HTML5 OVERVIEW

HTML5 is a new standard (not yet fully ratified), and includes a number of simplifications, enhancements and improvements over all previous versions of HTML. It includes a number of new page elements plus improved versions of several existing ones. Audio, Video and Graphics (2D and 3D) are now part of HTML, along with improvements to form handling, browser-based data storage, geo-location and arrangements for creating web-apps that can go offline (i.e. that will work without a web connection).

Core HTML

HTML is not a programming language; this is clear from the lack of elements that are required in programming languages – variables (HTML variables exist but work differently), functions, structures for selecting and iterating code statements etc. It is correctly described as a *mark-up* language, which simply means that the content of an HTML document is embellished by additional content that acts to define a structure that can affect the document's appearance or mark specific bits of content for special purposes.

Best practice in HTML creation is to consider the three aspects of a document or web page separately, with each having a language dedicated to that purpose:

- HTML defines the ***content*** of a document – how different parts relate to other parts, how the document is structured
- CSS (Cascading Style Sheets) are used to define the appearance of a document. Each HTML element has a style associated with it – for example, each <p> element has, by default, a certain typeface and size, margins, borders (usually none), colour, line spacing etc. The advantage of this is that by replacing the default CSS definitions in a document, the entire appearance of the document can be altered without touching the HTML itself
- Javascript is used (of course) to define behaviours within an HTML document. The document may be a web-page which allows the user to send data to an online store, and in this case Javascript is used to upload the data, validate user-entries (e.g. to check that a date is valid) etc.

The three components of HTML documents and applications (HTML, CSS and Javascript) are probably by now the most widely used computer programming environment ever.

HTML, from version 1, was designed as a way of creating interacting inter-linked documents. The HT in HTML stands for Hyper-Text, a form of text in which links within one document can lead to other documents. So, even although most of what goes into a HTML document is used to apply type-specifications to specific bits of text, the most important feature stems from the idea of a link:

```
<a href="http://ui-patterns.com/">User-Interface Patterns in HTML</a>
```

The link shown above, when placed in a document, will make the text "User-Interface Patterns in HTML" into a clickable link. It is the ease with which this 'clickability' can be added to a document that gives HTML its power; it is hardly any more difficult to make clickable text do other things, such as run a script.

Much of the rest of the core of HTML is not much more than what a word-processor does – allowing you to mark specific text to have a particular format or purpose. The most commonly used tags for document mark-up are:

Tag type	Description	Example
<html>..</html>	marks all enclosed material as a HTML document	Any whole HTML document
<head>..<head>	separates out the head of a document (which contains meta-data, such as document title, script elements, links to style-sheets etc.)	<head> <title>HTML5 Development</title> </head>
<body>..</body>	encloses all of the actual document content	<body> This is a very short document </body>
..	mark text as a paragraph	Some text
..	mark text as a top-level heading	Writing HTML
..	mark text as a second level heading (etc. for levels 3, 4…)	Document Structure
<a>..	mark text as an ***anchor*** – i.e. a clickable link	Cascading Style Sheets
	Add an image to the document	
…	Mark a block section of a document, grouping any number of enclosed tags. This allows all of the contents to be given a style in one step, or the section to be identified for manipulation by JS code	All tags in here are considered children of the div element. Can be selected by document.getElementById(“results”)
…	Mark a linear section of a document (i.e. text within a block) – typically for applying a text formatting style	<span style=”color: red”>Red Text

Table 6.1: Common HTML Elements

There are of course many more HTML tag types – see http://www.w3schools.com/tags/ for a complete list of all of the tags in the HTML 4.01 standard, complete with examples and exercises. There are quite a few tag types that you may never use.

The HTML 5 “Standard”

HTML5 is still under development – the final proposed specification is expected in 2022: yes, 10 years from now (at time of writing these notes). However, this full specification will include some far reaching components that are no more than a dream of developers just now (e.g. a universal video format). The specification drafts have been in use since 2003, and all major browsers are now on track to implement the latest changes as they emerge. You can check a browser’s readiness for the current state of the standard by navigating to http://acid3.acidtests.org/.

Popular browsers are compared at http://www.quirksmode.org/compatibility.html. You’ll see at this page that most current browsers (and even several earlier versions) implement most of the current specifications. All of the HTML5 features described in these notes are fully implemented by most current browsers.

The features HTML 5 brings to web development can be discussed under a few groups:

- New mark-up elements
- New form elements

- Local storage and micro-data
- Audio and Video standards
- Geo-location features
- Offline apps and the HTML5 manifest
- Native browser graphics

Some of these elements are purely html mark-up elements (i.e. they are used entirely inside HTML documents), and others need Javascript code to make them do their job. We'll look into each of them in some detail.

HTML 5 Mark-up Elements

These are the easiest additions to the HTML to get an understanding of. The main idea of these elements is to make document mark-up more semantically clear. For example, web developers currently use a load of different techniques – divs, frames, tables, style-sheet descriptions etc. - to mark up key sections of a document in HTML: headers, footers, navigation bars and sidebars. These names have now been absorbed into HTML, so when you see a section of a document inside a <header>…</header> tag pair, you can be pretty sure that it is a header for the current page and that it will be rendered in a way appropriate to a header by any sensible browser. In the words of the HTML 5 specification for the header element a header represents…

> "a group of introductory or navigational aids. A header element typically contains the section's heading (an h1–h6 element or an hgroup element), but can also contain other content, such as a table of contents, a search form, or any relevant logos."

The following is a list of HTML 5 mark-up tags associated with document mark-up:

Tag	Description	Example
<article>	This tag encloses independent, self-contained content. e.g. a blog might contain a sequence of articles, or per day	<article> <h2>HTML 5 Mark-up tags</h2. HTML 5 adds a number of new…. </article>
<aside>	This tag encloses content that is related to the main page content but could be considered to be a separate subject area	<aside> HTML5 tags can be validated using the facilities ate validator.w3.org </aside>
<details>	This tag encloses additional details that the user can choose to show or hide.	<details>Copyright © A.McMonnies, alistair.mcmonnies@uws.ac.uk<details>
<figure>	This tag should surround mark-up of figures and diagrams (imported using the <img> tag	<figure></figure>
<figcaption>	This tag should be used to apply a caption to a figure entry.	<figure><figcaption>Fig 11: The New Logo</figcaption></figure>
<header>	This should surround the header material for a document or section. A document can have several headers.	<header> HTML 5 Tags Published: September 2012 </header>
<hgroup>	This tag can be used to group several related heading elements.	<hgroup> <h1> Nesting Tags </h1> <h2> Methods to make documents

		easier to manage</h2> </hgroup>
<mark>	This is used to highlight (mark) specific bits of text	Using the <mark>mark</mark> tag
<nav>	This tag is used to surround a group of links to other pages.	<nav> Javascript The DOM Chrome Tools </nav>
<output>	This tag can be used to enclose the output of calculations (e.g. from a script)	<output name="answer"></output>
<section>	A multi-section document can have each section enclosed in this tag.	<section> The Console The Browser console is… </section>
<summary>	This tag can contain a summary of a page's content.	<summary>This article was about HTML5</summary>

Table 6.2: HTML5 Document Mark-up tags

Using the tags shown in Table 6.2 in conjunction with tags from more established HTML, the structure of a document can be made clearer and easier to navigate, either by developers or HTML 'readers' (typically applications that can read out web pages to users with impaired vision).

While the new mark-up tags can make a big improvement to how HTML documents are organised, there is nothing in them that could not have been done using existing tags in earlier HTML standards. In fact, in use these tags do little to the way a document appears. Instead, they provide markers for how a document should be structured. These are important both as a guide to document creators and as markers within a document that make it easy to extract information about it: for example, a list of the individual sections or articles in a document can be easily extracted to build a table of contents.

The new mark-up tags need to be used with CSS definitions if they are to be used to assert a typographical style in a document. However, some of the other new HTML features can change the things a browser can do radically, as we'll see.

Local Storage

HTML apps have, until version 5, been limited by their inability to store worthwhile amounts of data on the local machine. Web applications have long had the ability to store very small amounts of data (in the form of cookies) locally, or larger amounts of data at a server. However, storing data at the server is in itself a limitation: an application is only usable when there is a web connection, and if there are a million users, all of their data must be stored at the server. When a user logs on to a site that stores their personal data (e.g. emails or purchases), that data must be downloaded and any updates to it uploaded to the site. All these round-trips to the server consume bandwidth, time and server storage. The immediate benefits of using local storage are:

- Reduced network traffic
- Faster data access
- Application state can be restored quickly at start-up
- Short-term storage (i.e. caching) of downloaded data to save having to re-download it
- No need for the complex server software needed to identify users and secure their individual data from other users

HTML5 supports two local storage mechanisms: simple storage can save key-value pairs of string data on the client machine; database storage can store and access more structured information. Depending on the browser settings, the maximum amount of data that can be stored (per user, per application) can be anywhere between 5Mbytes and 10MBytes.

HTML5 Simple Storage

Javascript code is needed to access localStorage on a machine. The data is stored in string format as key:value pairs, where (typically) a string is used as a key to access matching data. For example:

```
localStorage.setItem("user-NAME", "Fred Bloggs");
```

At some future point, the user's name can be retrieved by:

```
var name = localStorage.getItem("user-name");
```

Of course, there are few situations in an application where you would store only literal strings on a device – the two statements shown above are examples used for clarity rather that realism. A more typical use is to store whole object-models, but since these are not usually single strings, an additional step is required to store objects. Listing 1 shows how a simple object can be stored on the client, and listing 2 shows how it can be retrieved. Note that objects of **any** size can be stored and loaded in this way, with few limitations.

```
var aPerson = { "name": "Joe Bloggs", "email": "joe@bloggo.com",
                "dob": "25 March 1982"};
localStorage.setItem("joePerson", JSON.stringify("aPerson"));
```

Listing 6.1: Storing a whole object using localStorage and JSON

```
var retrievedPerson = localStorage.getItem("joePerson");
console.log("Person: ", JSON.parse(retrievedPerson));
```

Listing 6.2: Retrieving an object from localStorage using JSON

JSON is the acronym for JavaScript Object Notation (some people misname it Java Simple Object Notation), and is effectively a mechanism for turning a Javascript object into a string and vice-versa. Since strings are text, and the web is a text-based network, JSON is frequently used to transmit object data between client and server machines. Its other use is for converting complex object structures into a flat-format that is easy to store in a text file.

So far so good – HTML 5's local storage is easy to use, fast and provides adequate storage space for significant amounts of data. However, it is not good practice to simply assume localStorage is available; your web-app will almost certainly be accessed by older browsers which don't "do" localStorage, and crashing the app is no way to keep your users happy. W3C guidelines for local storage (or in fact any HTML5 enhancement) are to check that the facility exists before using it. This turns out to be fairly easy to do:

```
if(localStorage){   // if this is undefined, localStorage isn't supported
    var aPerson = { "name": "Joe Bloggs", "email": "joe@bloggo.com",
                                "dob": "25 March 1982"};
    localStorage.setItem("joePerson", JSON.stringify("aPerson"));
} else {
    alert("HTML5 local storage is not supported!");
}
```

Listing 6.3: Testing for the availability of localStorage

Of course if ***localStorage*** is not available, you will have to come up with an alternative in your web-app: reverting to storing data on the server is a common fall-back, but we'll not consider it here (this is a module about HTML5 after all).

Database Storage

When a large amount of data made up of many separate pieces is to be stored locally in support of an app, it is often necessary to access that data in a flexible way. In general PC software, you would use a database (Access or MySQL for example) to allow this. Part of the HTML5 specification is called the WebDB specification, and this defines how a database that supports a web application should be accessed.

Again, WebDB is a specification for technology that can only be used in web clients via Javascript code, although it turns out to be a bit more substantial than the localStorage API. WebDB includes functions for creating database tables, accessing the data in tables, transaction handling and updating data in tables. The advantages of using a WebDB in an app are:

- Large amounts of data can be accessed, filtered and updated using standard SQL (Structured Query Language statements), which many developers are familiar with
- Apps are able to use structured data (i.e. organised in tables), which improves the flexibility of the data. This enables a lot of the features commonly found in desktop applications – such as storing large amounts of information for analysis, retrieving items quickly, performing batch operations etc.
- Apps can be made ***transactional***, which effectively means that data manipulation can be done in a way that is fault tolerant and consistent (the acronym ACID: Atomic, Consistent, Isolated and Durable is used to describe this way of managing data)
- Data in a structured DB can be extracted in a flexible way to support different types of operation in an application.

Some browsers allow the user to decide how big a database can get to before the user is asked whether to allow more space. For example, in Safari's settings:

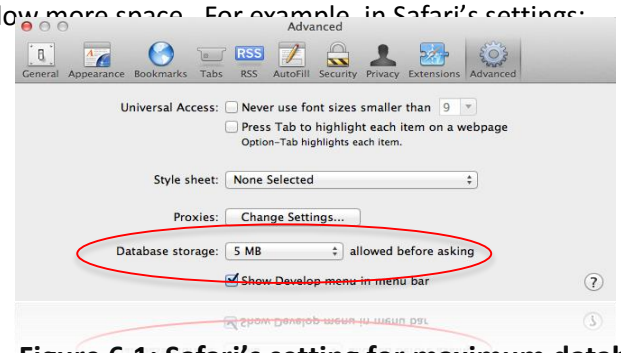

Figure 6.1: Safari's setting for maximum database size

WebDb is a more complex API to use so I'll not provide any examples here, but you'll find one on the

Moodle website for this module.

See http://www.html5rocks.com/en/tutorials/webdatabase/todo/ for a good tutorial on HTML5 data access using WebDB standards.

HTML5 Audio and Video

Web sites and applications have included multimedia for many years now. However, until HTML5 there has never been a ***standard*** way of using audio and video presentations, unless you count proprietary methods (such as Adobe Flash, Apple Quicktime or Microsoft Silverlight); currently the most widely used of these is Flash. The problem with proprietary technologies is that a) they are owned by large commercial companies who charge quite a lot for their development tools, b) being proprietary, they are not universally supported and c) the formats supported by the various commercial companies need that company's software to generate them.

So, for example, you can embed a Quicktime video on a web-page that can only be viewed on Apple machines or machines that have the Quicktime plug-in installed; the plug-in is free, but you have to accept Apple's terms of use to use it. If you wanted your web page with video and audio to work on every machine, you would have to provide alternatives so that Microsoft Windows machines could use Silverlight or Flash (again via a plug-in). This complicates the job of creating a multimedia site. A major example of this is that until recently many YouTube videos could not be accessed on an iPad or iPhone because they were in Flash video format. See http://support.google.com/youtube/bin/answer.py?hl=en&answer=56115 for the standard YouTube response to this.

The ideal situation would be that all audio files and all video files were in a format that all browsers could access. This is a tall order that could lead to world war, as Adobe, Microsoft and Apple fought it out for industry supremacy. There *is* a common open source format for audio (Ogg Vorbis) and video (Ogg Theora), but if you don't use a Linux machine you may never even have heard of these formats.

W3C has long recognised this limitation of the web as a multimedia platform, and has a long-term goal to fix it. The HTML5 standard includes mark-up tags for <video> and <audio>. Unfortunately, these do not access a universal format that every browser can work with, but they do at least provide a standard format for a page definition that includes audio and video elements. This replaces the <object> tag that has until now been used to incorporate audio and video elements via a proprietary code library.

What the <audio> and <video> tags do is to allow multimedia to be incorporated simply into pages that will play or display on a machine ***that has the correct codecs installed*** (note – a codec is a coder/decoder for multimedia files). The tags also provide for a fallback, so that if the required codec is not installed, the user's browser can be directed to a download link to install them.

This doesn't sound like an ideal solution, and it isn't since it does not mean that any browser can access any content. However, it does get around the age-old problem of badges on websites stating "Best viewed on Internet Explorer". Early drafts of the HTML5 specification mandated that all browsers should at least have built-in support for multimedia in two codecs: Ogg Vorbis for audio and Ogg Theora for movies. This stipulation was dropped after Apple and Nokia objected on the grounds that it would reduce uptake of their technologies (this is something that doesn't bother them at all when the same accusation is levelled at them - who's browser is it anyway?).

So, the <audio> and <video> tags include a facility so that multiple codecs can be supported. This does mean that if you want to support multiple formats, you have to host multiple versions of each video and audio file on your web server, but at least it reduces the amount of code you need to write to include the material on your page.

To include a video on a page (the process is similar for an <audio> tag), simply add the <video> html tag to the appropriate place in the file. Provided the end-user's browser supports the format you link, it will work:

```
<video src="myVideo.ogv">
    If your browser does not support this, download the video file
    <a href=myVideo.ogv>here</a>
</video>
```

Listing 6.4: A <video> tag with a fall-back option.

Note that a normal <href> tag has been placed within the <video> tag so that the user can click on a download link if their browser does not support the Ogg Theora format linked to. To provide a video in a couple of different formats, an alternative style of mark-up can be used:

```
<video width="320" height="240" controls="controls">
  <source src="myVideo.mp4" type="video/mp4" />
  <source src="myVideo.ogg" type="video/ogg" />
  If your browser does not support this, download the video file
  <a href=myVideo.ogv>here</a>
</video>
```

Listing 6.5: Providing the same video in two different formats.

Note that the HTML code shown in listing 6.5 has ***width*** and ***height*** attributes defined for the video. This is good practice, since without them the browser won't know the size of the video until a good portion of it has been downloaded, and so the page would need to re-format in the browser once this information was available. Also note that the ***controls*** attribute will display Play, Pause, Seek, Volume and FullScreen control elements along with the video.

For more information on HTML5 video, see http://www.html5rocks.com/en/tutorials/video/basics/. For detailed information on HTML5 audio, see http://html5doctor.com/html5-audio-the-state-of-play/.

Geo-location APIs and operation

Not so long ago, there would have been no point to having a geo-location facility in HTML – browsers were big programs tied to desktop and laptop machines. However, your phone probably has a browser, and if it is a smartphone it probably also includes some software that would like to know where you are when you use it.

This has been a major revolution in web-application design; locale aware software. It is now common for the information shown in a browser page to be tailored to the location the page has been viewed at: Google Maps is an obvious example of this, but the Post Office, the Electoral Register, Dominos Pizzas, just about any bank's website, any retail store that has a website etc. all benefit from being able to find out where you are geographically, and tailor the displayed information so that you can find the nearest

branch, store or food.

HTML5 geo-location is actually based or several technologies and the browse's ability to select the appropriate one for your situation. If your device has a GPS chip installed (most current smartphones do), then the geo-location API ought to be able to find your position accurate to within 10 metres. If not but you are using a cell-phone, cell-mast triangulation (based on detecting the small time differences your phone signal takes to reach a number of masts in the locale) can place you to within a larger radius – typically between 100M and 1KM. If you are using a laptop PC, the location reported will be based on the IP address of the machine, so that may result in an accuracy of anything between hundreds of meters and a few miles.

Geo-location and Privacy

The availability of geo-location within a browser comes at a price – if you allow a website to access your location, you could compromise your own privacy. For example, say you called in a sickie to your employer so that you could go to a football match, and were then to access your email on your phone during the half-time break. With constantly-on geo-location, your employer would be able to determine via server logs where you were at what time, and may be inclined to sack you for taking time off fraudulently. Civil liberties groups also cite the possibility of the police or security services being able to track suspects through their Internet service provider or cell-phone provider.

To protect client privacy, the geo-location API can only be enabled by the user of a computer or device. Any service that wishes to use the device's location must ask for access to it; this is done through a standard dialog tied in to the API so that the site developer can't simply work around it.

For example, when I try to access the HTML5 Geo-location demo at html5demos.com/geo/, I get:

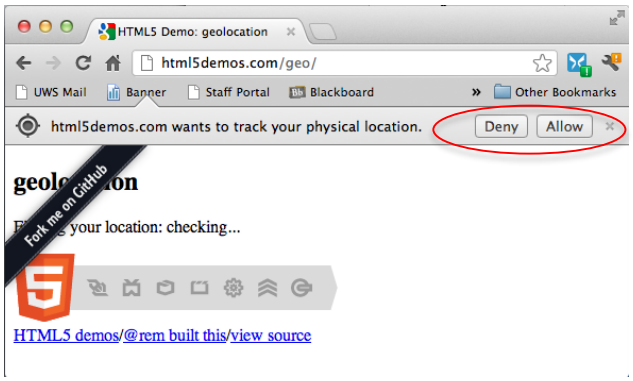

Figure 6.2: A web-page which wants to find where you are

If you were to press the ***Deny*** button at the top of the browser in figure 6.2, the page would not be able to access your location. If you press ***Allow***, it would be allowed to access where you are. Different browsers then manage this setting in different ways: typically an allowed site can continue to access your location until you revoke the permission in the browser's settings.

Once I click *Allow* in Chrome, the geo-location code can do its work, and the page refreshes to show:

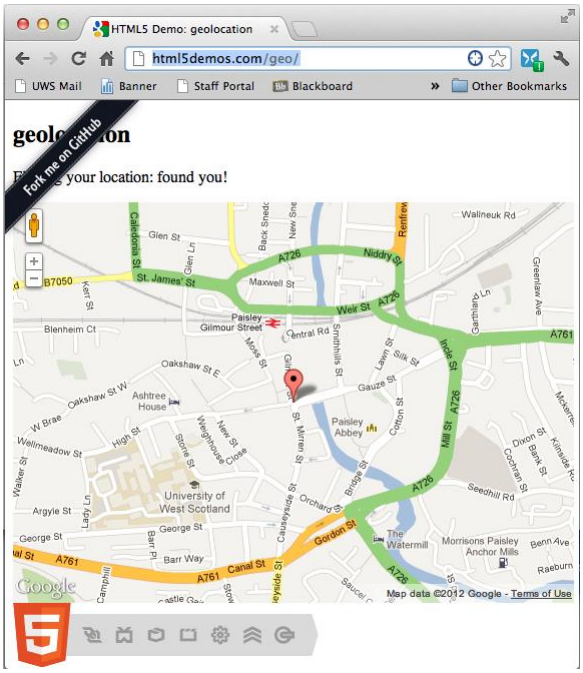

Figure 6.3: An app that has been allowed to access location data (note the icon in the URL bar)

Having allowed access to geo-location data, Chrome will put an extra icon in the URL bar (next to the Bookmark icon) so that you can quickly clear the setting or manage all of the browser's location settings. This is good design that has not been followed by Firefox (you need to go to the browser settings page to clear it) or Safari (same thing). However these browsers provide some good alternative options – for example, in Safari you can select how long to allow the geo-location functionality to be accessed for:

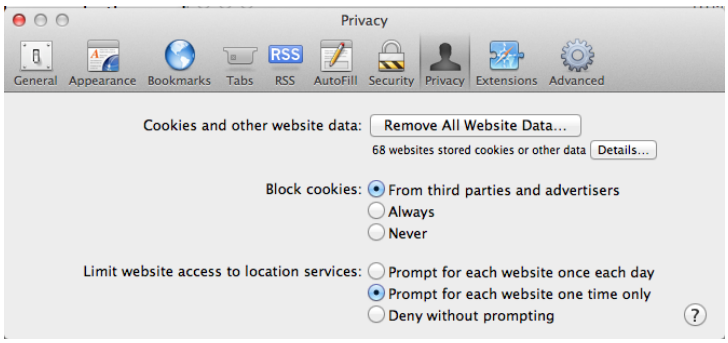

Figure 6.4: Safari's setting for location services

Coding geo-location

Geo-location can only be accessed and used in a web-app through Javascript code. To add location services to a page, the recommended method is to first check that the browser supports geo-location, then to try to access the location (the Allow/Deny dialog appears automatically at this point). Code for this is shown in listing 6.6:

```
// First define a function that calls the location service function..
function get_location() {
    if(navigator.geolocation){
        navigator.geolocation.getCurrentPosition(handle_location);
```

```
    } else {
        alert("Location services not supported or not allowed.");
    }
}
// Next provide a call-back function for when the result is available…
function handle_location(position) {
  var latitude = position.coords.latitude;
  var longitude = position.coords.longitude;
  // let's show a map or do something interesting!
}
// Finally, make sure it is called when the page is accessed…
window.onload = get_location;
```

Listing 6.6: Using the location services API

Note that the ***navigator.geolocation.getCurrentPosition()*** function requires a call-back. The function whose name is passed to ***getCurrentPosition()*** will be called when the locations services API has worked out where you are (it could take a little time).

What you do with the location data is up to your application. For example, you might display a Google map of that location (fairly easy to do), or find the nearest coffee shop or railway station (usually by accessing an online database of locations). A trivial example would be to discover whether you are in the northern or southern hemisphere:

```
function handle_location(position) {
    var latitude = position.coords.latitude;
    // We don't need the longitude for this.
    if(latitude > 0){
        alert("You are in the Northern hemisphere!");
    } else if(latitude < 0){
        alert("You are in the Southern hemisphere!");
    } else {
        alert("You are on the equator!");
    }
}
```

Listing 6.7: Doing something trivial with location data

One further thing you can do with the location services API is to track a user's location. This uses a different function call to set up the callback…

```
var id;
function follow_position(){
    if(navigator.geolocation){
        id = navigator.geolocation.watchPosition( handle_location );
    }
}
```

Listing 6.8: Keeping track of a browser's location

Using the code in listing 6.8, the browser will now call the ***handle_location()*** function periodically (whenever the API detects that the location has changed significantly). Note that the ID value returned from the ***watchPosition()*** function is used if you wish to stop the callback from being called (just call **navigator.geolocation.clearWatch(id);** - typically as the user navigates away from a page).

That is more or less all there is to location services. In the lab we'll have a look at integrating location services with Google maps.

HTML5 Forms and Form Elements

A form, in programming terms, is a group of user-input devices that operate together to allow a user to enter information on a web page. If you've ever signed up for an on-line service like an email account or online gas billing, you've used one.

Forms have been a specified part of HTML since 1993, so there is nothing new about them. However, they have always been a limited way of creating a user-interface within a web page, providing only a few different types of input. Web designers just had to become ingenious if they wanted to create an easy to operate user-interface; getting the user to enter anything other than simple text (including numbers) or more complicated than selecting an item from a short or long list usually meant hours of Javascript coding and testing.

HTML5 now specifies a good range of input devices that can be used on forms. The range of input devices now includes:

- Text input into a single-line textbox
- Text input into a multiple line textarea
- Entry boxes for numbers, with options for precision and validation
- Entering a value from a range using a slider
- Selecting a colour from a set of swatches
- Entering telephone numbers, email addresses or web addresses with validation
- Selecting dates from a calendar
- Entering and validating the time-of-day in international or local forms

The main point about HTML forms is that they are most useful for collecting data to send back to a server – something we will not touch on in this module. However, they are also useful for creating great user-interfaces for Javascript code, and we'll cover some of this in the labs.

As an example of the kind of thing you can expect from HTML form elements, listing 6.9 produces the output shown in figure 6.5:

```
<!DOCTYPE html>
<html>
<head>
   <title>Form Elements</title>
</head>
<body>

   <form>
      <p>Enter a number from 1 to 10: <input type="number" id="number" min="1" max="10"/></p>
      <p>Enter a word or phrase to search on:<input type="search" id="search" name="googlesearch"/></p>
      <p>Move the slider to select a percentage:<input type="range" id="range" min="0" max="100"/></p>
      <p>Enter a time of day:<input type="time" id="time"/></p>
      <p>Select a date:<input type="date" id="date"/></p>
   </form>
</body>
</html>
```

Listing 6.9: A full HTML web page with a form definition

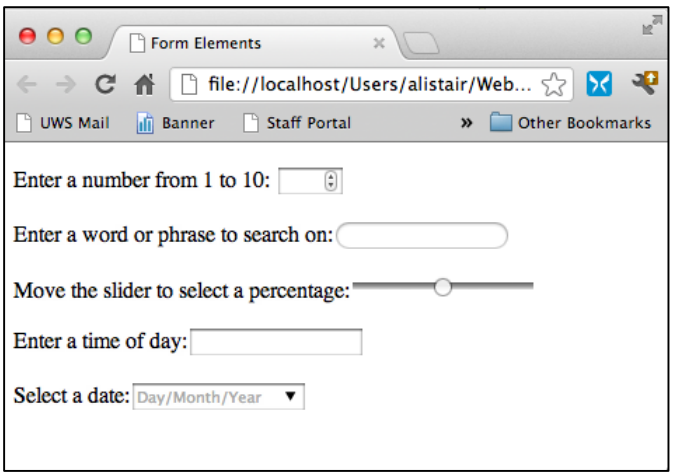

Figure 6.5: The resulting web-page form

For more information on HTML5 form elements, see http://diveintohtml5.info/detect.html#input-types and http://www.w3schools.com/html5/html5_form_input_types.asp.

Offline apps and the Manifest

So far, we've looked into HTML5 as a way of providing web-applications with the understanding that the apps have to operate from a server. That is after all what HTML was designed for. However the designers of HTML5 have recognised that often a web-page is simply an application that comes from a website; Google Docs illustrates this point beautifully – a fully functional set of office programs that just happen to run inside a browser.

When Google first introduced Google Docs, they first had to devise a robust and scalable way to allow a user to use an on-line word processor or spreadsheet in such a way that their work could survive an Internet outage. The result was Google Gears; a software library that supported (some) web applications in offline mode. Effectively, Gears emulated the online support that web apps (like Google Docs) needed – data storage, an engine for running Javascript scripts in parallel, desktop services and a local server that would serve HTML, Javascript and Image requests. This was a heavyweight set of libraries and took up a

fair amount of development and support time.

HTML5 now natively provides a similar (if differently organised) service; an application can continue to work offline provided that provision has been made for all of the resources that it depends on are available on the local machine. HTML has always made use of a cache (a folder of files maintained by the web browser) so that once you had downloaded a page, image or script, a copy of it would be in the local machine from where it could be re-loaded to save web traffic. The new Offline facility in HTML5 simply makes use of this cache in a slightly different way.

The Application Manifest

The word 'manifest' is most widely used in naval shipping: a manifest is a complete list of the cargo and/or passengers a ship is carrying, and is used as a way of checking for lost (or extra smuggled) cargo or missing (or stowaway) passengers.

That meaning works quite well to describe the purpose of a web-app's manifest. It is a simple text file linked to the HTML pages of a website that lists every other file and resource that the website uses. In a web app, there are three categories of file that would be added to the manifest:

- HTML files that the index page links to directly or indirectly
- Javascript files that the web app uses
- Content files – images, audio files, text files etc.

An HTML 5 manifest has to specify ***explicitly*** each file used in the app; by doing so, it allows an HTML5 compliant browser to pre-load these into the browser cache so that if the app goes offline, all of the content files are still available.

Making a web app capable of working in offline mode now becomes very simple – a 2 step process:

1. Include a link to the manifest file in the <html> tag of the every HTML page of a document.
2. Provide a manifest file that has a full list of all of the files that need to be downloaded to the browser cache.

For example, assume you have a web app that is organised as follows (taken from the WebStorm project window):

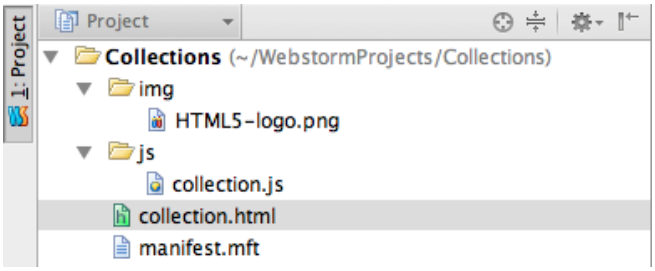

Figure 6.6: The full list of files used in a simple web application

If accessing these files online, the main page would be collection.html, which contains links to the other three files (img/HTML5-logo.png. js/collection.js and manifest.mft). If any of the files referred to be the main file was accidentally removed form the server, the app would not work fully (either it would not contain the graphic or there would be no script).

If we need the app to work offline, we clearly must make sure that all of the files are available and so the manifest file (called manifest.mft here, but there is no law that says it must be called that) provides a full list for the browser to download into the cache:

```
CACHE MANIFEST
# 2012-08-15:v1.0
# These files will be placed in the browser cache the first time
# the app is loaded.  From this point on, the application can be used.
collection.html
img/HTML5-logo.png
js/collection.js
```

Listing 6.10: The manifest file for the web app in figure 6.6

This is a fairly simple manifest file, containing only the core information needed. The first line (CACHE MANIFEST) is required and must be that text I that format. The second line is a date/version stamp – it is not mandated by the browser, but it would not be sensible to exclude it, because this line provides the browser with information that is important if an app is to be updated. When an app is loaded, even offline, the browser will attempt to re-download the manifest. If it can't, nothing is lost, but if it can, the browser can check to see if the date or version number has changed. If it has, it will re-populate the browser cache according to the new manifest file. This provides a very simple and effective version control mechanism.

Any line beginning with a # is treated as a comment (even the date/version stamp is considered a comment so a manifest that does not have one will still work). The last three lines are the important stuff – the name of (and path to) every file in the app. Paths can be relative (as here) or absolute – a complete URL. However the domain name part (e.g. http://myserver.com/) must be the same for all of the files (this is a basic world wide web requirement), so can be omitted.

More complex application manifests can be created to allow apps to download material from the server if it is available (i.e. if the app is online) but access fallback resources if it is not. See http://www.html5rocks.com/en/tutorials/appcache/beginner/ for a very good description of a more complex manifest.

By providing a manifest, a web app can be bookmarked in a browser and if the user accesses the bookmark the app will run, even if there is no Internet connection. In Smartphones, an app that can be used offline can be added in such a way that it behaves like a normal program running on the device. For example if you use an iPhone, you can go to an offline capable app in the Safari browser, and these use the browser's share button (⍐) followed by Add to Home Screen to mark the page as an app.

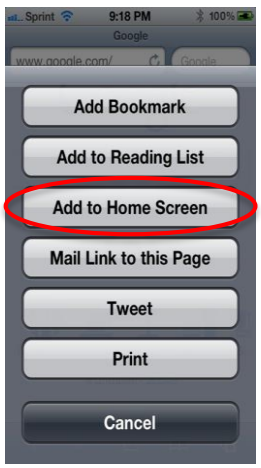

Figure 6.7: Turning a web-app into an app on the iPhone

Graphics in HTML5 – Canvas and Context

Until recently, putting graphics on a web page was a matter of linking to an image on the web server. Scalable Vector Graphics (SVG) has been around for a while as a way of having updatable images but some mainstream browsers didn't support it very well. What was always missing was a way of procedurally ***drawing*** on to a web page. In the vacuum of this, Adobe Flash became very popular, especially with developers.

However, Apple was never particularly keen on flash – Steve Jobs often accused Adobe of providing very inefficient Flash plug-ins for the Apple OS and browsers (see http://www.apple.com/hotnews/thoughts-on-flash/), to the extent of banning from being deployed on the iPhone and iPad. His opinion was that an iOS flash plug-in would guzzle processing cycles and drain the battery (users who have jail-broken their iPhone and installed Flash can confirm this).

As a consequence of their (boss's) perception of Flash, Apple went on to develop alternatives for the iOS device browser; these went on to become the core of HTML5. Major among the features Apple decided on for their pocket browser was a way of drawing on the screen. It would use Javascript as a scripting language, and a very lightweight form of mark-up (simply defining an area of the page to use). The HTML canvas is easy to use, quick in operation and extensive in features – an ideal feature to add to many types of web pages, from graphs and games to image processing and data visualization.

The canvas element is easy to insert on a page:

```
<canvas id="example" width="200" height="200">
    This text is displayed if your browser does not support HTML5 Canvas.
</canvas>
```

Listing 6.11: The HTML5 Canvas element

It can then be easily manipulated in Javascript code:

```
var example = document.getElementById('example');
var context = example.getContext('2d');
context.fillStyle = 'red';
context.fillRect(30, 30, 50, 50);
```

Listing 6.12: Javascript code to draw a red rectangle on a canvas

Resulting in…

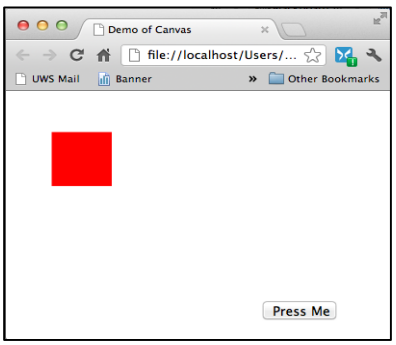

Figure 6.8: The result of the canvas code in listing 6.12

The canvas element is powerful enough to deserve a chapter, lecture and lab on its own – next week.

Questions

1. HTML authors have been producing documents with headers, footers, sections, navigation bars, summaries etc. since the early days of the web. Why are the new mark-up tags important?
2. Considering the code in listing 6.2, describe how you would arrange to save the data from two person objects: joePerson and fredPerson?
3. The **localStorage.getItem(key)** method will return either the data associated with the key, or **null** If the item doesn't exist in the store. How would you write code to show an **alert()** that contained either the data item or a message that it was not available?
4. HTML5 audio and video does not solve the problem of a browser not being equipped to play a file if it is not in the supported format. What is the benefit of it if it can't fix this problem?
5. Apple and Nokia objected to the use of a standard web video and audio format, yet both companies provide access to suitable codecs to match their proprietary formats for free. What benefit do they hope to gain by squashing the standard formats? What risk does their objection carry?
6. What additional code do you need to add to a site to give the user the option to allow or deny the site from accessing the browser's location data?
7. Location services can provide a browser with a simple record of the current location of a browser (in the form of two numbers – latitude and longitude). How do you think this could be used to display a map on the browser?
8. Web pages have long been capable of providing a way for a user to enter numbers, dates, times etc. What is the purpose of the new forms of input control?
9. A mobile application might collect data that can be used to update maps shown the position of local pet shops. What would be the point of providing this app with an off-line capability, when if it was off-line, it would not be able to update the maps?