive in computing terms (they take a comparatively long time and soak up a lot of processor power), and by getting around the need to refresh a whole page just to update a single element on it, AJAX makes it possible to create web-apps.

As you've seen already, Javascript, with or without jQuery, is capable of updating a single element or groups of elements in a DOM. Web browser engines are fairly fast at re-drawing parts of a page, and so a web page can be used as a quick and flexible user interface. However, the power of the web is still a crucial factor, since otherwise why bother involving it at all.

Google Suggest is a good example of this; you start typing a search term and by the time you've entered a couple of letters, Google has come up with a list of suggestions; one that gets better as you enter more letters:

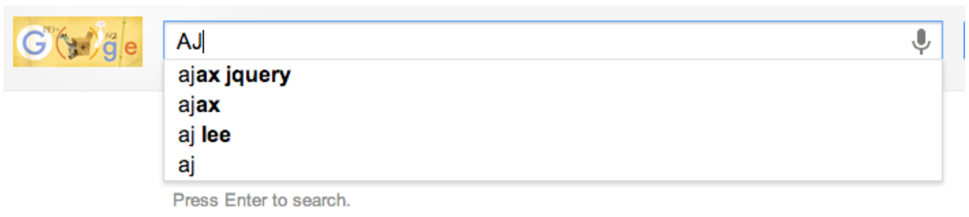


Figure 9.1: Google Suggest – a service that uses AJAX

The list of words suggested by Google Suggest comes directly from Google's Servers; this indicates that a lot has to happen to make Google Suggest work. Every key a user types must be added to the current search term (e.g. "AJ" is the search term in figure 9.1), this term must be sent to the Google service, the list of matching terms (a list of common words and phrases used in Google searches) has to be returned and the display must be updated to match. If this took more than a few milliseconds, users would consider Google Suggest to be just something that gets in the way.

AJAX is, as you might imagine, fairly complex and can be difficult to set up. It is a client-side technology (it runs within the browser) but must also manage access to a web service (e.g. the Google service that returns lists of words for Google Suggest). Fortunately we don't need to worry about the web service ***here*** (we can just assume that one will exist to provide the data we want; Google already had access to a list of common search terms when Google Suggest was developed), so all we need to manage is sending out the request for data, collecting the data when it is returned by the server and updating the document with it. That's ALL. Of course, in a real situation, you may just have to produce that important web service.

The biggest problem with this is that it is not possible to request data from the service and then immediately update the document with it. AJAX's purpose is to deal with requests ***asynchronously***, so that once a request is made to a web service, AJAX will nominate a separate block of code to handle the returned results whenever they arrive.

A request to a web service is very slow compared to a bit of local processing (e.g. collecting the same data from local storage) – so slow that we need to build in a facility that waits for the returned data to be delivered. That is the 'Asynchronous' part of AJAX, indicating that the process needs to involve dealing with components that work according to a different time-base. In client terms (in the browser), processing is generally fast and results can appear almost instantaneously. When a server gets involved, the whole combined time of sending out a web request, waiting for the server to produce the results and these results being returned to the client.

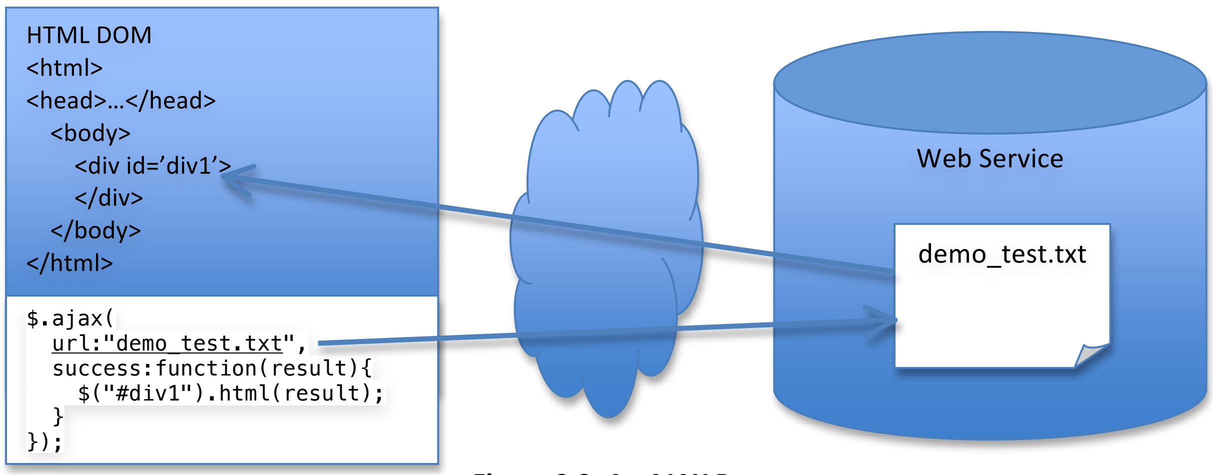
The asynchronous part of AJAX works because Javascript is good for writing code that can handle unpredictable delays. Recall that Javascript functions are fully featured objects – they can be stored in variables, passed as parameters into functions and used as 'callbacks' - i.e. routines that can be called by other code at a time that is convenient to the other code. This facet of Javascript functions is used when an event-handler is needed; e.g. a function to call when a document has fully loaded (window.loaded or $(document).ready()) or when a button is clicked. In AJAX, the unpredictable delay is the amount of time it can take to retrieve values from a web server (e.g. an updated list of Google search phrases).

In jQuery, a simple Ajax request can be made like this:

```
$.ajax({
    url:"demo_test.txt",
    success:function(result){
        $("#div1").html(result);
  }
});
```

Listing 9.1: A simple AJAX request in jQuery

In listing 9.1, a text file (demo_test.txt) is available at the server that the current page was hosted from. The `ajax()` function requests this file (which will take some time to retrieve), and indicates a 'success' function to execute when the file is available. Under the hood, the jQuery code will now wait until the complete file has been returned and then use the nominated function to place the file text (returned in the `result` parameter) into `<div id="div1">` in the DOM.



```
$.ajax(
  url:"demo_test.txt",
  success:function(result){
    $("#div1").html(result);
  }
});
```

Figure 9.2: An AJAX Request

AJAX can do much more than simply retrieving a file from a server; AJAX calls can execute processes on a web server so that live results can be compiled and returned (e.g. weather reports and bank statements), can send data to the server (e.g. authentication for accessing an online bank account) and can include handlers for if the results are corrupted or take to long to come back.

The most common use of AJAX in mobile web apps is to retrieve raw data from a server to update the app in some way (a GET operation), or to upload a small amount of data to be added to a database at the server (a POST operation). For example, if you have an online TO-DO list, the actual list of items will typically be stored in a database that can be accessed by a web service. When the TO-DO app is started up, it will retrieve the most recent updates using a GET request so that the app shows a current list. If items are added, that information will be sent back to the server in a POST request so that it always keeps a completely up to date list. Items can also be deleted and updated. If you access your TODO list from more than one device, all of the devices will show the same list. As an aside, an HTML5 TO-DO list app would usually store the most up to date list locally in case there was no web connection. Any updates made while the app was offline could then be synched to the server when a connection became available.

It is worth repeating – the power of AJAX lies in the speed with which a small part of a webpage can be updated compared to the time it would take to do a complete page refresh. This is partly because of the time it takes to transport a whole page of data, but more because of the time it takes a browser to organise and render a complete web page compared to simply updating a fragment of it.

A complete, annotated list of jQuery AJAX functions is provided at http://www.w3schools.com/jquery/jquery_ref_ajax.asp. There is a good tutorial at http://www.sitepoint.com/ajax-jquery/.

jQuery Mobile

We've already had a good look at jQuery Mobile (jQM) in earlier chapters. Most of the current mobile web browsers are well featured and are able to cope perfectly well with webpages and apps that were written for normal, desktop browsers. The only problem with normal web apps is that they are written for much larger screens, and so a normal page that is rendered on a mobile browser would appear so small to be readable, and the control areas like buttons and list elements would be too small for a user to tap reliably with a finger.

Because of this, much of jQM is about re-styling apps so that they are of a readable size and are "finger-friendly" – i.e. so that on screen control areas are large enough to tap on confidently with an average finger. This is largely done in CSS style-sheets, which render page elements in a format more suitable for mobile devices. AJAX is also used in jQM to implement page transitions and to add and remove CSS styles as page elements are updated.

Another core part of jQM is that it works behind the scenes to make sure that features that work on one platform work on all platforms; it removes the differences between browsers that cause web-app features to work well on some browsers and not at all on others. For some illustrations of this, check http://www.texaswebdevelopers.com/html5/, and for a complete list of current browser status, you will find comparisons art http://html5test.com/. Of course jQM can not make ALL browsers work in exactly the same way, but it does eliminate the major compatibility problems that can affect common web-app operations on the main mobile browsers. http://jquerymobile.com/gbs/ gives a complete list of the browsers that are fully compatible with jQM (most of them).

In addition to the general features provided by jQuery Mobile for styling web apps and making them compatible with common mobile browsers, it contains a full Application Programmers' Interface (API), which incorporates:

- Widgets: i.e. on screen controls and display elements, such as listviews, buttons, checkboxes, control-groups, collapsible content etc.
- Responsive Screen elements: i.e. screen elements that will change how they are drawn on different display sizes

- jQM Methods: i.e. functions that can be used to simplify mobile app code for common operations such as accessing data from remote sources, page navigation and effecting page transitions
- jQM events: events that handle occurrences are useful in mobile apps, such as orientation changes (turning your phone from portrait to landscape format), scrolling pages, swipes and taps (user-input events), requesting new pages from the server etc.
- Themes: i.e. sets of page element styles for different app appearances. The jQM website includes a "theme-roller" app so that you can create your own colour schemes and widget appearances – this is important if you wish to build pages to match a corporate look-and-feel (e.g. developers of a banking app would want to use the bank's own colour scheme, logos etc.)

Full references for jQuery and jQuery Mobile are available at their home websites:

jQuery Reference: http://jquery.com

jQuery API Reference: http://api.jquery.com

jQuery Mobile Reference: http://jquerymobile.com

jQuery Mobile API Reference: http://api.jquerymobile.com