Programming the Internet—Lecture Notes

Week 7: Advanced Topics

Introduction

Web programming is a vast topic. It is difficult to cover everything we want in one module. We have introduced the basics of HTML, CSS, JavaScript and PHP. That should have given you a good start towards learning the intricacies of web programming. This week you will briefly look at a range of advanced topics for which we shall give a brief overview. Some of these will be more relevant to you than others, depending on your work situation and the personal goals you have.

One of the discussion questions this week takes account of this and allows you to be selective in concentrating on the topics that interest you most. All of these subjects are accompanied by suggestions for further reading and provide opportunities for self-directed learning now and in the future. This will also be a further opportunity to develop your research skills in exploring these topics more deeply on your own. This week's topics are:

- AJAX
- jQuery
- What's New in HTML5
- CSS3
- The Open Web Application Security Project (OWASP)
- Mobile Web Programming

AJAX

By tradition, AJAX stands for Asynchronous JavaScript and XML. The first use of the term is attributed to Jesse James Garrett (2005). It is really a collection of existing techniques rather than a separate technology. Normally, when we click on a link or click a *Submit* button, our browser is sent a complete Web page by the server, even if it is only a slightly amended version of the previous page. AJAX provides a method of replacing part of the content of a Web page without having to download a complete page from the server.

It makes use of the JavaScript XMLHttpRequest object and the Document Object Model to target just part of a page and update its contents. An AJAX engine handles communications between the browser and the Web server to make this possible. Because the user interface is updated without reloading a complete page, changes appear much smoother and faster. This engine is not a separate piece of software but a series of functions in JavaScript written by the programmer to handle communication between browser and server. Think of it as a convenient concept to help us envisage what is going on.

The AJAX engine usually receives information from the server in the form of XML (Extensible Markup Language). This is a markup language highly suited

to describing the structure of data (such as that held in a database). It has strict syntax rules and allows programmers to define their own tags. It can be used as a language for writing particular markup languages. More details about XML can be found at W3C (2012).

Having received XML from the server, the AJAX engine then uses JavaScript and the Document Object Model to target the content in the parts of the Web page that need to be updated. Google Maps uses this approach to load a new section of a map.

Here is a simple example of AJAX in action. It writes the current time and date on the Web server to a field in an HTML form without the need to click a button. We have seen something like this in our study of client-side programming with JavaScript, but this provides a similar experience in interactions with the Web server. For this example, refer to the three files in the Code Examples zip file in this week's Learning Resources. Plenty of comments have been included in the code to explain what is going on. First the HTML:

```
<!DOCTYPE html>
<html>
<head>
<title> An Example of AJAX </title>
<meta charset="utf-8" />
<script type = "text/javascript" src="getTime.js"></script>
</head>
<body>

<form name="myForm">
Name: <input type="text" name="username" size=15
onChange="ajaxtime();" /> <br />
<!-- No submit button is necessary with AJAX. The event is used to
trigger a request to the server -->
Time: <input type="text" name="time" size=30 />
<!-- When the "username" field is changed and loses focus, the data
from the server will appear in the "time" field -->
</form>
</body>
</html>
```

Figure 1: getTime.html

Now the JavaScript:

```
function ajaxtime(){
var timeRequest;

timeRequest = new XMLHttpRequest();
/* timeRequest is a new instance of the XMLHttpRequest object */
/* We are now going to create a function to receive data from the
server */
timeRequest.onreadystatechange = function(){
if(timeRequest.readyState == 4 && timeRequest.status==200){
/* readystate == 4 indicates that the response is ready and status ==
```

```
200 means OK */
document.myForm.time.value = timeRequest.responseText;
/* Here we use the DOM to alter the value in the HTML form input box
called "time". We write the time and date details returned by the
server */
}
}
timeRequest.open("GET", "getTime.php", true);
/* This prepares our request to the PHP file on the server */
timeRequest.send(null);
/* This sends the request */
}
```

Figure 2: getTime.js

And finally the PHP:

```
<?php
echo date("D, d M Y H:i:s O");
/* This gives the time and date on the server.
The symbols explained:
D is a shortened version of the day in letters
d is the day of the month expressed as two digits (with leading
zeros)
M is a shortened version of the month in letters
Y is the year in four digit format
H is the hour in 24 hour format with leading zeros
i is the minute with leading zeros
s is the second with leading zeros
O is the time relative to GMT (UTC)
*/
?>
```

Figure 3: getTime.php

Chapter 16 of the module textbook is a good place to start your additional reading if this topic interests you. There are useful AJAX tutorials at Google Code University (2012) and W3 Schools (2012).

jQuery

We learned earlier in the module that there are a number of JavaScript libraries that can help us perform common tasks. Perhaps the best known of these libraries is jQuery. Here we shall take a quick look at how jQuery works.

All of jQuery's facilities are contained in an external JavaScript file. Reference to this has to be made in the `<head></head>` tags of our HTML file, just like the other JavaScript applications we wrote. There are two ways to do this: either the relevant file can be downloaded from the jQuery site (jQuery, 2012) and then uploaded to the Web server we are using, or reference can be made to the JavaScript file residing on some other Web site. Here we shall refer to the version on jQuery's own Web site. It will then be downloaded automatically on a temporary basis by our Web page when our HTML loads.

```
<script type = "text/javascript" src="http://code.jquery.com/jquery-
1.8.1.min.js"></script>
```

The *min* refers to the minified or compressed Production version of the file. This will be very difficult to read but takes up less memory. If you want to read the rather complicated source code, download the Development version, which is uncompressed and readable.

The jQuery file contains a lot of general functions which will require the programmer to write some additional JavaScript to specify exactly what he or she wants to do with these functions. Following our previous practice we shall put this additional code in an external JavaScript file. Let's call this *jqFile.js*. We shall therefore need another line of code in our HTML `<head></head>` tags. This second `<script>` reference **must be below** the other line of code referring to *jquery-1.8.1.min.js*. This is because the functions in *jqFile.js* are calling the functions in *jquery-1.8.1.min.js*, and the browser will be confused if we have not previously told it where to find the jQuery library. This second line of code will read:

```
<script type = "text/javascript" src="jqFile.js"></script>
```

One of the peculiarities of jQuery is that it passes some of this additional code that we programmers write to a jQuery function in the form of a function with no name. This is known as an 'anonymous function'. Here is an example of the jQuery *$(document).ready* function, which waits until the complete HTML page has loaded before it runs our JavaScript code. This is useful because it means all of the page is available to be accessed through the DOM before any JavaScript executes.

```
$(document).ready(function() {
/* Our code is inserted here */
});
```

The syntax is quite peculiar and getting all those brackets and semi-colons right can be challenging. There is therefore a learning curve in adopting jQuery. A key part of using JavaScript is its ability to identify elements in a Web page. Here is how jQuery does this using the element's *id*. The first line is orthodox JavaScript, and the second is the jQuery equivalent.

```
var mysurname = document.getElementById("surname");
var mysurname = $("#surname");
```

In using *#surname*, jQuery follows the CSS syntax. If we want to identify all the *h1* elements by their tag name, we should do something like this (orthodox JavaScript first, followed by the jQuery equivalent):

```
var h1collection = document.getElementsByTagName("h1");
var h1collection = $("h1");
```

jQuery has a whole range of functions for selecting elements on a page, e.g.

```
$('input[type="text"]')
```

would identify all text boxes and

```
$('a[href$=".pdf"]')
```

would identify all the links pointing to pdf files.

Behind the scenes, jQuery converts the DOM into its own proprietary format, and the jQuery functions thereafter do not operate directly on the DOM. However, there are jQuery functions to do most things we would do if we were using orthodox JavaScript directly on the DOM. There is however, a requirement to learn jQuery's own notation system to make these functions work.

Some third parties write plug-ins for jQuery. These are additional pieces of code, usually contained in separate JavaScript files, which work with jQuery to make it more useful. One example is the Validation plugin of Jörn Zaefferer (2012), which helps with the validation of HTML form fields. Another such plugin, goMap, was written by Jevgenijs Shtrauss (2011) and helps in placing a Google Map on our Web page.

We shall finish this brief introduction by including a jQuery application that makes use of the goMap plugin. This example is inspired by an example in a book on both JavaScript and jQuery which is recommended for further reading (McFarland, 2012). This example uses the latitude and longitude of the Laureate office in Amsterdam to produce a Google map of the area. To find the latitude and longitude of any street address, visit iTouchMap (2012).

```
<!DOCTYPE html>
<html>
<head>
<title> jQuery Google Map </title>
<meta charset="utf-8" />
<script type = "text/javascript" src="http://code.jquery.com/jquery-
1.8.1.min.js"></script>
<script type = "text/javascript"
src="http://maps.google.com/maps/api/js?sensor=false"></script>
<script type = "text/javascript" src="jquery.gomap-
1.3.2.js"></script>
<!-- In this case we have uploaded goMap to the Laureate Web server
-->
<script type = "text/javascript" src="googleMap.js"></script>
<link href="googleMap.css" type="text/css" rel="stylesheet" />
</head>
<body>
<div id = "mapSpace">
<!-- An empty <div> is provided so that the Google map can be
displayed here. The dimensions are defined in the CSS file -->
</div>
</body>
</html>
```

Figure 4: googleMap.html

```
#mapSpace {
width: 760px;
height: 400px;
}
```

Figure 5: googleMap.css

```
$(document).ready(function() {
$('#mapSpace').goMap(
{
latitude: 52.302643,
longitude: 4.953766,
/* The location of the Laureate office in Amsterdam */
zoom: 17,
/* The larger the zoom figure, the more detail is shown */
scaleControl: true
/* Shows the scale of the map in the bottom left hand corner */
});
});
```

Figure 6: googleMap.css

Figure 7: Output from the three googleMap files

What's New in HTML5

We have earlier touched on a few of the new advanced features of HTML5. Here we can introduce a few more. They come with the usual warning that browser support is currently very patchy for HTML5; however, the example code provided has been found to run successfully on Google Chrome.

Sometimes it will be useful to test whether a browser supports a new feature in HTML5 or CSS3 (see the next section of these notes) before deciding what actions should be performed. There is a useful tool called Modernizr (2012). It is downloaded as a JavaScript file (*modernizr.js*) and referenced in the `<head></head>` tags of our HTML5 file. To test whether the HTML5 local

storage facility is available in our browser, for example, we should then say something like this:

```
if (Modernizr.localstorage) {
// Some code that takes advantage of local storage
}
else {
// Some alternative method of storing information, such as cookies
}
```

HTML5 Audio and Video

Historically, browsers have had to use third party plug-ins such as Adobe Flash to play audio and video files. HTML5 provides native support for these files. However, different browsers may support different file formats, and you may therefore have to offer several versions of a video. The standard does allow for *controls* to be displayed in the browser, covering functionality like play/pause and volume.

Here is some code to illustrate the use of these new tags.

```
<!DOCTYPE html>
<html>
<head>
<title> Demonstrating HTML5 Video and Audio </title>
<meta charset="utf-8" />
</head>
<body>
<h2>Tim Berners Lee talking on The Future of the Web at the <br
/>University of Oxford in England on 14 March 2006</h2>
<h3>Video (MP4) and Audio (MP3) versions  </h3>
<video src = "20060314_139_small.mp4" controls> </video>
<audio src = "20060314_139.mp3" controls> </audio>
</body>
</html>
```

Figure 8: Berners-Lee.html

Here is what the page looks like in Google Chrome. The video control is on the left, and the audio control is on the right. The MP4 and MP3 files have not been included in our zip file, but they can be obtained from the Oxford Internet Institute (2006).

Figure 9: Output from Berners-Lee.html

HTML5 Local Storage

We have looked at the use of cookies for storing data on the client's computer. They require a Web server script to set the cookie, and storage is limited to about 4000 bytes. HTML5 introduces a local storage facility that can usually hold up to 5MB of data on the client's machine, and the process can be implemented by client-side JavaScript.

It uses a *localStorage* object for data that persists between browser sessions and a *sessionStorage* object for data that will be deleted when the current browser session ends, e.g. when the browser is closed. Here is some code illustrating the use of *localStorage*. All data is stored as strings. To simplify the example, it does not include any error checking, but by now you should be confident enough to add that yourself.

```
<!DOCTYPE html>
<html>
<head>
<title> Demonstrating Local Storage </title>
<meta charset="utf-8" />
<script type = "text/javascript" src="localStorage.js"></script>
</head>
<body>
<h2>localStorage Example</h2>

Email address: <input type = "text" name = "email" id = "email" size
= "20"><br />
<input type = "button" id = "storeEmail" value = "Store the email
address" onclick = "storeEmail()" /><br />
<input type = "button" id = "retrieveEmail" value = "Retrieve the
email address" onclick = "retrieveEmail()" /><br />
<input type = "button" id = "deleteEmail" value = "Delete the email
address" onclick = "deleteEmail()" />

</body>
</html>
```

Figure 10: localStorage.html

```
function storeEmail() {
var email = document.getElementById("email").value;
localStorage.setItem("emailaddress", email);
/* "emailaddress" is the identifier/variable name we shall use in
local storage */
document.getElementById("email").value = "";
}

function retrieveEmail() {
var storedEmail = localStorage.getItem("emailaddress");
/* We retrieve the email address from local storage using the
identifier/variable name "emailaddress" */
alert("The stored email address is " + storedEmail);
}

function deleteEmail() {
localStorage.removeItem("emailaddress");
alert("Now click the button marked Retrieve the email address to
check that it has been deleted.");
}
```

Figure 11: localStorage.js

Displaying Code and Tooltips in HTML5

We shall now look at a number of new tags introduced by HTML5. The first group of tags mainly help in the display of code on a Web page. They are:

- `<code></code>` to display program code in a different font
- `<var></var>` to highlight a reference to a variable in some text
- `<samp></samp>` to highlight a reference to sample program input and output values
- `<address></address>` to highlight an address (often a URL). The browser may place the text between `<address></address>` tags on a separate line.

The second set of tags we are going to look at will display a tooltip (a little pop up box) when the mouse hovers over the text enclosed in these tags. It has always been possible to implement this in JavaScript, but now there is native HTML5 support for tooltips. The new tags which support tooltips are as follows:

- `<abbr></abbr>` an abbreviation
- `<cite></cite>` a citation or reference to some source
- `<dfn></dfn>` a definition
- `<kbd></kbd>` where the user should enter something at the keyboard

The text to be displayed in the tooltip is contained in a *title* attribute. This is widely supported by browsers. Here is some code that illustrates the use of all of these tags:

```
<!DOCTYPE html>
<html>
<head>
<title> Displaying Code and Tooltips </title>
<meta charset="utf-8" />
</head>
<body>
<h2>Displaying Code and Tooltips</h2>
<p><abbr title = "World Wide Web Consortium">W3C</abbr> is
responsible for setting standards on the Web.</p>
<p><address>It can be contacted at www.w3.org/</address></p>
<p>Another of its Web pages can be found at <cite title = "W3C (2012)
HTML & CSS [Online]. Available from:
http://www.w3.org/standards/webdesign/htmlcss (Accessed: 8 September
2012).">W3C (2012)</cite>.   </p>
<p>Here is some sample <dfn title = "A client-side scripting language
used in Web programming">JavaScript</dfn> code:  </p>
<code>
function hello()
{
var hello = "Hello ";
echo(hello + "world !");
}
</code>
<p>In the above code, <var>hello</var> is an example of a
variable.</p>
<p>In the text box below, examples of valid input would be
<samp>25</samp> and <samp>42</samp>.</p>
<p>Your age: <kbd title = "Don't lie about your age !"><input type =
"text" name = "age" id = "age" size = "5"></kbd></p>
</body>
</html>
```

Figure 12: codeTooltips.html

Geolocation in HTML5

Geolocation is the ability to find the user's location. This location is approximate, using things like the IP address or mobile phone mast triangulation. The HTML5 specification supports the Geolocation API (Application Programming Interface), which means that it works in co-operation with some JavaScript functions. The user's consent is required before these details are sent to a server that requests them. Therefore, in the example we have provided, don't forget to click the 'Allow' button which will be displayed after you have clicked on 'Find your location'. More sophisticated applications can use the location information to bring up a Google Map of your location.

Here is the HTML and the JavaScript:

```
<!DOCTYPE html>
<html>
<head>
<title> Demonstrating Geolocation </title>
<meta charset="utf-8" />
<script type = "text/javascript" src="geolocation.js"></script>
```

```
</head>
<body>
<h2>Geolocation Example</h2>

<p><input type = "button" id = "findLocation" value = "Find your
location" onclick = "useGeolocation()" /></p>

Location: <input type = "text" name = "location" id = "location" size
= "50"><br />
</body>
</html>
```

Figure 13: geolocation.html

```
function useGeolocation() {
navigator.geolocation.getCurrentPosition(findLocation);
/* The parameter is a function to which your location details will be
passed, via findLocation's position parameter */
}

function findLocation(position) {
document.getElementById("location").value = "Latitude: " +
position.coords.latitude + ". Longitude: " +
position.coords.longitude;
}
```

Figure 14: geolocation.js

CSS3

Most of the Cascading Style Sheets we have used so far comply with the CSS 2.1 standard, which is widely supported by browsers. CSS3, which is not yet an established standard, introduces some new functionality, but most of these are not yet universally implemented by modern browsers. We shall see that, in some cases, browsers have implemented CSS3 quite differently, to the extent that the features will only work by placing a browser prefix in front of them.

W3C has divided the CSS3 specification into different ‘modules’, and progress on different aspects of the proposed standard is proceeding at different speeds. The topic of CSS3 could occupy a module all of its own, but here we have selected just a few of its functionality to give an idea of what it can do.

CSS3 Shadows

CSS3 provides the capability for a shadow effect (a drop shadow) to be applied to both text and boxes. A text shadow requires two coordinates: x, the horizontal distance of the shadow from the text, and y, the vertical distance. A third (optional) parameter is the blur radius. This specifies the extent of the blurring. A fourth (also optional) parameter is the colour of the shadow.

Box shadows are similar, except that a positive x value will place a shadow to the right of the box, whereas a negative value will place it to the left. With the y value, positive = below the box, negative = above the box. This functionality is best illustrated by an example:

Figure 15: Output from shadows.html and shadows.css

And here is the code that produces that effect:

```
<!DOCTYPE html>
<html>
<head>
<title> Demonstrating Shadows </title>
<meta charset="utf-8" />
<link href="shadows.css" type="text/css" rel="stylesheet" />
</head>
<body>

<div id = shadowBox>
<h1>Me and My Shadow</h1>
</div>

</body>
</html>
```

Figure 16: shadows.html

```
h1 {
color: black;
font-family: Arial, sans-serif;
font-weight: bold;
font-size: 400%;
text-shadow: 3px 3px 3px gray;
}

#shadowBox {
width: 50%;
color: black;
background-color: cyan;
box-shadow: 5px 5px 5px gray;
}
```

Figure 17: shadows.css

Rounded Corners in CSS3

We saw an example of boxes with rounded corners in Week 2. Now a few words of explanation. CSS3 allows this functionality with the *border-radius* property. Technically, each corner of a box is regarded as a quarter of an elliptical figure (oval or circular). The x-axis is the horizontal radius from the side to the centre of the elliptical figure. The y-axis is the vertical measurement. That is why the key CSS property is *border-radius*. The quarter of the elliptical figure is regarded as 'regular' (having a regular curve) if the x and y measurements are the same. Each corner can be set differently with *border-top-right radius*, *border-top-left radius*, *border-bottom-right radius*, and *border-bottom-left-radius*. However, they are usually the same and the curve is regular, so *border-radius* with one value will suffice.

Rounding applies only to the background and border of a box. Text positioned precisely in the top left corner with absolute positioning could possibly extend beyond the rounded corner of the border. In this slightly different version of our *shadows.html* program, the *border-radius* figure has been adjusted to make the box into a circle (almost). Experiment with the radius figure to produce different effects. The Candidate Recommendation, which covers rounded corners, can be found at W3C (2012b).

Figure 18: Output from rounded.css

The HTML is almost identical to *shadows.html*, but here is the CSS code. Note that a *padding* value has been introduced to make sure the text is within the circle.

```
h1 {
color: black;
font-family: Arial, sans-serif;
```

```
font-weight: bold;
font-size: 150%;
text-shadow: 3px 3px 3px gray;
}

#roundBox {
border-radius: 150px;
padding: 30px;
width: 240px;
height: 240px;
text-align: center;
color: black;
background-color: cyan;
box-shadow: 5px 5px 5px gray;
}
```

Figure 19: rounded.css

Column Layout in CSS3

The content of a Web page can currently be set out in columns but only with some difficulty or by employing tables, which should not ideally be used to design a page. Remember that there are benefits in keeping content and design separate.

CSS3 introduces an extremely useful way of easily dividing content (predominantly text) into columns. This can be done quite easily. This functionality is supported by Opera 11.10 or later and IE10. Unfortunately, the WebKit browsers, Chrome and Safari, together with Firefox, have implemented this functionality in a non-standard way. There is a Candidate Recommendation at W3C (2012c). Columns can only be made to work properly across a range of browsers by also using the *–webkit-* (Chrome and Safari) or *–moz-* (Firefox) prefix which means we have to write more code than we should like. Be aware that some browsers may not implement all of the column features (such as *column-span* in Firefox). This is one of the abiding frustrations of using CSS3 before full browser support is available.

A similar CSS3 feature is the grid layout, which will enable content to be ordered in rows and columns in a very flexible way. At this time it is only supported by IE10. The Working Draft of the documentation can be read at W3C (2012d).

The following example of multiple columns uses comments, which should guide the reader about what is going on.

```
 <!DOCTYPE html>
<html>
<head>
<title> CSS Columns </title>
<meta charset="utf-8" />
<link href="columns.css" type="text/css" rel="stylesheet" />
</head>
<body>
```

```
<div id = columnBox>
<h1>CSS Columns - the headline spans all columns</h1>
<p>Lorem ipsum dolor sit amet, consectetur adipisicing elit, sed do
eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim
ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut
aliquip ex ea commodo consequat. Duis aute irure dolor in
reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla
pariatur. Excepteur sint occaecat cupidatat non proident, sunt in
culpa qui officia deserunt mollit anim id est laborum. </p>
<p>Lorem ipsum dolor sit amet, consectetur adipisicing elit, sed do
eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim
ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut
aliquip ex ea commodo consequat. Duis aute irure dolor in
reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla
pariatur. Excepteur sint occaecat cupidatat non proident, sunt in
culpa qui officia deserunt mollit anim id est laborum. </p>
<p>Lorem ipsum dolor sit amet, consectetur adipisicing elit, sed do
eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim
ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut
aliquip ex ea commodo consequat. Duis aute irure dolor in
reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla
pariatur. Excepteur sint occaecat cupidatat non proident, sunt in
culpa qui officia deserunt mollit anim id est laborum. </p>
<p>Lorem ipsum dolor sit amet, consectetur adipisicing elit, sed do
eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim
ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut
aliquip ex ea commodo consequat. Duis aute irure dolor in
reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla
pariatur. Excepteur sint occaecat cupidatat non proident, sunt in
culpa qui officia deserunt mollit anim id est laborum. </p>
<!-- Lorem ipsum dolor is a standard used when we want to insert
dummy text. It is Latin from the work of Cicero -->
</div>

</body>
</html>
```

Figure 20: columns.html

```
#columnBox {
-webkit-column-count: 3;
/* For the WebKit browsers - Chrome and Safari */
-moz-column-count: 3;
/* For Firefox (Mozilla) */
column-count: 3;
/* The number of columns. "column-width" can also be set if required
*/
-webkit-column-rule: 1px solid gray;
-moz-column-rule: 1px solid gray;
column-rule: 1px solid gray;
/* This will insert lines between columns. It is a shorthand method
of setting "column-rule-width", "column-rule-style" and "column-rule-
color" */
/* By default the gap between columns is 1em. If a different gap is
required we can use e.g. "column-gap: 2em" */
width: 75%;
border-radius: 25px;
padding: 10px;
color: black;
```

```
font-family: Arial, sans-serif;
background-color: cyan;
}

#columnBox p {
/* Applies to all paragraphs of the div called columnBox */
margin-top: 0;
margin-bottom: 0;
/* Avoids blank lines between paragraphs */
text-indent: 1em;
/* The start of each paragraph will be indented */
text-align: justify;
/* Text will be justified i.e. both left and right edges of the text
will be aligned */
}

#columnBox h1 {
-webkit-column-span: all;
-moz-column-span: all;
column-span: all;
/* The headline will extend across all columns */
color: blue;
font-family: Arial, sans-serif;
font-weight: bold;
font-size: 150%;
}
```

Figure 21: columns.css

This is how the code looks in Google Chrome:

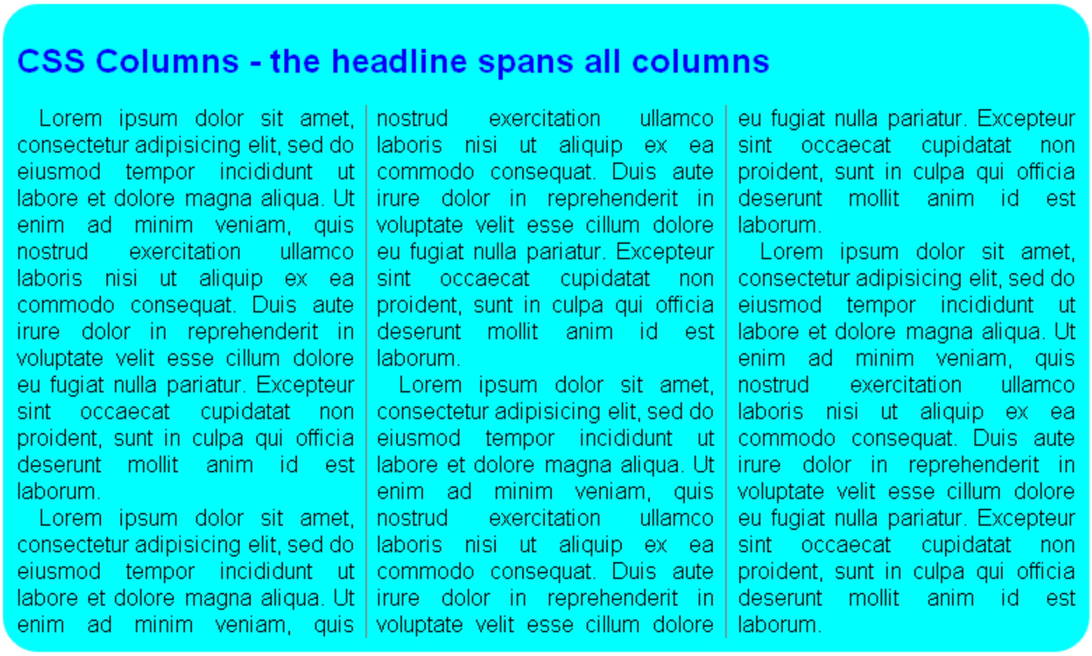

Figure 22: Output from columns.html and columns.css

CSS3 Transition Function

Currently, when an HTML element changes as a result of some event, the change normally takes place immediately. The new transition functionality in CSS3 provides for this change to take place gradually. Because of patchy browser implementation, everything may not work as intended, but transition may well be important in the near future. Here is some code that shows what it is intended to do.

```
<!DOCTYPE html>
<html>
<head>
<title> CSS Transition </title>
<meta charset="utf-8" />
<link href="transition.css" type="text/css" rel="stylesheet" />
</head>
<body>

<h1>Change Me</h1>

</body>
</html>
```

Figure 23: transition.html

```
h1 {
color: black;
font-family: Arial, sans-serif;
font-weight: bold;
font-size: 150%;

transition-property: color, font-size;  /* The colour and the size of
the headline text will change when the mouse hovers over it */
transition-duration: 5s;  /* The transition will take 5 seconds  */
transition-timing-function: linear; /* The transition will proceed at
the same pace throughout. The default is "ease", which starts slowly,
accelerates, and then finishes slowly */
}
/* A shorthand version of this would be:
transition: color 2s linear 0;
The last value (0) is the delay before the transition starts
(transition-delay: 0;). Zero is the default value.
You are warned that these effects may not be implemented as intended
by a particular browser */

h1:hover {
color: blue;
font-size: 400%;
}
```

Figure 24: transition.css

The Open Web Application Security Project (OWASP)

The way we write our Web programs can create security vulnerabilities that can be exploited by malicious users. The Open Web Application Security Project (OWASP) is an open source project that aims to improve Web programming standards to reduce security risks (OWASP, 2012a). Its methods have been endorsed by the National Security Agency (NSA) in the United States (NSA, 2012).

You will probably learn something of network security elsewhere in your studies which will involve things such as firewalls and anti-virus software. However, if our Web programs have inherent security weaknesses, the network will be undermined. Loss of public confidence in sites that have been 'hacked' in some way can result in serious business losses. Keeping the site safe is just as much the programmer's responsibility as it is that of the network administrator.

OWASP has launched a number of initiatives to make Web programmers more aware of security issues. Although not all students will be responsible for writing secure code, it is important that they are at least aware of the issues as they may have, or soon take on, managerial responsibilities for overseeing Web developers.

A key OWASP service is to maintain a list of the 'Top ten Web application security risks' (OWASP, 2008). This is often referred to simply as the 'Top Ten'. The two most serious risks are covered in the next two sections.

OWASP and Injection

One of the most common examples of injection is where user-supplied information contains malicious SQL commands which extract information from our database to which the user is not entitled. Someone could, for instance, type SQL into an HTML text box in circumstances where we were only expecting something innocuous such as a name or address. The malicious user could then inject code into a query string. This is often used to steal confidential information, but it can equally be employed to change or delete data in our database.

One solution is to add backslashes before escape characters in user input that are necessary for the construction of SQL queries, such as quotation marks. This means they cannot be used for injection attacks. With PHP we can say:

```
mysql_real_escape_string($Username)
```

There are, however, more sophisticated techniques to combat injection, and these are covered in one of OWASP's 'cheat sheets' (OWASP, 2012b). In PHP, the use of prepared statements can combat injection (PHP Group, 2012).

Cross-Site Scripting (XSS) with OWASP

An example of this problem would be where a user is induced to click on a link that includes malicious JavaScript code. That code is then downloaded to the browser and executes there. This malicious code could be designed to capture keystrokes or steal a session variable. Because the code is running in the browser, it is likely to have the same permissions as the user of that browser. More details about this vulnerability can be found at OWASP (2012c).

There are three useful OWASP videos on YouTube about these security risks, and they can be accessed from OWASP (2012e).

Another of OWASP's contributions to application security has been the Enterprise Security Application Programming Interface (ESAPI) (OWASP, 2012d). This interface includes a set of code libraries for different programming languages designed to make it easier to produce secure code.

OWASP and SAMM

According to its Web site, 'the Software Assurance Maturity Model (SAMM) is an open framework to help organizations formulate and implement a strategy for software security that is tailored to the specific risks facing the organization' (OpenSAMM, 2012a). This is another OWASP project, and it is designed to evaluate an organisation's existing Web software security practices. It looks at business functions under the headings of governance, construction, verification and deployment and defines a number of security practices for each one.

Each security practice can be at one of four levels, depending on how mature the organisation's security arrangements are. At each level SAMM defines an objective, activities, results, success metric, costs, personnel and related level. The objectives provide building blocks for a security assurance programme. SAMM also provides an assessment worksheet for each security practice, which helps create scorecards that assess the organisation's progress over a period of time.

There are Roadmap templates for typical kinds of organisations such as financial services and government. SAMM has been released under a Creative Commons Attribution Share-Alike license, which means that, unlike other assurance models, it can be implemented without excessive costs. A presentation giving an overview of SAMM can be downloaded from OpenSAMM (2012b).

Mobile Web Programming

As Web programmers targeting mobile users, we shall be producing two kinds of applications: Web applications and native applications.

Web applications run in a browser and are written using HTML (or XHTML), CSS and JavaScript and are, therefore, independent of any particular device. Mobile browsers are less likely to tolerate sloppy HTML code, so the good XHTML habits you have been acquiring in writing HTML5 will be very useful when developing Web applications for mobile devices.

Mobile browsers often provide better support for new HTML5 features than desktop browsers. This is partly because Android, iPhone and modern Blackberry phones all use browsers based on the WebKit browser engine. A browser engine is that part of the software that actually displays or renders code on screen; browsers with the same engine will behave in similar ways. Google Chrome and Safari browsers for the PC also use WebKit (2012). Other browser engines are Gecko (Firefox), Trident (IE) and Presto (Opera).

Native applications are those we refer to when we talk about ‘downloading an app’. They are specific to a particular device and can take advantage of local features such as the phone’s camera and location. They are typically written in Java (Android and Blackberry), Objective-C (iPhone) or C++ (Symbian/Nokia). It is expensive to have to produce different applications for different kinds of phone and requires the mastery (or ‘buying in’) of a wider range of programming skills. A compromise is to use a solution such as PhoneGap (2012), which allows the programmer to write a Web application in HTML5, CSS and JavaScript. This code is then combined with PhoneGap code that can access some of the phone’s native facilities, and the result is a hybrid application. In what follows we shall concentrate on Web applications.

There is greater variety in mobile phones’ capacities to use the Internet than we find in PCs. Those with the greatest capability are referred to as ‘smartphones’ although there is no clear cut definition of the term. In general, they are more likely to have an operating system that can run apps and have better facilities for accessing the Web through WiFi or a mobile service provider’s 3G network. Smartphones also tend to have full QWERTY keyboards, often on screen, and they are more likely to be able to receive and send e-mail as well as texts. The main lesson for programmers to remember is that we cannot assume that all mobile phones can take advantage of all the coding features we include in our applications. DeviceAtlas (2012) provides a wealth of information about which mobile devices support which features.

A good principle to follow in mobile Web programming is that ‘small is beautiful’. Some phones have limited processing power and have only slow Internet connections. Users often have to pay significant charges to download data. Battery life and memory are limited on mobile devices. Because of this, small Web pages with few or ‘lightweight’ images will provide a better viewing experience for a greater number of users. A benefit of external JavaScript and

CSS files is that they can be cached by the mobile device so that they do not need to be downloaded when needed again.

Different mobile browsers tend to dominate in different parts of the world, so we need to check carefully where our main markets are. StatCounter (2012) compiles useful statistics for the position of different browsers worldwide and in particular regions. Note how very influential the Opera browser is in Asia and Africa. Many of these phones are running Opera Mini, which is a proxy-based browser (Opera, 2012). It is designed for phones with small screens and limited bandwidth. All communications with the Web go through Opera's server, and then JavaScript is executed on the Opera server, not the phone. In modern terminology we might say that much of the processing takes place 'in the cloud'. Phones running Opera Mini cannot run AJAX, and only a limited number of events will work as expected (load, unload, click, change and submit).

The ideal approach in mobile Web programming is to employ *progressive enhancement* whereby those who can make use of the newer facilities can do so, but where those who cannot are not prevented from carrying out essential functions. If you want to maximise access to your Web site, do not rely on people having to be able to use the latest technology. W3C (2008a) has defined some Web best practices. It also defines those CSS facilities that are likely to be supported by most mobile browsers (W3C, 2008b).

It is difficult to type long URLs in phones without full QWERTY keyboards. It is, therefore, worth considering the use of URL shortening sites such as kwz.me. The full Web address is typed into their Web page by the programmer and a short version of this address is returned to us. Any user typing the short address will be redirected to our site.

Tables are not highly recommended on mobile sites because some browsers try to create linear (single column) versions of them automatically. Many mobile browsers do not support more than one window being open at a time, so do not assume they can open a new window. Bear in mind that most mobile phones are operated via touch screens. The tip of the finger is about one square centimetre; therefore, buttons and links that have to be pressed need to be sufficiently wide or separated from the next selectable element in order to avoid mistakes. This is sometimes referred to as the 'fat finger problem'.

We have already introduced some JavaScript libraries. These libraries can be important in building complex applications for mobile devices. All those 'gestures' (taps, swipes and pinches) on mobile touch screens are events which can be captured by JavaScript. Only a few mobile browsers do not support JavaScript. jQuery Mobile (2012) was developed to work with the core functions of jQuery. It is aimed largely at those developing for the more sophisticated touch screen devices. Sencha Touch (2012) and iUI (2012) are mobile libraries similar to jQuery Mobile.

The Viewport

In one of the code examples we present later in this section, we employ the concept of the viewport (Apple, 2012). In most cases the viewport can be thought of as the browser window. The width of this viewport is measured in CSS pixels, also known as logical or reference pixels. This is not always the same as the number of device pixels (points on the screen) that define a screen's resolution. On an iPhone, for instance, 2 device pixels would normally equal 1 CSS pixel. On a smartphone with a lower resolution, the ratio might be 1.5:1.

We can set the width of the viewport (window) to the viewable width of the device by using *device-width* like this:

```
<meta name = "viewport" content = "width=device-width,initial-
scale=1.0" />
```

The *initial-scale* value of 1.0 means that the page will not initially be zoomed in or out. Both attributes will here be used in support of CSS media queries, like in the following code:

```
<link href="CA-2cols.css" type="text/css" media="all and (min-
width:600px)" rel="stylesheet"  />
```

This is saying in the *media* attribute that the named style sheet (*CA-2cols.css*) will apply to all kinds of devices which have a minimum viewport width of 600 CSS pixels. Why all devices? We could have said that the device was *handheld* or *screen*, for example. These are examples of CSS media types. This functionality has been around for some time, but it is poorly supported by browsers. The reference to *min-width: 600px* is an example of a CSS media query. Such queries are well supported.

Style and Content Adaptation

When a user visits our Web page, we do not usually know in advance what kind of device he or she is using. This device could be a PC, a mobile phone or a tablet (and tablets present their own challenges which we cannot adequately explore here). We shall often want to adapt the content that is presented to the user depending on the kind of device being employed. There are two main ways of doing this.

In style adaptation, we send each user the same page but try to adapt its appearance using CSS to the device he or she is using. In the example below, we change the number of columns depending on the CSS pixel width of the device's viewport. Mobile phones are more suited to a linear, one column format. With content adaptation, as its name suggests, we send different content to the user depending on the device he or she is using. This content might contain fewer and smaller images, for instance, in the version sent to mobile phones.

All browsers send a *user agent* string to Web servers when requesting a page. This string contains information about the browser and the operating system on the client machine. If you want to view the information sent by your own browser to a Web site, visit Archer (2012) and the page will echo your information back to you. Sometimes the information is difficult to decipher, but the *user agent* string can be used to make, at worst, a very good guess about the kind of device being used. Fortunately, there are ready-made PHP scripts that can send a *user agent* string for us.

We are here going to use an open source program called *php-mobile-detect* (2012). It is described as a lightweight solution and is downloaded in the form of a single PHP file (*Mobile_Detect.php*) which can then be placed on our server. In this example of content adaptation, a different file is sent to the user depending on which device he or she has, but this could have been done dynamically with different PHP *echo* statements in an *if* statement being used to alter a page's content. If a more heavyweight device detection solution is required, then we can look at something like WURFL (ScientiaMobile, 2012). It is no longer open source, but there is a free version.

Firstly, we are going to introduce a simple application that presents the layout of our content in different ways depending on the width of the device's viewport. We are going to do this with a few lines of HTML and three CSS files. Our objective is to present text divided into different numbers of columns depending on a device's viewport width in pixels. Adjust the numbers as you see fit.

We shall reuse most of the code in the *columns.html* example introduced earlier in these notes. This time, the file will be called *styleAdaptation.html*. The full code is included in the Code Examples zip file in this week's Learning Resources, but as we are only changing material within the `<head></head>` tags of *columns.html*, only that part of our new file is reproduced below:

```
<head>
<title> Mobile Content Adaptation </title>
<meta charset="utf-8" />
<meta name = "viewport" content = "width=device-width,initial-
scale=1.0" />
<link href="CA-base.css" type="text/css" rel="stylesheet" />
<link href="CA-2cols.css" type="text/css" media="all and (min-
width:600px)" rel="stylesheet"  />
<link href="CA-3cols.css" type="text/css" media="all and (min-
width:800px)" rel="stylesheet"  />
</head>
```

Figure 25: Part of styleAdaptation.html

It can be seen that we have defined the width of the viewport as the width of the device's screen (*width=device-width*), and have asked the browser not to zoom the content (*initial-scale=1.0*). This is important because if the browser felt free to use an area wider than the device width (which would require

horizontal scrolling), the *min-width* code which follows would not work properly.

This brings us to the three style sheets the example HTML file uses. *CA-base.css* contains most of the CSS code that determines the appearance of the page. It sets the number of columns to 1, which will be the arrangement most suited to mobile phones. All we have changed in the file *columns.css* to create this new CSS file is the following:

```
-webkit-column-count: 1;
/* For the WebKit browsers - Chrome and Safari */
-moz-column-count: 1;
/* For Firefox (Mozilla) */
column-count: 1;
```

Figure 26: Part of CA-base.css

We are saying that one column will be the default style unless it is overruled. Having provisionally applied the CSS code in *CA-base.css*, the browser will then quickly read the next line of code, which is:

```
<link href="CA-2cols.css" type="text/css" media="all and (min-
width:600px)" rel="stylesheet"  />
```

The *media* attribute is saying if the viewport has a minimum width of 600 CSS pixels, then the content of *CA-2cols.css* should be applied. This is a very small file, the entire contents of which are as follows:

```
#columnBox {
-webkit-column-count: 2;
/* For the WebKit browsers - Chrome and Safari */
-moz-column-count: 2;
/* For Firefox (Mozilla) */
column-count: 2;
}
```

Figure 27: CA-2cols.css

This third CSS file says that if the viewport has a width of 600 CSS pixels or greater, content should be displayed in two columns. If that condition is met, this contradicts what was said in *CA-base.css*. The browser reacts by implementing the latest instruction about columns but continues to apply other instructions that have not been contradicted. Therefore, one column is changed to two for bigger devices with widths of 600 CSS pixels or greater.

The browser then reads the next line of code, which is:

```
<link href="CA-3cols.css" type="text/css" media="all and (min-
width:800px)" rel="stylesheet"  />
```

This tells the browser that if the viewport has a minimum width of 800 CSS pixels, then a new CSS file, *CA-3cols.css*, should be applied. Its contents are as follows:

```
#columnBox {
-webkit-column-count: 3;
/* For the WebKit browsers - Chrome and Safari */
-moz-column-count: 3;
/* For Firefox (Mozilla) */
column-count: 3;
}
```

Figure 28: CA-3cols.css

Devices with a viewport of between 600 and 800 CSS pixels will continue to display two columns. For higher resolution devices, the number of columns will be changed to three and, again, earlier instructions that have not been contradicted will continue to apply.

This is a simple example of style adaption, but it introduces a useful tool that can be employed to introduce more complex variations in layout depending on the width of the device.

Now, let's look at an example of content adaptation. Remember that a separate file will be sent to a device depending on what that device is. It will not normally be necessary to send separate files to iPhones, Androids and Blackberrys, but it is a good illustration of how device detection works. In each case, the content of the files that are returned consists of just a single line of text: only what is necessary to convince us that the device has been identified correctly.

```
<?php include 'Mobile_Detect.php';
/* The above line of code will import all the code, particularly the
methods (functions), from Mobile_Detect.php  */
$detect = new Mobile_Detect;
/* Creates a new instance of the Mobile_Detect object, which we shall
call $detect. It contains all the User Agent information sent by the
user's browser to the Web server */
if ($detect->isMobile() && !$detect->isTablet())
/* If it is a mobile device and it is not (!) a tablet */
{
   if($detect->isAndroidOS())
   {
   header("Location: android.php");
   /* If this is an Android device, send the file android.php to the
user's browser. Here the file is assumed to be in the same directory
as this program. Because of the previous if statement the device
should not be an Android tablet */
   exit;
   /* Exit the program. This tells the program not to execute any
more code */
   }
   elseif($detect->isiPhone())
   {
   header("Location: iphone.php");
   exit;
   }
```

```
   elseif($detect->isBlackBerry())
   {
   header("Location: blackberry.php");
   exit;
   }
   else
   {
   header("Location: unidentifiedPhone.php");
   /* If the device is not an Android or iPhone or Blackberry. php-
mobile-detect provides other methods (functions) to test for further
devices if required */
   exit;
   }
}
else
/* The code below will execute if this is not a mobile phone */
{
header("Location: notphone.php");
exit;
}
?>
```

Figure 29: detectMobile.php

If this is an iPhone, the program will simply return this:

```
<!DOCTYPE html>
<html>
<head>
<title> iPhone device page </title>
<meta charset="utf-8" />
<meta name = "viewport" content = "width=device-width,initial-
scale=1.0" />
</head>
<body>
<h2>This is the iPhone device page</h2>
</body>
</html>
```

Figure 30: iPhone.php

Device detection is often placed only on a site's home page, but some argue that it should be present on all pages as we cannot predict how users will enter our site. The W3C-sponsored notion of 'One Web' suggests that each resource should have a separate address and that different representations of the same resource should be approached via the same address (W3C, 2008c). One implementation of this concept might involve users all being directed to the same URL but then being presented with a different version of the content depending on their device.

References

AJAX

Garrett, J.J. (2005) *Ajax: a new approach to Web applications* [Online]. Available from http://adaptivepath.com/publications/essays/archives/000385.php (Accessed: 8 September 2012).

Google Code University (2012) *AJAX tutorial* [Online]. Available from http://code.google.com/edu/ajax/tutorials/ajax-tutorial.html (Accessed: 8 September 2012).

W3C (2012a) *Extensible markup language (XML)* [Online]. Available from http://www.w3.org/XML/ (Accessed: 8 September 2012).

W3 Schools (2012) *AJAX introduction* [Online]. Available from http://www.w3schools.com/ajax/ajax_intro.asp (Accessed: 8 September 2012).

jQuery

iTouchMap (2012) *Latitude and longitude of a point* [Online]. Available from http://itouchmap.com/latlong.html (Accessed: 8 September 2012).

McFarland, D. (2012) *JavaScript & jQuery: the missing manual.* Sebastopol, CA: O'Reilly.

Shtrauss, J. (2011) *$.goMap() / about* [Online]. Available from http://pittss.lv/jquery/gomap/index.php (Accessed: 8 September 2012).

Zaefferer, J. (2012) *jQuery plugin: Validation* [Online]. Available from http://bassistance.de/jquery-plugins/jquery-plugin-validation/ (Accessed: 8 September 2012).

HTML5

Modernizr (2012) *Documentation* [Online]. Available from http://modernizr.com/docs/ (Accessed: 8 September 2012).

Oxford Internet Institute (2006) *The future of the Web: a lecture by Sir Tim Berners-Lee* [Online]. Available from http://webcast.oii.ox.ac.uk/index.cfm?view=Webcast&ID=20060314_139 (Accessed: 8 September 2012).

CSS3

W3C (2012b) *CSS backgrounds and borders module level 3* [Online]. Available from http://www.w3.org/TR/css3-background/ (Accessed: 8 September 2012).

W3C (2012c) *CSS multi-column layout module* [Online]. Available from http://www.w3.org/TR/css3-multicol/ (Accessed: 8 September 2012).

W3C (2012d) *CSS grid layout* [Online]. Available from http://www.w3.org/TR/css3-grid-layout/ (Accessed: 8 September 2012).

W3C (2012e) *Cascading Style Sheets articles and tutorials* [Online]. Available from http://www.w3.org/Style/CSS/learning (Accessed: 8 September 2012).

OWASP

NSA (2012) *Web application security overview* [Online]. Available from http://www.nsa.gov/ia/_files/support/I733-034R-2007.pdf (Accessed: 8 September 2012).

OpenSAMM (2012a) *Software assurance maturity model* [Online]. Available from http://www.opensamm.org/ (Accessed: 8 September 2012).

OpenSAMM (2012b) *Download* [Online]. Available from http://www.opensamm.org/download/ (Accessed: 8 September 2012).

OWASP (2012a) *About the Open Web Application Security Project* [Online]. Available from https://www.owasp.org/index.php/About_OWASP (Accessed: 8 September 2012).

OWASP (2012b) *SQL injection prevention cheat sheet* [Online]. Available from https://www.owasp.org/index.php/SQL_Injection_Prevention_Cheat_Sheet (Accessed: 8 September 2012).

OWASP (2012c) *XSS (cross site scripting) cheat sheet* [Online]. Available from https://www.owasp.org/index.php/XSS_(Cross_Site_Scripting)_Prevention_Cheat_Sheet (Accessed: 8 September 2012).

OWASP (2012d) *OWASP enterprise security API* [Online]. Available from https://www.owasp.org/index.php/ESAPI (Accessed: 8 September 2012).

OWASP (2012e) *OWASP Appsec tutorial series* [Online]. Available from https://www.owasp.org/index.php/OWASP_Appsec_Tutorial_Series (Accessed: 8 September 2012).

PHP Group (2012) *Prepared statements and stored procedures* [Online]. Available from http://php.net/manual/en/pdo.prepared-statements.php (Accessed: 8 September 2012).

Mobile Web programming

Apple (2012) *Configuring the viewport* [Online]. Available from http://developer.apple.com/library/safari/#documentation/appleapplications/reference/safariwebcontent/usingtheviewport/usingtheviewport.html (Accessed: 8 September 2012).

Archer, P. (2012) *User agent string* [Online]. Available from http://philarcher.org/cgi-bin/ua_string.cgi (Accessed: 8 September 2012).

DeviceAtlas (2012) *Devices* [Online]. Available from http://deviceatlas.com/resourcecentre/Explore+DeviceAtlas+Data/Devices (Accessed: 8 September 2012).

iUI (2012) *iUI: user interface framework for mobile Web devices* [Online]. Available from http://code.google.com/p/iui/ (Accessed: 8 September 2012).

jQuery (2012) *jQuery is a new kind of JavaScript library* [Online]. Available from http://jquery.com/ (Accessed: 8 September 2012).

jQuery Mobile (2012) *jQuery Mobile: touch optimized Web framework for smartphones and tablets* [Online]. Available from http://jquerymobile.com/ (Accessed: 8 September 2012).

Open Mobile Alliance (2012) *Frequently asked questions* [Online]. Available from http://openmobilealliance.org/AboutOMA/FAQ.aspx (Accessed: 8 September 2012).

Opera (2012) *Opera mini & Opera mobile* [Online]. Available from http://www.opera.com/mobile/specs/ (Accessed: 8 September 2012).

PhoneGap (2012) *Easily create apps with the only free open source framework that supports 7 mobile platforms* [Online]. Available from http://phonegap.com/ (Accessed: 8 September 2012).

php-mobile-detect (2012) *Introduction* [Online]. Available from http://code.google.com/p/php-mobile-detect/wiki/Mobile_Detect (Accessed: 8 September 2012).

ScientiaMobile (2012) *WURFL device detection* [Online]. Available from http://www.scientiamobile.com/ (Accessed: 8 September 2012).

Sencha Touch (2012) *Sencha Touch: build mobile Web apps with HTML5* [Online]. Available from http://www.sencha.com/products/touch (Accessed: 8 September 2012).

StatCounter (2012) *Global stats: top 9 mobile browsers* [Online]. Available from http://gs.statcounter.com/#mobile_browser-ww-monthly-201108-201208 (Accessed: 8 September 2012).

W3C (2008a) *Mobile Web best practices 1.0* [Online]. Available from http://www.w3.org/TR/mobile-bp/ (Accessed: 8 September 2012).

W3C (2008b) *CSS mobile profile 2.0* [Online]. Available from http://www.w3.org/TR/css-mobile/ (Accessed: 8 September 2012).

W3C (2008c) *One Web* [Online]. Available from http://www.w3.org/TR/mobile-bp/#OneWeb (Accessed: 8 September 2012).

WebKit (2012) *The WebKit open source project* [Online]. Available from http://www.webkit.org/ (Accessed: 8 September 2012).