HTML and HTML 5

Introduction

HTML is a document format that defines the ***content*** of web pages. The more or less puts it on an equal footing with the capabilities of a Word Processor in that it allows a user to specify the roles of different parts of a document (headings, paragraphs, table of contents etc.) but says nothing about how the user interacts with it. People who got used to word processors in the 1980s and 1990s (such as WordStar and WordPerfect) would recognise raw HTML documents for what they were instantly.

Like these older word processing programs, HTML got updated and some of the software got cleverer at hiding the basic nature of a document, so there are many people who work with HTML who never get to see the underlying document format. Programs like Adobe Dreamweaver, Microsoft Expression and NVU allow the users to work in a what-you-see-is-what-you-get (WYSIWYG) mode that hides all of the HTML mark-up from them. However, developers who want to go beyond the basic capabilities of creating static (i.e. fixed-format, fixed-content) web pages need to get their hands dirty at some point and descend into HTML mark-up.

Much of HTML 5 is based on the premise that a web page should be a live, interactive document that is not static – the user interacting with an HTML 5 document can change it. HTML 5 is not a single technology, but a combination of a number of features that embellish HTML to make it more capable of interoperability, working with multimedia and managing the user's current context (e.g. where they are, what they are currently doing, what their preferences are) in a seamless and platform independent way. HTML 5 is a specification for features that should be available (eventually) to every browser, regardless of who makes it or what type of device it is installed in. To do this, HTML 5 draws heavily on two other core technologies – Cascading Style Sheets version 3 (CSS3), and Javascript. Some HTML 5 features are independent of these (e.g. forms, semantic mark-up, media playback) but many rely on them completely (geo-location, drag-and-drop, micro-data, local data storage). Amazingly, almost all browser manufacturers have decided to put aside their fundamental differences to work towards the common HTML 5 standard, so an application you build for one browser in HTML 5 has a much better chance of working just as well in the full range of browsers than has ever been possible before.

HTML 5 and Mobile

While HTML 5 standards have made it much easier to create cross-browser applications, browsers still contain differences that are enough to cause some problems with some core features. This is due to a number of reasons – browsers tend to be based on core 'engines' that work in fundamentally different ways, different manufacturers interpret standards differently, some manufacturers get a bit pig-headed about specific standards – for example, Microsoft have yet to implement support for some common web standard formats for video (MPEG-4, Ogg Theora, WebM) which other browser manufacturers fully support, few browsers have yet to fully support a number of web-forms features etc. However, HTML 5 is a moving standard that is not due to be finalised until 2014, so this is not a surprise.

However, if you are working on building a mobile app based on HTML 5, these differences between browser can cause significant problems. Various mobile frameworks have become popular because of this. The frameworks (e.g. jQuery Mobile, jqTouch, Sencha Touch, Mobi) are designed to eradicate the differences between browsers so that a single app works the same across a wide range of devices. They are not completely successful in this, but the situation improves almost daily.

Mobile web frameworks have three jobs to do:

1. Remove the differences between browsers so that a single app works properly on a range of browsers in different devices
2. Style a web-app so that it appears like a native application on the mobile platform. This usually means removing all of the standard browser 'furniture' (e.g. the URL bar, the browser's status bar)
3. Adapt the look and feel of a web app so that it can be interacted with in the same way that a native mobile app (e.g. Android, iOS or Windows Mobile) would. Typically, this means providing larger, "finger-friendly" buttons and control areas in the user-interface of the app and handling touch-based gestures such as taps and swipes

On addition, the mobile frameworks make it easier to define interactions between the client (i.e. the program running on a device's browser) and the server (i.e. the online location that the app is accessed from).

An example HTML 5 App

To see how HTML 5 and mobile frameworks can be used to create an application, it is worth looking at a simple example. We'll take the example of a simple currency convertor app (i.e. a program that allows you to convert a price in pounds into dollars or Euros) and go through the stages from a simple web-app to a mobile app. I hope you can already see that this is an ideal application to make into a mobile version, since it would provide useful support for someone who wanted to get an idea of the cost of potential purchases while travelling abroad.

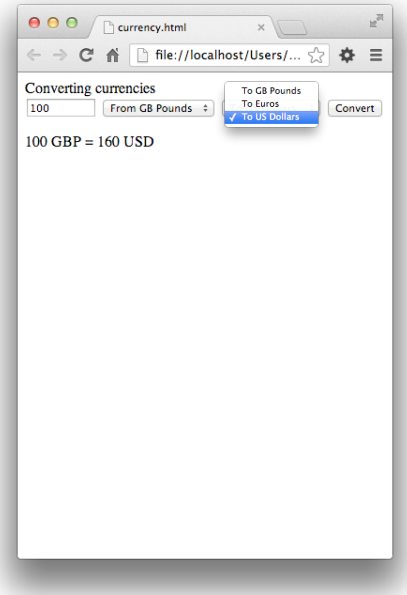

Figure 2.1: A simple HTML 5 Application

Figure 2.1 shows the currency app developed for standard HTML 5. This application contains two core components:

1. A HTML file that contains the basic mark-up of the application's user-interface
2. A simple Javascript file that contains all of the program code required to do the currency conversion

This is typical of a HTML 5 application: the look of the app is defined in HTML while its behaviour is defined in Javascript code.

HTML is used to define the static on-page text (i.e. Converting currencies), the pop-up lists of currencies supported (i.e. From GB Pounds/Euros/US Dollars, To GB Pounds/Euros/US Dollars) and the "Convert" button. Javascript is tasked with providing the raw data that currency conversions are based on (i.e. the Pounds/Euros/Dollars conversion rates) and performing the actual calculations that turn an amount in Pounds/Euros/Dollars into an amount in some other currency.

The HTML in the app is fairly simple:

```
<html>
   <head>
         <script type="text/javascript" src="currency.js"></script>
   </head>
<body>
   <header>Converting currencies</header>
   <input type="text" id="amount" size="10"
          placeholder="Amount"></input>
   <select id="from">
         <option value="GBP">From GB Pounds</option>
         <option value="EUR">From Euros</option>
         <option value="USD">From US Dollars</option>
   </select>
   <select id="to">
         <option value="GBP">To GB Pounds</option>
         <option value="EUR">To Euros</option>
         <option value="USD">To US Dollars</option>
   </select>
   <button id="convert">Convert</button>
   <br/>
   <p id="result">Result goes here</p>
</body>
</html>
```

Listing 2.1: The HTML code in the currency app

The Javascript in the app is also fairly simple (maybe not right now, but it will be fully explained in the next chapter):

```
// This is the currency conversion data...
var rates={"GBP":1.0,"EUR":1.2, "USD":1.6};

// This statement sets up the event handler for the button...
window.onload = function() {
    document.getElementById("convert").onclick = doConversion;
};

// This collects values from the form, calls a function to do the conversion
// and then inserts a suitable message into the result paragraph...
function doConversion() {
    var amount = document.getElementById("amount").value,
        from = document.getElementById("from").value,
        to = document.getElementById("to").value;
    document.getElementById("result").innerText = amount + " " + from +
                  " = " + getExchangeValue(amount, from, to) + " " + to;
}

// This does the actual conversion arithmetic...
function getExchangeValue(amount, from, to){
    var fromRate = rates[from];
    var toRate = rates[to];
  return  amount * toRate/fromRate;
}
```

Listing 2.2: The Javascript code in the currency app

While the different parts of this application may appear to be complex and impenetrable, there is nothing very complex going on and in particular, the HTML part of it is quite easy to follow.

HTML Mark-up

Every properly formatted HTML document is made up of three types of thing:

1. HTML tag pairs, such as <html>…</html> and … , which indicate what the purpose of the enclosed text is – e.g. an entire document appears between <html> and </html>,

while a single paragraph appears between and . Tags are also used to indicate that a section of text should be displayed differently (e.g. heading tags like <h1>..</h1>), or to introduce active elements, such as boxes the user can type into (textboxes), lists the user can select from (select option lists), buttons, checkmarks etc. Tag names can be in upper or lower case, although current convention is to use lower case

2. Content, which is placed between pairs of tags – for example the <header> tag of the above HTML document contains the text "Converting currencies". Content can take several forms, including plain text, text formatted with further HTML, images, videos and audio. Formatting within the content between tags (e.g. line breaks, tabs, extra spaces) is ignored in the rendered content
3. Attributes, which are names and associated values that appear within the opening tag of a tag pair and define some feature of the tag – for example the <button> tag has the attribute **id**, which has the value "**convert**". In this case, the id attribute makes it possible to refer to the button in Javascript code

In HTML terminology, the combination of a pair of tags and whatever they enclose is called an Element, so **<header>Converting currencies</header>** is a complete HTML element, with tags and content

There are a number of simple rules that need to be followed for an HTML document to be considered to be valid and "well formed". These are:

- Every opening tag must have a corresponding closing tag – e.g. <header> needs a </header> tag. Browsers actually allow this rule to be relaxed in some cases (there is no real need to close a tag for example) but it is good practice for developers to ignore what browsers allow and be more rigorous when writing HTML
- Elements can appear inside other elements, so, for example, the tag pair <html> and </html> encloses an entire HTML document. Similarly, the code:

```
<select id="from">
        <option value="GBP">From GB Pounds</option>
        <option value="EUR">From Euros</option>
        <option value="USD">From US Dollars</option>
</select>
```

Listing 2.3: HTML Nested elements

is an example of one element (a <select> element) containing three other elements (<option>), which are considered to be 'child' elements of the <select>. In some cases, elements can be nested in this way to arbitrary depth, but most often there is a specific reason why one element contains others, s in the example of the <select> element, where the options the user can select from need to be defined

- Nested elements must be properly enclosed, so, for example, in listing 2.3 it would have been wrong to close the <select> element (</select>) before closing one of the <option> elements that are inside it
- Attributes must always be defined within the opening tag, and their values should always be placed in quote marks, either "double quotes" or 'single quotes'. Again, browsers tend to relax this rule, so you will see a lot of HTML where attributes are not in quotes – this is bad practice and the developers responsible will go to hell for it

Following these rules, we can build an HTML document up to almost any level of complexity. Since all HTML documents are nested within a "document element" which is the <html> tag, it is fairly easy to follow the structure of the document.

Start with the outer <html> tag pair:

```
<html>
</html>
```

Now add <head> and <body> sections:

```
<html>
      <head>
      </head>
      <body>
      </body>
</html>
```

The <head> section contains data ***about*** the document – e.g. its title, resources such as scripts or style-sheets that the document uses; definitions about the way the page should fill the browser etc. Typically, we give a page a <title> element, since this is what will be shown in the title bar of the browser showing the page:

```
<html>
      <head>
            <title>My New Web-Page</title>
      </head>
      <body>
      </body>
</html>
```

The <body> section contains the actual document content and its associated mark-up. We can now go on to add the actual information we want to appear on the page:

```
<html>
      <head>
            <title>My New Web-Page</title>
      </head>
      <body>
            <h1>This is my first web page</h1>
            <p> At some stage I intend to make this page sing and
               dance.</p>
      </body>
</html>
```

We can continue in this manner to build up the document until the page contains all of the information we need it to. One thing to note is that the format of the displayed HTML is not defined by the way you lay out the HTML code. For example, pressing the enter key on a line of text (as has been done with the <p> element above) will not affect how the text is displayed in the browser; the only action that will start a new paragraph would be to add a new paragraph element with tags and content. Since HTML is often generated by software, it is quite common to see whole, very large documents that all appear in a single line of HTML text, even though they are formatted very neatly in the browser.

HTML and Hyper-Links

The 'H' in HTML stands for Hypertext, which is an indication that the whole point of HTML is for documents to contain links to other documents – so called *hyper-links*. The world-wide-web can be described as one big hypertext document, since all documents on it are interlinked to some degree.

In HTML, a link between two documents is created using an "Anchor" element, which uses the tag-pair <a>…. A typical hyperlink, which would take you to the UWS website's home page is:

```
<a href="www.uws.ac.uk">UWS Website</a>
```

As you can see, the href attribute contains the URL (Uniform Resource Locator) of the target site, and the tag pair contains the text that this should show up as on the browser.

Associated with the idea of a hyper-link are *hash-links* and *deep links*. Within a HTML document, any element can be given an ID attribute and this attribute can have a number of uses – for example, we can access an element to attach a Javascript function to it using its ID. However, the main purpose for an ID attribute is so that an internal link to it can be made. That makes it possible to refer to a specific element in the browser, at which the usual behaviour is to scroll the browser window to make the element visible. This is useful when a page is so big that it does not all fit into the browser window. For example:

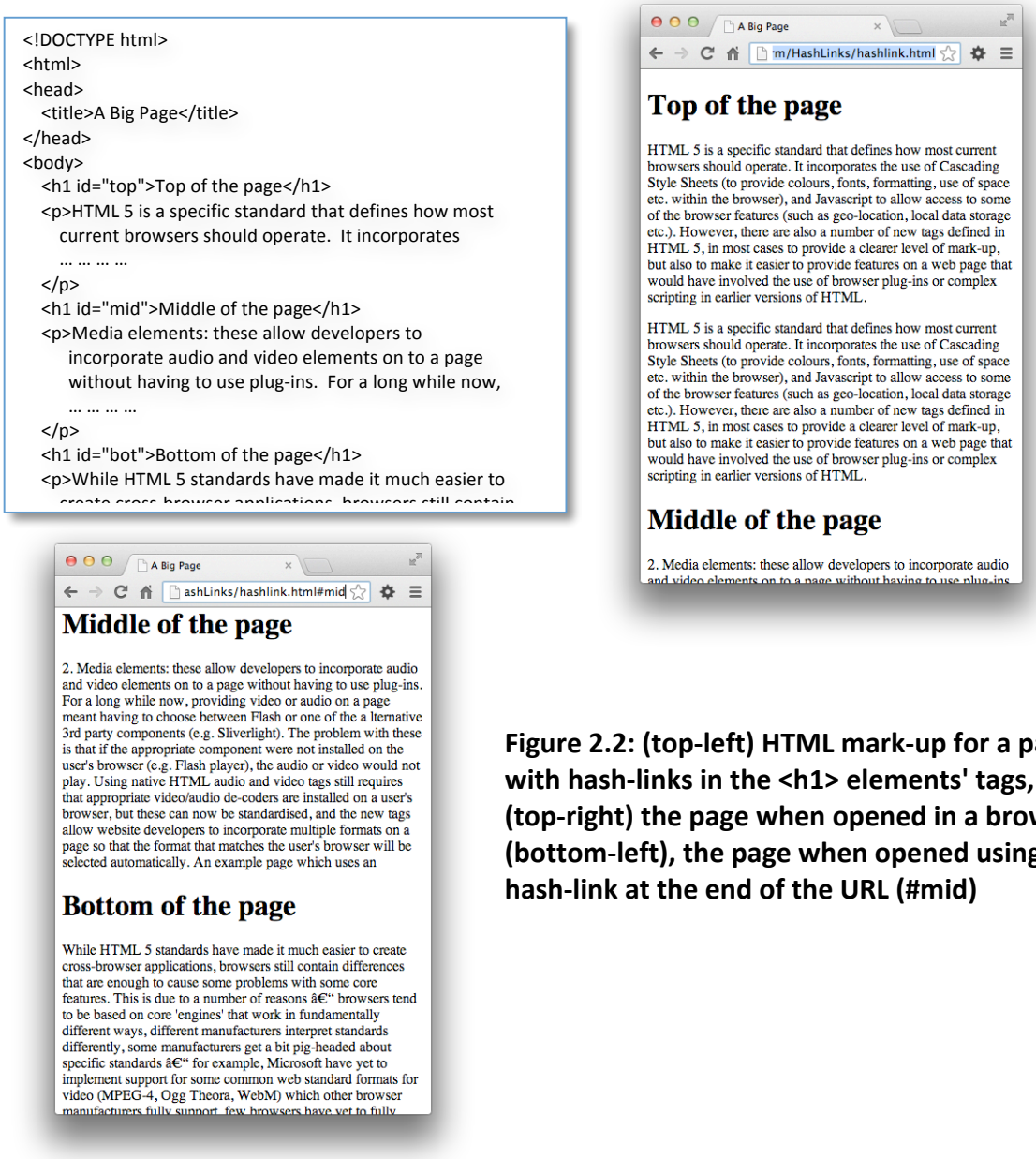

Figure 2.2: (top-left) HTML mark-up for a page with hash-links in the <h1> elements' tags, (top-right) the page when opened in a browser, (bottom-left), the page when opened using a hash-link at the end of the URL (#mid)

A hash-link within a page can be used to access a specific element in the page without any of the rest of the url – e.g. `<a href="#mid">Middle of the page</a>`. The hash-tag can also be used at the end of a complete URL to link from some other page to a specific point in that page, for example – `<a href="hashlink.html#mid">Middle of the document</a>`. In that

case, the full URL (including the hash-tag) would be called a deep-link.

All of the current mobile web frameworks make use of hash-links and deep-links to access different pars of a page that then appear as links to different screens in the same application.

The Document Object Model

HTML mark-up is defined with a view to making it as easy as possible for the author of a web page to use; essentially little different from a word-processor format from a few years ago (in WordPerfect, document mark-up for bold, paragraph styles, tables etc. was visible as you created a document). However, this format is not very efficient for a browser to work with. Browsers need to apply rigid rules defining how different elements are to be displayed and (in 'active' web-pages) must be able to manipulate the content of a document without losing track of any of the mark-up. The Document Object Model (or DOM as all developers refer to it) is the internal representation of a webpage that the browser uses to make it easy to render the page and manipulate its contents.

The DOM is created as a set of hierarchical elements, so, for example, a simple webpage would be organised as shown in figure 2.3:

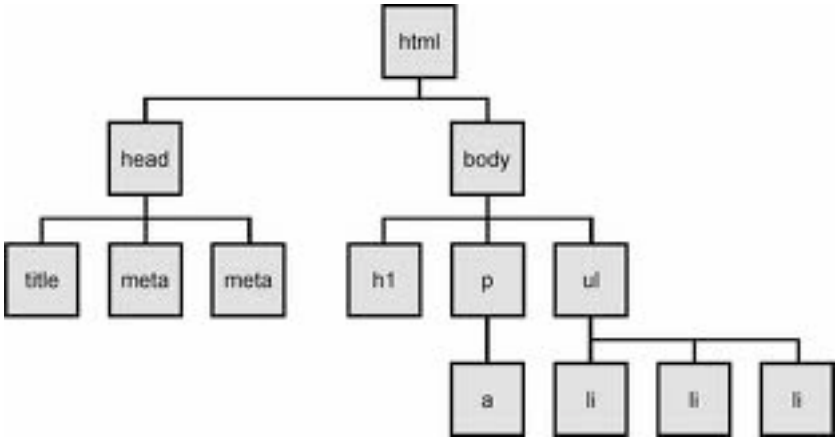

Figure 2.3: The DOM for a simple web page

What you can see in figure 2.3 is that the top element (<html>) has two *child* elements (<head> and <body>). In turn, the <head> element has three children and the <body> element has another three, two of which have children of their own. This structure works very well for several reasons:

- Rules that apply to an element (e.g. the <ul> element inside the <body> element above) also apply to their children
- It is easy for the browser to be able to manipulate elements – detaching child elements or adding new ones as necessary)
- It is easy to manipulate in program code – a programmer can add a child element to an existing one using simple Javascript functions

Although DOM manipulation functions are not part of core Javascript, they are part of the browser's definition of the DOM, so programmers can easily access parts of a live web document and work with it. Typical operations could be:

- Adding new elements to a `<ul>` list element
- Changing the displayed style of a specific element or family of elements using CSS rules (later)
- Generating complex structured elements in program code – for example, inserting a depiction of a family tree using structured lists

- Accessing the contents of user-interface elements (e.g. textboxes, check-boxes etc.) to process user-data
- Responding to interactions within a page (e.g. user taps/clicks on a <p> element on the page, user drags a `<li>` element from one part of a page to another) – event handling

Frameworks like jQuery Mobile work by manipulating the DOM to make documents appear and interact in ways more appropriate to small-screen devices. The event-handling part of the DOM in particular is used to make pages more interactive than a simple, static web page, and provide the mechanisms needed for creating interactive applications within a browser.

HTML 5

HTML 5 is a specific standard that defines how most current browsers should operate. It incorporates the use of Cascading Style Sheets (to provide colours, fonts, formatting, use of space etc. within the browser), and Javascript to allow access to some of the browser features (such as geo-location, local data storage etc.). However, there are also a number of new tags defined in HTML 5, in most cases to provide a clearer level of mark-up, but also to make it easier to provide features on a web page that would have involved the use of browser plug-ins or complex scripting in earlier versions of HTML.

The new HTML 5 elements fall into four broad categories:

1. Semantic/Structural Elements: these are elements that are used to make the purpose of the mark-up clearer. In many cases, this makes it easier for programs and web-crawlers (e.g. Google's indexing components) to figure out more about the page. However, many of these elements ought to also make the job of marking up pages easier for developers. New tags such as <article>, <aside>, <section>,<summary>, <figure>, <footer> and <header> will improve a readers understanding of the meaning of much of the mark-up. Using semantic elements, a developer can define a template web page using the new tags much more easily and clearly – for example:

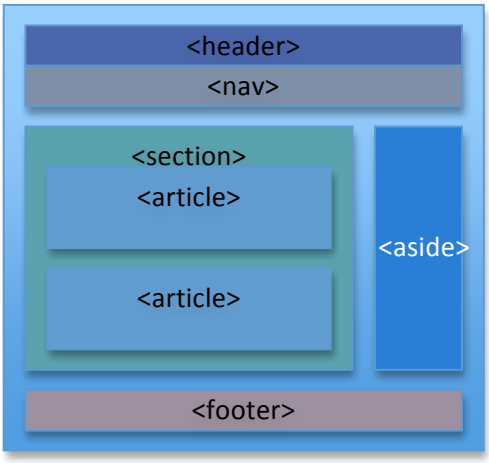

Figure 2.3: The HTML 5 Semantic elements: how they could be used in page design

Note that the semantic elements have no effect on the document layout (that would normally be established using CSS rules to create margins, side-by-side elements etc.)

2. Media elements: these allow developers to incorporate audio and video elements on to a page *without having to use plug-ins*. For a long while now, providing video or audio on a page meant having to choose between Flash or one of the alternative 3rd party components (e.g. Silverlight). The problem with these is that if the appropriate component were not installed on the user's browser (e.g. Flash player), the audio or video would not play. Using native HTML audio and video tags still requires that appropriate video/audio de-coders to be supported by the browser and different

browsers support different standards. However, built-in browser support is an improvement over plug-in based support, and the new tags allow website developers to incorporate multiple formats on a page so that the format that matches what the user's browser supports will be selected automatically. An example page which uses an <audio> tag with multi-format content is shown in figure 2.4 along with the result shown in a browser window – note the controls for audio playback are defined by a simple attribute of the <audio> tag:

```
<body>
   <audio controls autoplay>
       <source
src="assets/tune.mp3"
                type="audio/mp3" />
       <source
src="assets/tune.ogg"
              type="audio/ogg" />
       <source
src="assets/tune.wav"
```

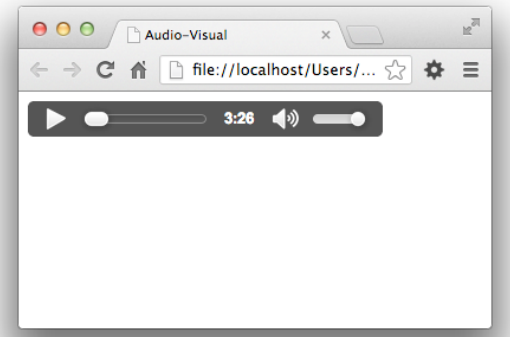

Figure 2.4: The HTML 5 <audio> element

3. Forms: web pages have used forms since HTML version2. However, the support until recently has been rudimentary, and plug-ins and Javascript code is often used to provide browser forms with more up to date features, such as date-pickers, progress bars and text fields for entering numbers, email addresses and other items where the format was critical. HTML 5 now defines a range of these; currently not all browsers support the smarter form elements (e.g. on-screen calendars for selecting dates) but gradually the browser manufacturers are falling into line. Figure 2.5 shows the HTML code for a web-form to collect a user's name, email address and date of birth, and the form as rendered in a browser window – in particular, the "date" field is active:

```
<body>
    <h1>Web Forms</h1>
       <form id="form" action="processForm">
         <label for="name">Name:</label>
              <input type="text" id="name"/><br/>
         <label for="email">email:</label>
              <input type="email" id ="email"/><br/>
         <label for="dob">Date of Birth:</label>
              <input type="date" id="dob"/><br/>
              <button id="ok">OK</button>
              <button id="cancel">Cancel</button>
              <button id="load">Load Data</button>
       </form>
</body>
```

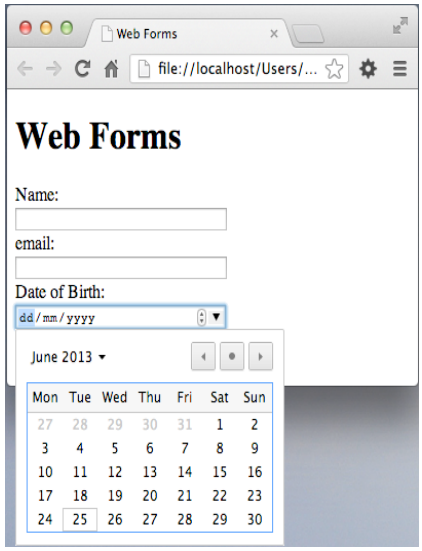

Figure 2.5: A web-form with HTML 5 specific controls

4. The best new HTML 5 element (in my humble opinion) is the <canvas> element. This allows Javascript code to draw directly into an area on the browser page. Although graphics have been a big feature of web pages from the earliest days, the have always had to be pre-prepared and delivered from the server. Previously, plug-ins like Flash have been needed to make it possible to incorporate 'live' graphics (e.g. graphics drawn in real-

time on the browser or animations), but now HTML includes the <canvas> component to make this a native feature.

Figure 2.6 shows a simple drawing application that uses the HTML 5 <canvas> tag.

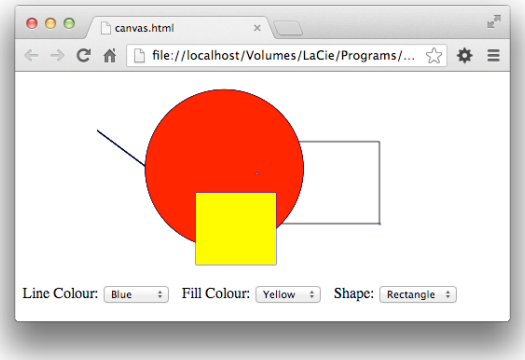

Figure 2.6: The HTML 5 <canvas> element

HTML 5 Programmed features

Several of the features specific to HTML 5 can only be accessed using Javascript code. Since these features a specifically designed to make HTML into a ***web-applications platform***, capable of performing programmatic operations, that is what you would expect. The <canvas> element described above would produce no more than a blank area in the browser window but for the Javascript code that drew into it. Similarly, some key HTML 5 features rely completely of Javascript code to do their job. In all cases, the features are built around an API (Application Programmers' Interface):

- Geo-location: this is an API that allows Javascript code on the page to find the current location of the machine the browser is running on. While this is fairly useless for a desktop machine (I'd assume you know where you live), it is a godsend for anyone who has ever had to find directions using Google maps, or locate the nearest hospital, bus station or coffee shop
- localStorage: this API allows a web page or app to store data on the local machine. Normally, a browser is not allowed to do this, since it would give websites access to your computer. However, locally stored data is necessary for many times of application, and so HTML 5 compliant browser can make a request to the user to store a limited amount of data. Again, Javascript code is used to store data on the local machine and retrieve it as necessary
- Drag-and-drop: HTML 5 applications can make use of a drag and drop feature to make it easier and quicker to use functions by providing the same method of moving items about as you expect from a desktop operating system. Javascript code is required to initiate a drag operation and to respond to the items (e.g. files) as they are dropped
- Web Workers: web applications can initiate processes that run in parallel with the user-interface. This can improve the responsiveness of the user-interface, since there is no lag while a long-running task completes

The feature that all of these items have in common is that HTML mark-up alone is not enough to implement them. Javascript APIs are pre-defined functions and objects that provide access to the features. For example, to save data in a text box to local storage, you would need to access the localStorage object, making use of its getItem() and setItem() functions:

```
document.getItemById("saveButton").onclick = function() {
      var data = document.getElementById("textbox").value;
      if(localStorage) {
            localStorage.setItem("progData", data);
      }
}
```

Listing 2.4: Javascript code to save a textbox at the click of a button

Content, format and behaviour

A typical HTML 5 application is made up of three types of code:

1. HTML, which is used to define the content in the application, including text, images, audio and video, along with the structural relationships between these items
2. CSS, which defines the format that will be applied to the HTML elements, including fonts, colours, spatial relationships (margins, line spacing etc.) and animations and transitions that can be applied to them
3. Javascript, which is used to manipulate the content and the formatting to perform work on behalf of the user

A fourth component of any HTML 5 application is the actual browser that it runs within. Without a browser, an HTML 5 app has no venue. The browser provides an execution environment, renders the application's user interface and hosts the APIs. There are a large number of "more-or-less" HTML 5 compliant browsers available, and it says a lot about the HTML 5 standard than most HTML 5 mark-up and code that is written executes in exactly the same way in each of them.

There are differences between browsers, and sometimes these cause apps that work fully in one browser to fail in another. However, as browser development progresses, these differences are becoming less and less significant. The final draft of the HTML 5 standard is due in 2014 (advanced by quite a bit from the original 2022 due date), and it is widely expected that by that time, all current browsers will have been updated to implement the existing standard fully.

Cascading Style Sheets

Without CSS, web pages would have a very dull appearance. CSS is defined as a set of rules for how a browser should render specific types of element, or elements that have been given a specific 'class'. CSS rules are built of simple text-based statements placed inside a `<style>..</style>` tag pair, or in a CSS file linked to a document using a `<link>` tag in the document's <head> element. Some typical CSS rules are:

```
h1{
   font-size: 16;
   color:blue;
}


p.centered {
   text-align:center;
}
```

The two rules shown above define how ALL <h1> elements and *certain* <p> elements will appear on the page. Any <h1> element in the document will be rendered in a font size of 16 points in blue. Any <p> element that has been given the 'class' centred will be centred on the page – e.g.

```
<p class="centred">This text will be centred on the page</p>
```

CSS rules can be used to change almost any aspect of how elements are displayed on the page – the font, colour and size are obvious features, but also how much white space appears around an element (margins, padding), whether elements are positioned absolutely on the page or relative to their container element, whether elements have borders, a background colour or even a background image and even how elements will animate when CSS rues are applied to them (instead of a paragraph simply centring on the page, it could be caused to drift slowly across the page until it is centred.

While HTML forms the bones of a web page – what it contains and how it is organised, CSS rules dictate how it will appear to the user, and can provide some very surprising effects. Have a look at the CSS Zen Garden website (http://www.csszengarden.com/), which displays the same HTML mark-up in an ever changing range of styles based on CSS style-sheets that visitors are encouraged to submit to the page. I think you'll find what a few rules can do to be very surprising.

HTML 5 and Mobile Frameworks

HTML 5, the standard, became a mainstream topic in the developers' community in in 2010 after Apple Inc's CEO, Steve Jobs, published a public letter called "Thoughts on Flash". Jobs argued that plug-ins such as flash were not necessary for common web tasks such as viewing videos or playing audio files, since it was now technically possible to incorporate these features into HTML 5. Up till then, HTML was still considered by most to be a 'viewing platform'. Apple went further by blocking the installation of flash on to their new iPhone while making sure that the iPhone's browser was capable of doing all the things that till then Flash had been considered necessary for.

For more than a year after the release of the first iPhone models, the iOS development kit (necessary for building native iPhone Apps) was not made available to the public, but Apple did already provide an Integrated Development Environment (IDE) for developing web apps and dashboard (i.e. Apple desktop) mini-applications called Dashcode (see figure 2.7).

Figure 2.7: Dashcode – Apple's Web-App development kit. Note the project types available (Safari is selected on the left, so the right-hand pane shows web type projects)

Using this, Apple promoted the development of iPhone apps based on HTML 5, Javascript and CSS. In doing so they effectively invented the idea of building apps *designed to run inside a mobile browser* (note the checkbox in the centre of figure 7 for indicating that you intend to build

an application for a mobile version of safari – Apple's web browser).

Using Dashcode, a developer can define a user-interface for an app (i.e. the controls and widgets that would appear on the screen) and attach Javascript functions to handle events generated by the user-interface elements. Dashcode can then generate a folder containing the elements of the Web App that can run within the browser in an iPhone or iPad. Although the development kit for this is specific to Apple's devices, the code created is nothing more than HTML 5, CSS and Javascript. A Dashcode iPhone app is based on an iPhone ***framework***; a set of code files – mainly CSS and Javascript – that define a common core of operations for an iPhone web application.

CSS provides a look and feel to a web-app; the style of buttons, lists, screen backgrounds etc., and simple definitions of on-screen transitions and animations can be defined to enhance how an app appears and interacts with the user. Javascript gives developers the power to manipulate data, define operations and initiate user-interactions. The applications developed using Dashcode can run within a standard browser or the browser on another type of mobile device (e.g. an Android or Windows Mobile phone) perfectly well, although the running app will always look like an iPhone app.

By creating Dashcode and providing the frameworks for iPhone development, Apple kick-started a whole new Web Development sector for creating mobile web apps. Of course, the Dashcode application only runs on Apple machines, and it does create apps that look very iPhone specific, but it provided an idea that was then taken up by a range of developers, mostly working on open source software.

Web developers are an inventive lot, and by using the same principles as Apple promoted to create iPhone apps (HTML 5, CSS & Javascript), they were able to create similar frameworks for different types of device. Mostly, the frameworks provide a look & feel and all of the real work of a web-app is done in Javascript, using HTML 5 features where necessary. The general breakdown is that Javascript provides for basic data manipulation, interaction with the (CSS styled) user-interface and creation of operations in the form of functions. HTML 5 provides access to features of the web browser that support more machine-specific types of operation (access to location data, data storage etc.).

Currently the most popular frameworks are jQuery Mobile and Sencha Touch. Both of these come in the form of files that can be added to an HTML 5 web-app. Once added, pre-styled HTML tags can be added to the app to define pages, buttons, lists, checkboxes and menus etc., all of which have an appearance that fits in with the mobile platforms that host the app. For example, figure 2.8 shows the same app before and after the inclusion of the jQuery Mobile framework.

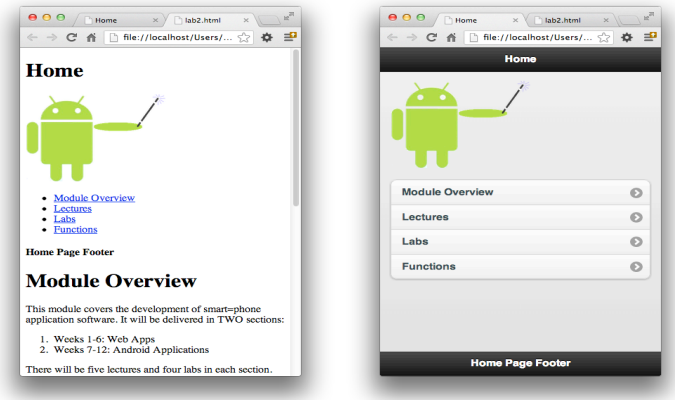

Figure 2.8: A simple HTML page (left) and the same page styled using the jQuery Mobile framework (right)

Rather than the cosmetic appearance of the app, the most important feature shown in figure 2.8 is that in the mobile version, hyperlinks have been re-styled so that they now appear in a size

suitable for selecting with a fingertip. The original version, with its normal hyperlinks, could only be confidently operated with a mouse pointer. The re-jigged version also renders the app as a number of separate "pages", which tends to fit in with small-screen operation more successfully.

All mobile frameworks make this change, rendering an app as "finger-friendly". In addition, mobile frameworks provide ranges of other on-screen controls that are more appropriate for a 'touch-based' interface, handle gestures (such as left-to-right and top-to-bottom swipes), provide slick transitions between "pages" of the app and incorporate control modes that work better on devices with a limited screen area. Most also provide further leverage to make it easier to use standard web-app and HTML 5 features such as remote data access, geo-location and offline storage.

HTML 5 is not, however, a fixed standard. It has been defined in such a way that new features can be added, existing ones extended and in some cases obsolete features can be removed, all without breaking existing applications. Recent additions to the HTML 5 specification include access to on-device imaging (i.e. the camera on your phone), access to the device's local file system, editable content and access to device sensors (such as accelerometers, microphone etc.). The general aim for HTML 5 is to support apps for a wider range of device types, including phones, tablets and phablets (large phones, small tablets). If you continue to program in HTML 5, you will need to keep learning as new features become popular with users. This is true of all good programming environments – they move with the times and incorporate innovations as they become mainstream. You just need to get used to the idea.

Exercises

1. An HTML <h1> element is to have an id attribute with the value "headingText" and the content "Home Page". Write this HTML element.
2. As described in this chapter, an HTML page should contain an <html>..</html> tag pair which enclose <head> and <body> tag pairs. Starting with a blank text file (give it the name ***test.html***):
    a. add some plain text (e.g. "Test") and load the file into a browser (double-click on it)
    b. add a content element (e.g. Test), and re-load the page into the browser
    c. add an enclosing <body> section and again re-load the page
    d. add an enclosing <html> element and re-load
    e. finally produce a full (short) HTML document, adding a <head> section and giving the document a <title> element.

    Examine the page in the browser at each stage. Your aim is to find out how the browser responds to incomplete HTML
3. Create a very simple web page in a text editor such as Notepad – call it ex3.html, and give it <html>, <head> and <body> sections in the correct configurations. Add a element and give it an id attribute with the value "para" and content "Hello World". Now open this page in the Chrome web browser, and then press Ctrl+Shift+J (Windows) or Cmd+Option+J (Mac) to open the Javascript console.

    Enter the following at the prompt:

    - document.getElementById("para")

    The element should be shown in the console, with a little triangle at the left: you can click on this to "open-up" the element to inspect its contents. Now press the Up-Arrow cursor key – the document.getElementById("para") should re-appear in the console

window. Append .innerText = “Hello fred” to this (make sure there is no space between the end of the closing bracket and the period) and check what has happened to the text in the browser window

Questions

1. A (very concise) HTML document is:

   ```
   <html><body><h1 id=”h1”>A top level heading</h1></body></html>
   ```

   What is the content of the HTML element whose id is “h1”?

2. What is wrong with the following HTML fragment?:

   ```
   <select id=”cars”>
       <option>Audi</option>
       <option>BMW</option>
       <option>Chrysler</option>
       <option>Daiwoo</option>
       <option>Eagle</option>
       <option>Ford</option>
   <select>
   ```

3. HTML 5 semantic elements are used to define the layout of parts of a document, such as headers and footers etc. – true or false?

4. HTML hash-links (#) are used (select one):

   a. To mark elements that contain numbers

   b. To define different styles of underlining on the page

   c. To define internal hyper-links and deep-links in a document

   d. To separate items in lists

5. HTML 5 audio and video is defined so that (select all that are correct):

   a. There is no need for decoder software to play an audio of video tag

   b. Audio and Video are now supported in a universal format

   c. The browser has audio and video decoders installed

   d. The browser is able to detect multiple formats being available and select the most appropriate one to download

   e. The browser is able to provide controls for playback using simple attributes of the <audio> element