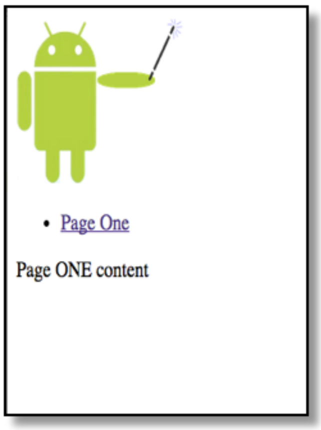
HTML5 and jQuery-Mobile Apps

In this practical, we will start with a simple HTML page structure and add Smartphone styling and application behaviour via a 3rd Party code library – jQuery Mobile. The final application can be downloaded and installed on an iPhone, Android or Win Mobile device) via a web server. In a later lab we will cover how to install it for offline operation.

Working methods

As you work on this lab, the main principle should be "test continuously". Every time you make a change to a line of code (HTML or otherwise), add a line or delete one, you should refresh the browser displaying the page to see the result. If you don't do this and then introduce an error, it will be much more difficult to find the error's source.

```
<!DOCTYPE html>
<html>
    <head>
    </head>
    <body>
        <img src="./img/Logo.png" width="150"
             height="120" alt="Logo"/>
        <ul>
            <li><a href="#one">Page One</a></li>
        </ul>
        <p>Page ONE content</p>
    </body>
</html>
```



Listing 1 and Figure 1: A simple web page and the page rendered in a browser.

Listing 1 shows the HTML of a very simple web page that adheres to HTML5 standards – in particular, the DOCTYPE attribute is simply 'html', which is an HTML5 requirement. We will consider this page as having two parts:

- a "HOME" section, which is the logo and the unordered list of links (currently containing a single href link), and
- a 'content' section (which contains a single paragraph element).

The first of these will become the 'home' page and the second will become a 'content' page once we incorporate the mobile framework. Figure 1 shows how the page should appear. To create the project:

1. Start a new empty project in WebStorm. Create the project in a folder on your H: drive or memory stick, and give it the name Lab1.
   Note - WebStorm is installed on lab machines, and is available as a download from https://www.jetbrains.com/webstorm/download/. See the ***Install WebStorm on your PC*** link on Moodle for instructions on how to install and register a copy on your home machine or laptop).
2. Add an HTML file to the project – call it ***lab1.html***. WebStorm will automatically add some boilerplate HTML code to this file. You should add all of the code inside the <body> tag in listing 1 to this
3. Add a new directory to the project (right-click the project title in the Project view and select New/Directory) – call it ***img***. Find the file ***logo.png*** on the Moodle site (in the Lab 1 folder) and download it to your machine. You can drag this image file from Windows Explorer to the ***img*** folder in the WebStorm project
4. Open the page in a web browser and make any necessary corrections.

Your page should appear as shown in Figure 1

2. Add jQuery Mobile declarations

A problem with the page built in step 1 is that it will not work particularly well in all mobile browsers. To demonstrate this:

1. Open the page in the Chrome web browser, and then open Chrome's developer tools (Ctrl+Shift+I).
2. Press the icon at the top of the developer tools window (Show/Hide Drawer) and click on the Emulation tab at the top of the drawer.
3. From the Model drop-down, select a an early iPhone model (iPhone 3GS, 4 or 5)

The emulated iPhone display should show the page with a tiny logo and very small print (see figure 2). We will fix this by using an app framework that optimizes a page for small-screen display.

jQuery is a set of libraries in Javascript that enhance the power of the language in various ways. **jQuery Mobile** adds some style elements to the core jQuery functionality to style apps automatically for a range of mobile platforms. Add the following statements to the document head section and refresh the page:

Figure 2: Listing 1 in a mobile browser

```
<meta name="viewport" content="width=device-width, initial-scale=1" />
<link rel="stylesheet"
      href="http://code.jquery.com/mobile/1.2.1/jquery.mobile-1.2.1.min.css" />
<script src="http://code.jquery.com/jquery-1.7.2.min.js"></script>
<script src="http://code.jquery.com/mobile/1.2.1/jquery.mobile-1.2.1.min.js">
</script>
<style type="text/css">.img {
         display: block; margin-left: auto; margin-right: auto;
}
</style>
```
Listing 2: HTML code to incorporate jQuery and jQuery Mobile

There are five statements in listing 2. See the end of this lab sheet for a short cut for inserting the first four of them. Note that <script> and <style> tags need to be closed (</script>, </style>) but the <meta /> and <link /> tags are self-contained:

1. A meta tag that sets up the page so that it displays at the full width of the device
2. A link to a CSS (Cascading Style-Sheets) style-sheet at the jQuery Mobile website
3. A script link to the jQuery main code library at the jQuery website
4. A script link to the mobile-specific libraries at the jQuery Mobile website
5. A simple style declaration to centre the logo image on the page.

The first three declarations are links to external pages – in a web app, this will turn out to be a problem since if the development machine is off-line the resources can't be loaded. That can be fixed by using local copies of the files – just download them when your machine is online and change the URLs to suit.

If you refresh the page now in the browser, you should find that the page display is improved – this is a consequence of the CSS in jQuery Mobile, and what it does for the <body> html element. However, to get the full benefit of jQuery Mobile (hereafter, jQM), you will need to add some more framework specific mark-up.

3. Add jQuery Mobile page-element definitions

We'll start by separating out the 'pages' of the application:

1. Surround each section with <div>…</div> tags to mark-off each main page in the <body> section:

```
<div data-role="page" id="home">
    <img src="./img/Logo.png" width="150" height="120"
alt="Logo"/>
    <ul>
        <li><a href="#one">Page One</a></li>
    </ul>
</div>
<div data-role="page" id="one">
    <p>Page ONE content</p>
</div>
```

Listing 3: Mark-up for a Page in a jQuery Mobile App

Note that these <div> elements each contain mark-up for a complete page. Each <div> has been given two attributes: a **data-role** attribute that indicates that the <div> is to be treated as a page, and an ***id*** attribute that we can use to identify the page. If you now refresh the page, you will find that you can use the link on the first page to move to the second page. The browser's back button is needed to return to the home page, but we'll fix that later.

2. We can add further mark-up to the page to define key sections of it. Each page-div can be divided into three sections by placing further div elements inside them. These will act as 'header', 'content' and 'footer' sections. The existing content in the home page should be inside the middle (content) div. Listing 4 shows the mark-up for the first (home) page. The second page mark-up should be similar and should follow this in the mark-up.

```
<div data-role="page" id="home">
    <div data-role="header">
        <h1>Home Page</h1>
    </div>
    <div data-role="content">
        <img src="img/logo.png" alt="Logo"
               width="150" height="120"/>
        <ul>
            <li><a href="#one">Page One</a></li>
        </ul>
    </div>
    <div data-role="footer" data-position="fixed">
        <h4>Home page footer.</h4>
    </div>
</div>
```

Listing 4: Better mark-up for the "home" page, with Header and Footer sections

You can now think of the first top-level <div> as being the 'home' page of the app, and the second (page "one") as a separate page. Refresh the page in the browser and click on the Page One link to test this.

3. To improve the look of the page-link in the home page, add attributes to the opening <ul> tag that surrounds the page link on the home page and the enclosed <li> tag:
The 'data-role' attribute changes the style of the link to a more suitable style for a home page, and the 'data-inset' attribute improves the link's position on the page. Again, test

the changes by refreshing the page and clicking on the Page One link

```
<ul data-role="listview" data-inset="true">
    <li><a href="#one">Page One</a></li>
</ul>
```

Listing 5: Styling a <ul> element as a "listview"

4. You will note that while the main page applied a nice appearance to the link to the second page, you need to press the browser's 'back' button to return to the main page. A finished web app has no browser back button, so we should fix this by adding an anchor (<a>) element to the second page's header as a back button:
   Refresh and test again – you should now find a 'Back' button has been automatically added to Page One of the app. Note that the page footer mark-up has also been added to; the data-position="fixed" attribute keeps the footer at the bottom of the display.

```
<div data-role="page" id="one">
    <div data-role="header">
        <h1>Page ONE</h1>
        <a href="#" data-rel="back" data-
icon="back">Back</a>
    </div>
    <div data-role="content">
        <p id="">Page ONE content</p>
    </div>
    <div data-role="footer" data-position="fixed">
        <h4>Page Two Footer</h4>
    </div>
</div>
```

Listing 6: Adding a "back" button to the page header

Exercises

Since all of the various changes we've made have been explained as we go along, you now ought to be able to add some more to the page without such detailed explanations:

1. Add another content page to the app. (a separate <div></div> element within the <body> element, with additional <div> elements inside it to define page section roles). For now, give it the heading "Page Two" and give it a suitable ID ('two' for example). Give its header section an appropriate style (h1, h2 etc.) and a back button. Add a page link to it from the home page (by simply adding another <li> element to the <ul> listview element, with a target href of "#two"). Add simple text content (inside suitable HTML paragraph styles like <h2>…</h2> for a header, <p>...</p> for simple text etc. *N.B. copy & paste is a big time-saver here.*
2. You can assign a data-transition attribute to a page-link (including 'back' links). Experiment with different types of page transitions in the list of links to application pages in the 'home' div. Possible options are "flip", "slide", "slideup", "slidedown", "pop" "turn", "flow", "fade" and "slidefade". Try all of these – remember you'll need to save the HTML code and refresh the browser page after each change.
3. It is possible to give page <div> elements a 'data-theme' attribute, which affects their styling. jQuery Mobile provides three possible data theme styles by default, named "a", "b" and "c". Try them and select your favourite for use throughout the app. You can add your own from the jQuery Mobile Theme-Roller page (http://themeroller.jquerymobile.com/)
4. Continuing as you did in exercise 1, alter the whole application so that it is a five-page app with one home page and four subsidiary content pages. Make the home page into a

series of links to content pages about the Programming for Mobile Devices module, with the structure shown below:

a. Home
    i. Link to Module Overview
    ii. Link to list of lectures
    iii. Link to list of labs
    iv. Link to a list of functions
b. Module Overview Page

    This should be a simple text page with a heading <h2>..</h2> and a couple of paragraphs of text <p>..</p>

c. List of Lectures

    This should be a simple HTML ordered list <ol>..</ol> with the titles of three lectures – "Introduction", "Web-App Development" and "Using RSS data feeds"

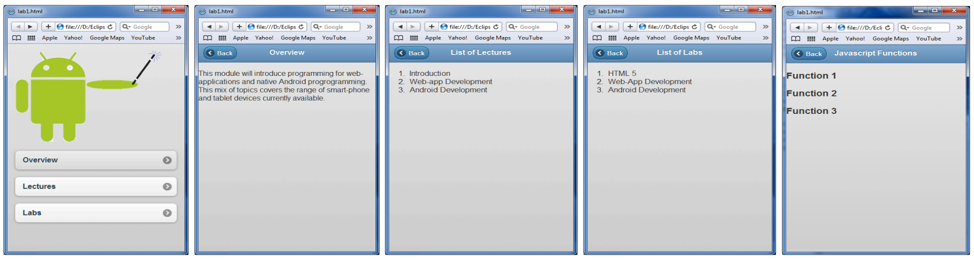
d. List of Labs

    As above, a simple list of three labs – "HTML 5", "Web-App Development" and "Using jGFeed"

e. Functions

    This page will be an entry point to a number of demonstration functions we will add to the app; i.e. a set of links to JavaScript code. This will eventually be another listview but for now some simple text will be fine

Note that where relevant you should rename pages and links to match with the application's content (e.g. rename the ID "one" and the link "#one" to "overview" and "#overview". Also note that once you have set up the first page structure, copy and paste is a good way of adding new pages (although you need to remember to change links and IDs).



This module will introduce programming for web-applications and native Android progrogramming. This mix of topics covers the range of smart-phone and tablet devices currently available.

1. Introduction
2. Web-app Development
3. Android Development

1. HTML 5
2. Web-App Development
3. Android Development

Figure 3: All five pages of the extended app.

4. Adding Javascript Functionality to the App

In the lecture this week you will have covered a small amount of Javascript, so at this stage we ought to incorporate some Javascript functions into the web-app just to see how it works. We'll add three simple functions to the page:

a. A simple function to put an alert message on the display
b. A function call that accesses some device information (the time of day) and displays it in an Alert
c. A function to access the geo-location functionality that is supported by HTML5

a. Displaying a simple message

Displaying a message in Javascript is normally very simple – just pass the message string, or a variable containing the message to the alert() function. Using this feature in jQuery Mobile is a bit more complicated since mobile devices usually don't support the alert() function. In addition, we also need to find a way to trigger the alert, and while jQuery Mobile provides an easy way to link between <div> pages, it doesn't make it so easy for Javascript.

There is a way to implement an 'alert' box in jQuery Mobile, but this is simply a variant of a standard <div> page (with data-role set to "dialog") so that it looks like one, and provides a built-in close button. The required mark-up is shown in listing 7, and we can add this at the end of the <body> section, after all the other pages.

```
<div data-role="dialog" id="alert">
    <div data-role="header" data-position="fixed">
        <h3><div id="alert-title"></div></h3>
    </div>
    <div data-role="content">
        <div id="alert-content"></div>
    </div>
</div>
```

Listing 7: Mark-up for an "Alert" dialog box – note, this will need some Javascript code to add content

Since in most situations, an alert box is usually a response to something that happens as a program runs, we need to trigger this from Javascript code. We can define a function to fill in the dialog details (i.e. a title and a message to appear in the dialog) – see listing 8.

Now when we need to open an alert box from Javascript code, we can write:

jqmAlert("Alert Title", "Alert Message");

Exercises

1. Add the alert-box mark-up shown in listing 7 to the HTML page (just before the closing </body> tag
2. Add a new text file to the Lab2 folder and call it ***lab2.js***. Add the following code to it:

```
function jqmAlert(title, message){
    // set up the 'alert' page..
    $("#alert-title").html(title); // Insert the title
    $("#alert-content").html(message); // Insert the dialog message
    $(".ui-dialog").dialog("close");   // Ensure dialog behaviour
    // Now open the page..
    $.mobile.changePage("#alert","pop", false, false);
}
```

Listing 8: Javasctipt code to activate the "Alert" dialog box

3. Insert a link to this file into a script tag in the HTML page (follow the format of the script tag used in Lab1)
4. Add a listview (<ul>…</ul>) to the <div> page you added in exercise 4 for a list of functions. Add a couple of <li> elements to use as links to Javascript functions calls; for example <li><a href="#" id="testAlert">Test Alert</a></li> (this would define an element called 'testAlert' that could then be attached to a Javascript function). You will need elements for Alert, Time and Location facilities
5. Define a testAlert() function that in turn calls on the jqmAlert() function, like this:

```
function testAlert() {
    jqmAlert("Hello", "A Hello Message");
}
```

Listing 9: Code to test the jqmAlert() function

Use ***$(document).ready()*** to bind this function to the appropriate *testAlert()* function – see the appendix about binding functions to the user-interface to see how to do this if you need to be reminded.

6. Save the JS and HTML files, refresh the browser and test the call to the ***jqmAlert()*** function

Adding more Javascript functionality

There are no limits to this; Javascript is a powerful language and we can use it to create fully featured applications than will almost rival apps built with the native code development kit; just bear in mind that while we can't do *everything* that a native app can do in Javascript, the apps we build will be cross platform.

We'll look at adding two simple facilities to this app, as mentioned earlier:

a. A time-of-day function, as a simple exercise in creating useful features in Javascript code
b. A geo-location function, which will demonstrate HTML5's geo-location capabilities

b. What time is it?

Javascript has access to a Date() class. This is used to create Date objects, which provide an API for date and time functions. See http://www.w3schools.com/jsref/jsref_obj_date.asp for a detailed description. In this example, we'll make use of simple time functions. We can create a time-of-day string using this function:

```
function timeOfDay() {
    var now = new Date(),
        tod = now.getHours() + ":" +
                  now.getMinutes() + ":" +
                  now.getSeconds();
    return tod;
}
```

Listing 10: A timeOfDay() function

1. Add the above Javascript code to the ***lab2.js*** file. We will also need to insert a call to it to display the result. The easiest way to do this is to add a second function that calls the ***timeOfDay()*** function and displays the result in a ***jqmAlert()*** call
2. To continue the principles that we were using in lab1, the best way to attach this function to the appropriate listview item is to give that item an ID attribute and then use ***$(document).ready()*** to bind the function. Try adding this now (an explanation is at the end of the lab)
3. The time will appear a bit odd sometimes, for example ***12:06:05*** would look like ***12:6:5***. You can fix this by inserting an additional zero before single digit values – something that you probably did in the HTML5 & Javascript module. Try to work out how to do this.

c. Getting the current location

Note that how (or whether) this works depends on the browser you use, the Computer's operating system and possibly even whether there is an 'r' in the month. There is a great deal of chat about using the HTML 5 geo-location functions online because they do not work equally across all platforms. Fortunately, they do seem to work robustly in mobile browsers, especially those that use Webkit as a core (iPhone, Android and the mobile Chrome browser). We have four things to do:

1. Determine if the browser supports geo-location, and if so…
2. Attempt to get the current location

3. Compile the location (or an error message) into a suitable format for display
4. Display the result.

The following (fairly horrible looking) function definition does all of that. It also introduces the idea of a ***lambda function***, which is a function with no name that is used by another function. Read the notes below it for an explanation:

```
function getLocation(){
    if(navigator.geolocation){
        navigator.geolocation.getCurrentPosition(
            // This function if the location is available...
            function (position) {
                var result = "GeoLocation Data\n";
                for (key in position.coords) {
                    result += key + ":" + position.coords[key] + '<p>';
                }
                jqmAlert("Location Data", result);
            },
            // This one if there has been an error...
            function (error){
                switch(error.code) {
                    case error.TIMEOUT:
                        jqmAlert('Error','Timeout')
                        break;
                    case error.POSITION_UNAVAILABLE:
                        jqmAlert('Error', 'Position unavailable');
                        break;
                    case error.PERMISSION_DENIED:
                        jqmAlert('Error', 'Permission denied');
                        break;
                    case error.UNKNOWN_ERROR:
                        jqmAlert('Error', 'Unknown error');
                        break;
                }
            });
    } else {
        jqmAlert('Error', "No mapping service!");
    }
}
```

Listing 11: A function to call the geo-location API in browsers

There are several significant features to the function in listing 11:

navigator.geolocation.getCurrentPosition() has two parameters passed to it – the first is a function that will be called on success (**function(position)..**). This function goes through all of the attributes of the return from **getCurrentPosition()** and adds them as name-value pairs as a paragraph to a string of text (result), which is displayed via our **jqmAlert()** function. The second function is called if geo-location failed (**function(error)..**); it interprets an error code and displays a suitable message

To add the location functionality to the web-app:

1. Add the code of the geo-location function to web-app.js. *Be very careful* as you enter the code - positioning of the brackets is crucial if it is to work
2. Bind the <li> item 'Location' to this function in $(document).ready()
3. Save (both files), refresh the browser and test.

Appendix/Notes

1. Binding functions to the user-interface.

In general, we can refer to an element that has an ID attribute in jQuery code using $("#id"). So, if we have a list item with an ID of "test", we can refer to it as $("#test"). Once we have a reference to an item, we can access all of the standard jQuery functionality for it. Here, the .bind() method is useful. If we wanted to bind our "test" element to a function called doTest() so that the function was called when we click on the item, use:

```
$("#test").on('click', doTest);
```

Finally, we can create this binding as the document loads into memory, using:

```
$(document).ready(function(){
        $("#test").on('click', doTest);
});
function doTest() { blah blah code code }
```

This method of binding has the advantage of not creating identifiers in the code that would only be used once (while binding the function to the event). It also makes the code work slightly faster.

2. jQuery Mobile Short Cuts in WebStorm.

WebStorm already includes loads of short-cut keypresses – for example, type h1 then tab to get <h1></h1> (and the same for any standard HTML tags). However, it also lets you define new short-cuts (it calls them "live templates"), and I have included the following ones in the installation in the labs. If you want to add these to a home installation, you can find the relevant file on Moodle, usually attached to the Installing WebStorm document – the file is called **jqm-templates.jar**, and can be incorporated into WebStorm using File→Settings→Import from the File menu. There is a list of them on the next page.

Short-cut	Purpose
jqm + Tab	Insert the jQuery Mobile and Viewport header into the page at the current cursor position
jqmpage + Tab	Insert a full jQuery mobile page of mark-up into the current cursor position (with default settings to be over-ridden as necessary
jqmlist + Tab	Insert mark-up for a jQuery Mobile listview at the current cursor position
jqmlistitem + Tab	Insert a new <li> (list-item) tag that includes an <a> (anchor) tag (useful for creating styled page-links)