4
HTML5 for Responsive Designs

HTML5 evolved from the Web Applications 1.0 project, started by the **Web Hypertext Application Technology Working Group** (**WHATWG**) before being later embraced by the W3C. Subsequently, large parts of the specification are weighted towards dealing with web applications. If you're not building web applications, that doesn't mean there aren't plenty of things in HTML5 you could (and indeed should) embrace when embarking on a responsive design. So, whilst some features of HTML5 are directly relevant to building better responsive web pages (for example, leaner code), others are outside our responsive remit.

HTML5 also provides specific tools for handling forms and user input. This set of features takes much of the burden away from more resource heavy technologies like JavaScript for things like form validation. However, we're going to look at HTML5 forms separately in *Chapter 8*, *Conquer Forms with HTML5 and CSS3*.

In this chapter, we will cover the following:

- What parts of HTML5 can we use right now?
- How to write HTML5 pages
- The economies of using HTML5
- Obsolete HTML features
- New semantic HTML5 elements
- Using Web Accessibility Initiative - Accessible Rich Internet Applications (WAI-ARIA) for increased semantics and aiding assistive technologies
- Embedding media
- Responsive HTML5 and iFrame videos
- Making a website available offline

What parts of HTML5 can we use today?

Although the full specification of HTML5 is yet to be ratified, most new features of HTML5 are already supported, to varying degrees, by modern web browsers including Apple's Safari, Google Chrome, Opera, and Mozilla Firefox and even Internet Explorer 9! So, whilst it's improbable everything in the current draft of the HTML5 specification will survive until recommendation by the W3C, there are plenty of new features that can be implemented right now.

Most sites can be written in HTML5

Currently, if I'm tasked to build a website, my default markup would be HTML5 rather than HTML 4.01. Where the opposite was the case only a few years ago, at present, there has to be a compelling reason not to markup a site in HTML5. All modern browsers understand common HTML5 features with no problems (the new structural elements, video and audio tags) and older versions of IE can be served **polyfills** to address all of the shortfalls I have encountered.

What are polyfills?

The term polyfill was originated by Remy Sharp as an allusion to filling the cracks in older browsers with Polyfilla (known as Spackling Paste in the US). Therefore, a polyfill is a JavaScript shim that effectively replicates newer features in older browsers. However, it's important to appreciate that polyfills add extra flab to your code. Therefore, just because you can add three polyfill scripts to make Internet Explorer 6 render your site the same as every other browser doesn't mean you necessarily should!

Polyfills, shims, and Modernizr

Ordinarily, older versions of Internet Explorer (pre v9) have no understanding of any of the new semantic elements of HTML5. However, some time ago, Sjoerd Visscher discovered that if elements are created with JavaScript first, Internet Explorer is able to recognize and style them accordingly. Armed with this knowledge, JavaScript whiz Remy Sharp created a lightweight enabling script (`http://remysharp.com/2009/01/07/html5-enabling-script/`) that, if included in an HTML5 page, magically switched these elements on for older versions of Internet Explorer. For a long time, pioneers of HTML5 would stick this script in their markup to enable users viewing in Internet Explorer 6, 7, and 8 to enjoy a comparable experience.

However, things have now progressed significantly. There's now a new kid on the block that does all this and a whole lot more. Its name is Modernizr (`http://www.modernizr.com`) and if you're writing pages in HTML5, it's well worth your attention. Besides enabling HTML5 structural elements for IE, it also provides the ability to conditionally load further polyfills, CSS files, and additional JavaScript files based on a number of feature tests.

So, as there are few good reasons for not using HTML5, let's get going and start writing some markup, HTML5 style.

Want a shortcut to great HTML5 code? Consider the HTML5 Boilerplate

If time is short and you need a good starting point for your project, consider using the HTML5 Boilerplate (`http://html5boilerplate.com/`). It's a pre-made "best practice" HTML5 file, including essential styles (such as the aforementioned normalize.css), polyfills, and tools such as Modernizr. It also includes a build tool that automatically concatenates CSS and JS files and strips comments to create production ready code. Highly recommended!

How to write HTML5 pages

Open an existing web page. There is a chance that the first few lines will look something like this:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8" />
```

Delete the preceding code snippet and replace it with the following code snippet:

```
<!DOCTYPE html>
<html lang="en">
<head>
<meta charset=utf-8>
```

Save the document and you should now have your first HTML5 page as far as the W3C validator is concerned (`http://validator.w3.org/`).

Don't worry, that's not the end of the chapter! That crude exercise is merely meant to demonstrate HTML5's flexibility. It's an evolution of the markup you already write, not a revolution. We can use it to supercharge the markup that we already know how to write.

So, what did we actually do there? First of all, we stated the new HTML5 Doctype declaration:

```
<!DOCTYPE html>
```

If you're a fan of lowercase, then `<!doctype html>` is just as good. It makes no difference.

HTML5 Doctype—why so short?

The HTML5 `<!DOCTYPE html>` Doctype is so short that this was determined to be the shortest method of telling a browser to render the page in "standard mode". This most efficient syntax mindset is prevalent throughout much of HTML5.

After the Doctype declaration, we opened the HTML tag, specified the language, and then opened the `<head>` section:

```
<html lang="en">
<head>
```

Sprechen sie Deutsche?

According to the W3C specifications (`http://dev.w3.org/html5/spec/Overview.html#attr-lang`), the `lang` attribute specifies the primary language for the element's contents and for any of the element's attributes that contain text. If you're not writing pages in English, you'd best specify the correct language code. For example, for Japanese the HTML tag would be `<html lang="ja">`. For a full list of languages take a look at `http://www.iana.org/assignments/language-subtag-registry`.

Finally, we specified the character encoding. As it's a void element it doesn't require a closing tag:

```
<meta charset=utf-8>
```

Unless you have a good reason to specify otherwise, it's almost always UTF-8.

Economies of using HTML5

I remember, in school, every so often our super-mean (but actually very good) math teacher would be away. The class would breathe a collective sigh of relieve as, rather than "Mr Mean" (names have been changed to protect the innocent), the replacement was usually an easy-going and amiable man who sat quietly, leaving us to get on without shouting or constant needling. He didn't insist on silence whilst we worked, he didn't much care how elegant our workings were on the page – all that mattered was the answers. If HTML5 were a math teacher, it would be that easy-going supply teacher. I'll qualify this bizarre analogy…

If you pay attention to how you write code, you'll typically use lowercase for the most part, wrap attribute values in quotation marks, and declare a "type" for scripts and stylesheets. For example, you might link to a stylesheet like this:

```
<link href="CSS/main.css" rel="stylesheet" type="text/css" />
```

HTML5 doesn't require such detail, it's just as happy to see this:

```
<link href=CSS/main.css rel=stylesheet >
```

I know, I know. It makes me feel weird, too. There's no end tag/slash, there are no quotation marks around the attribute values, and there is no `type` declaration. However, easy going HTML5 doesn't care. The second example is just as valid as the first.

This more lax syntax applies across the whole document, not just linked CSS and JavaScript elements. For example, specify a div like this if you like:

```
<div id=wrapper>
```

That's perfectly valid HTML5. The same goes for inserting an image:

```
<img SRC=frontCarousel.png aLt=frontCarousel>
```

That's also valid HTML5. No end tag/slash, no quotes, and a mix of capitalization and lower case characters. You can even omit things such as the opening `<head>` tag and the page still validates. What would XHTML 1.0 say about this!

A sensible approach to HTML5 markup

Although we are aiming to embrace a mobile first mindset for our responsive web pages and designs, I'll admit I can't fully let go of writing what I consider the best practice markup (note, in my case that was adhering to the XHTML 1.0 markup standards which required XML syntax). It's true that we can lose some minute amounts of data from our pages by embracing these coding economies but in all honesty, if necessary, I'll make up the shortfall by leaving an image out of my design instead!

For me, the extra characters (end slashes and quotes around attribute values) are worth it for increased code legibility. When writing HTML5 documents therefore I tend to fall somewhere between the old style of writing markup (which is still valid code as far as HTML5 is concerned, although it may generate warnings in validators/conformance checkers) and the economies afforded by HTML5. To exemplify, for the CSS link above, I'd go with the following:

```
<link href="CSS/main.css" rel="stylesheet"/>
```

I've kept the closing tag and the quotation marks but omitted the `type` attribute. The point to make here is that you can find a level you're happy with yourself. HTML5 won't be shouting at you, flagging up your markup in front of the class and standing you in a corner for not validating.

All hail the mighty <a> tag

One more really handy economy in HTML5 is that we can now wrap multiple elements in an `<a>` tag. (Woohoo! About time, right?) Previously, if you wanted your markup to validate, it was necessary to wrap each element in its own `<a>` tag. For example, see the following code snippet:

```
<h2><a href="index.html">The home page</a></h2>
<p><a href="index.html">This paragraph also links to the home page</
a></p>
<a href="index.html"><img src="home-image.png" alt="home-slice" /></a>
```

However, we can ditch all the individual `<a>` tags and instead wrap the group as demonstrated in the following code snippet:

```
<a href="index.html">
<h2>The home page</h2>
<p>This paragraph also links to the home page</p>
<img src="home-image.png" alt="home-slice" />
</a>
```

The only limitations to keep in mind are that, understandably, you can't wrap one `<a>` tag within another `<a>` tag and you can't wrap a form in an `<a>` tag either.

Obsolete HTML features

Besides things such as the language attributes in script links, there are some further parts of HTML you may be used to using that are now considered "obsolete" in HTML5. It's important to be aware that there are two camps of obsolete features in HTML5—conforming and non-conforming. Conforming features will still work but will generate warnings in validators. Realistically, avoid them if you can but they aren't going to make the sky fall down if you do use them. Non-conforming features may still render in certain browsers but if you use them, you are considered very, very naughty and you might not get a treat at the weekend!

An example of an obsolete but conforming feature would be using a border attribute on an image. This was historically used to stop images showing a blue border about them if they were nested inside a link. For example, see the following:

```
<img src="frontCarousel.png" alt="frontCarousel" border="0" />
```

Instead, you are advised to use CSS instead for the same effect.

In terms of obsolete and non-conforming features, there is quite a raft. I'll confess that many I have never used (some I've never even seen!). It's possible you may experience a similar reaction. However, if you're curious, you can find the full list of obsolete and non-conforming features at `http://dev.w3.org/html5/spec/Overview.html#non-conforming-features`. Notable obsolete and non-conforming features are `strike`, `center`, `font`, `acronym`, `frame`, and `frameset`.

New semantic elements in HTML5

My dictionary defines semantics as "the branch of linguistics and logic concerned with meaning". For our purposes, semantics is the process of giving our markup meaning. Why is this important? Glad you asked. Consider the structure of our current markup for the *And the winner isn't...* site:

```
<body>
<div id="wrapper">
  <div id="header">
    <div id="logo"></div>
    <div id="navigation">
      <ul>
        <li><a href="#">Why?</a></li>
      </ul>
```

```
        </div>
      </div>
      <!-- the content -->
      <div id="content">

      </div>
      <!-- the sidebar -->
      <div id="sidebar">

      </div>
      <!-- the footer -->
      <div id="footer">

      </div>
    </div>
    </body>
```

Most writers of markup will see common conventions for the ID names of the div's used—header, content, sidebar, and so on. However, as far as the code itself goes, any user agent (web browser, screen reader, search engine crawler, and so on) looking at it couldn't say for sure what the purpose of each div section is. HTML5 aims to solve that problem with new semantic elements. From a structure perspective these are explained in the sections that follow.

The <section> element

The `<section>` element is used to define a generic section of a document or application. For example, you may choose to create sections round your content; one section for contact information, another section for news feeds, and so on. It's important to understand that it isn't intended for styling purposes. If you need to wrap an element merely to style it, you should continue to use a `<div>` as you would have before.

> To find out what the W3C HTML5 specification says about `<section>`, go to the following URL:
>
> `http://dev.w3.org/html5/spec/Overview.html#the-section-element`

The <nav> element

The `<nav>` element is used to define major navigational blocks—links to other pages or to parts within the page. As it is for use in major navigational blocks it isn't strictly intended for use in footers (although it can be) and the like, where groups of links to other pages are common.

> To find out what the W3C HTML5 specification says about `<nav>`, go to the following URL:
>
> `http://dev.w3.org/html5/spec/Overview.html#the-nav-element`

The <article> element

The `<article>` element, alongside `<section>` can easily lead to confusion. I certainly had to read and re-read the specifications of each before it sank in.
The `<article>` element is used to wrap a self-contained piece of content. When structuring a page, ask whether the content you're intending to use within a `<article>` tag could be taken as a whole lump and pasted onto a different site and still make complete sense? Another way to think about it is would the content being wrapped in `<article>` actually constitute a separate article in a RSS feed? The obvious example of content that should be wrapped with an `<article>` element would be a blog post. Be aware that if nesting `<article>` elements, it is presumed that the nested `<article>` elements are principally related to the outer article.

> What the W3C HTML5 specification says about `<article>`:
>
> `http://dev.w3.org/html5/spec/Overview.html#the-article-element`

The <aside> element

The `<aside>` element is used for content that is tangentially related to the content around it. In practical terms, I often use it for sidebars (when it contains suitable content). It's also considered suitable for pull quotes, advertising, and groups of navigation elements (such as Blog rolls, and so on).

For more on what the W3C HTML5 specification says about `<aside>`, visit:

`http://dev.w3.org/html5/spec/Overview.html#the-aside-element`

The <hgroup> element

If you have a number of headings, taglines and subheadings in `<h1>`,`<h2>`,`<h3>`, and the subsequent tags then consider wrapping them in the `<hgroup>` tag. Doing so will hide the secondary elements from the HTML5 outline algorithm as only the first heading element within an `<hgroup>` contributes to the documents outline.

The HTML5 outline algorithm

HTML5 allows each sectioning container to have its own self-contained outline. This means it's no longer necessary to think constantly about which level of header tag you're at. For example, within a blog, I can set my post titles to use the `<h1>` tag, whilst my blog title itself also has a `<h1>` tag. For example, consider the following structure:

```
<hgroup>
  <h1>Ben's blog</h1>
  <h2>All about what I do</h2>
</hgroup>
  <article>
    <header>
      <hgroup>
        <h1>A post about something</h1>
        <h2>Trust me this is a great read</h2>
        <h3>No, not really</h3>
        <p>See. Told you.</p>
      </hgroup>
    </header>
  </article>
```

Despite having multiple `<h1>` and `<h2>` headings, the outline still appears as follows:

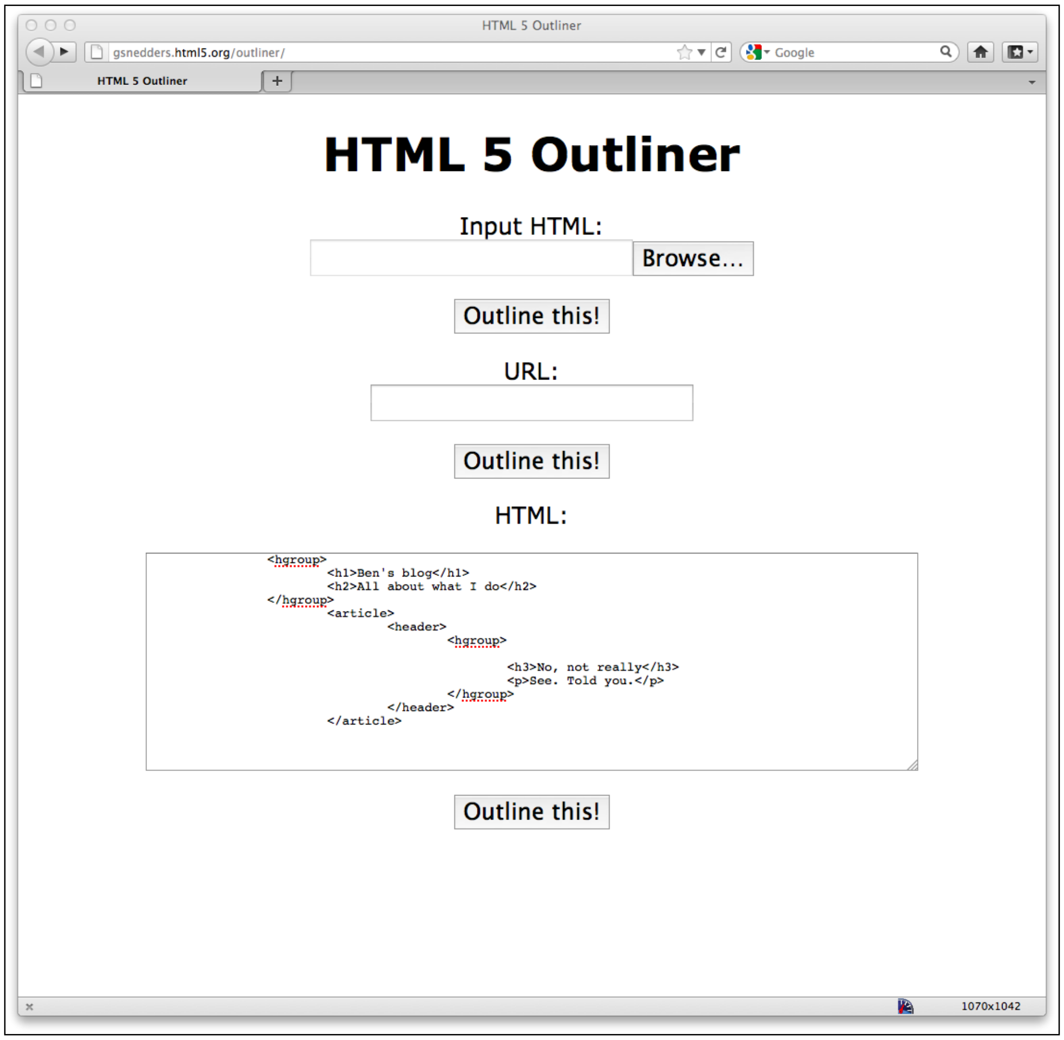
- Ben's blog
    - A post about something

As such, you don't need to keep track of the heading tag you need to use. You can just use whatever level of heading tag you like within each piece of sectioned content and the HTML5 outline algorithm will order it accordingly.

You can test the outline of your documents using HTML5 outliners at one the following URLs:

- `http://gsnedders.html5.org/outliner/`
- `http://hoyois.github.com/html5outliner/`

The following screenshot shows the HTML 5 Outliner page:



```
<hgroup>
        <h1>Ben's blog</h1>
        <h2>All about what I do</h2>
</hgroup>
        <article>
                <header>
                        <hgroup>

                                <h3>No, not really</h3>
                                <p>See. Told you.</p>
                        </hgroup>
                </header>
        </article>
```

For more on what the W3C HTML5 specification says about `<hgroup>`, visit:

`http://dev.w3.org/html5/spec/Overview.html#the-hgroup-element`

The <header> element

The `<header>` element doesn't take part in the outline algorithm so can't be used to section content. Instead it should be used as an introduction to content. Practically, the `<header>` can be used for the "masthead" area of a site's header but also as an introduction to other content such as an introduction to a `<article>` element.

What the W3C HTML5 specification says about <header>:

`http://dev.w3.org/html5/spec/Overview.html#the-header-element`

The <footer> element

Like the `<header>`, the `<footer>` element doesn't take part in the outline algorithm so doesn't section content. Instead it should be used to contain information about the section it sits within. It might contain links to other documents or copyright information for example. Like the `<header>` it can be used multiple times within a page if needed. For example, it could be used for the footer of a blog but also a footer within a blog post `<article>`. However, the specification notes that contact information for the author of a blog post should instead be wrapped by an`<address>` element.

What the W3C HTML5 specification says about <footer>:

`http://dev.w3.org/html5/spec/Overview.html#the-footer-element`

The <address> element

The `<address>` element is to be used explicitly for marking up contact information for its nearest `<article>` or `<body>` ancestor. To confuse matters, keep in mind that it **isn't** to be used for postal addresses and the like unless they are indeed the contact addresses for the content in question. Instead postal addresses and other arbitrary contact information should be wrapped in good ol' `<p>` tags.

[For more on what the W3C HTML5 specification says about `<address>`:

`http://dev.w3.org/html5/spec/Overview.html#the-address-element`]

Practical usage of HTML5's structural elements

Let's look at some practical examples of these new elements. I think the `<header>`, `<nav>`, and `<footer>` elements are pretty self explanatory so for starters, let's take the current *And the winner isn't...* home page markup and amend the header, navigation, and footer areas (see highlighted areas in the following code snippet):

```
<!DOCTYPE html>
<html lang="en">
<head>
<meta charset=utf-8>
<meta name="viewport"  content="width=device-width,initial-scale=1.0"
/>
<title>And the winner isn't…</title>
<script>document.cookie='resolution='+Math.max(screen.width,screen.
height)+'; path=/';</script>
<link href="css/main.css" rel="stylesheet" />

</head>

<body>

<div id="wrapper">
  <!-- the header and navigation -->
  <header>
    <div id="logo">And the winner is<span>n't...</span></div>
    <nav>
      <ul>
        <li><a href="#">Why?</a></li>
        <li><a href="#">Synopsis</a></li>
```

```
          <li><a href="#">Stills/Photos</a></li>
          <li><a href="#">Videos/clips</a></li>
          <li><a href="#">Quotes</a></li>
          <li><a href="#">Quiz</a></li>
        </ul>
      </nav>
    </header>
    <!-- the content -->
    <div id="content">
      <img class="oscarMain" src="img/oscar.png" alt="atwi_oscar" />
      <h1>Every year <span>when I watch the Oscars I'm annoyed...</
  span></h1>
      <p>that films like King Kong, Moulin Rouge and Munich get the
  statue whilst the real cinematic heroes lose out. Not very Hollywood
  is it?</p>
  <p>We're here to put things right. </p>
    <a href="#">these should have won &raquo;</a>
    </div>
    <!-- the sidebar -->
    <div id="sidebar">
      <div class="sideBlock unSung">
        <h4>Unsung heroes...</h4>
        <a href="#"><img src="img/midnightRun.jpg" alt="Midnight Run"
  /></a>
        <a href="#"><img class="sideImage" src="img/wyattEarp.jpg"
  alt="Wyatt Earp" /></a>
      </div>
      <div class="sideBlock overHyped">
        <h4>Overhyped nonsense...</h4>
        <a href="#"><img src="img/moulinRouge.jpg" alt="Moulin Rouge"
  /></a>
        <a href="#"><img src="img/kingKong.jpg" alt="King Kong" /></a>
      </div>
    </div>
    <!-- the footer -->
    <footer>
      <p>Note: our opinion is absolutely correct. You are wrong, even if
  you think you are right. That's a fact. Deal with it.</p>
    </footer>

  </div>
  </body>
  </html>
```

As we've seen however, where articles and sections exist within a page, these elements aren't restricted to one use per page. Each article or section can have its own header, footer, and navigation. For example, if we add a `<article>` element into our markup, it might look as follows:

```
<body>

<div id="wrapper">
  <!-- the header and navigation -->
  <header>
    <div id="logo">And the winner is<span>n't...</span></div>
    <nav>
      <ul>
        <li><a href="#">Why?</a></li>
      </ul>
    </nav>
  </header>
  <!-- the content -->
  <div id="content">
    <article>
      <header>An article about HTML5</header>
      <nav>
        <a href="1.html">related link 1</a>
        <a href="2.html">related link 2</a>
      </nav>
      <p>here is the content of the article</p>
      <footer>This was an article by Ben Frain</footer>
    </article>
```

As you can see in the preceding code, we are using a `<header>`, `<nav>`, and `<footer>` for both the page and also the article contained within it.

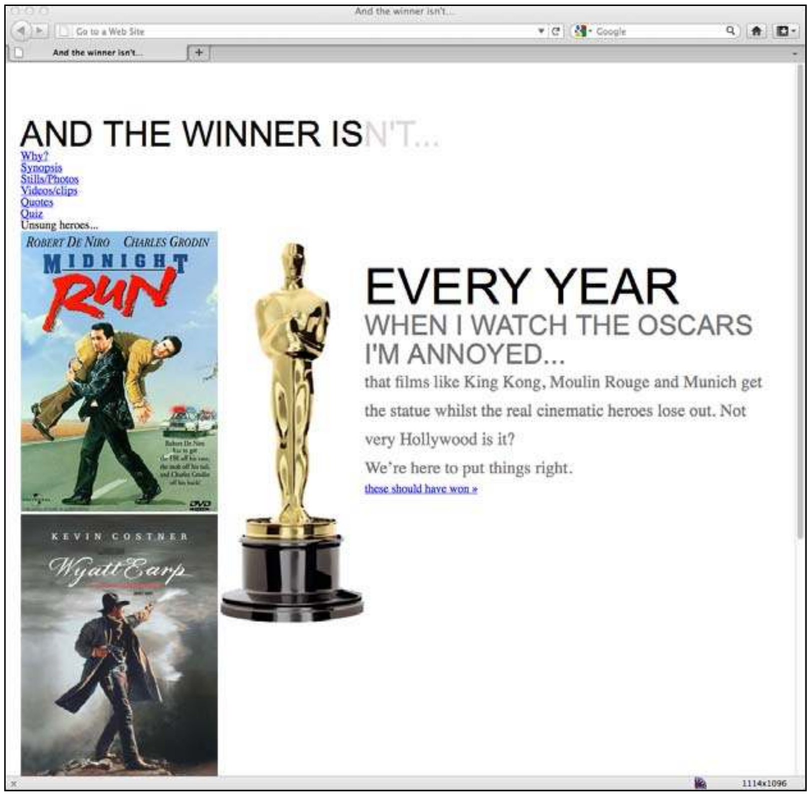
Let's amend our sidebar area. This is what we have at the moment in HTML 4.01 markup:

```
<!-- the sidebar -->
  <div id="sidebar">
    <div class="sideBlock unSung">
      <h4>Unsung heroes...</h4>
      <a href="#"><img src="img/midnightRun.jpg" alt="Midnight Run"
/></a>
      <a href="#"><img class="sideImage" src="img/wyattEarp.jpg"
alt="Wyatt Earp" /></a>
    </div>
    <div class="sideBlock overHyped">
      <h4>Overhyped nonsense...</h4>
      <a href="#"><img src="img/moulinRouge.jpg" alt="Moulin Rouge"
/></a>
      <a href="#"><img src="img/kingKong.jpg" alt="King Kong" /></a>
    </div>
  </div>
```

Our sidebar content is certainly "tangentially" related to the main content, so first of all, let's remove `<div id="sidebar">` and replace it with `<aside>`:

```
<!-- the sidebar -->
  <aside>
    <div class="sideBlock unSung">
      <h4>Unsung heroes...</h4>
      <a href="#"><img src="img/midnightRun.jpg" alt="Midnight Run"
/></a>
      <a href="#"><img class="sideImage" src="img/wyattEarp.jpg"
alt="Wyatt Earp" /></a>
    </div>
    <div class="sideBlock overHyped">
      <h4>Overhyped nonsense...</h4>
      <a href="#"><img src="img/moulinRouge.jpg" alt="Moulin Rouge"
/></a>
      <a href="#"><img src="img/kingKong.jpg" alt="King Kong" /></a>
    </div>
  </aside>
```

Excellent! However, if we take a look in the browser you'd be forgiven for letting a minor expletive slip out…



AND THE WINNER ISN'T...

Why?
Synopsis
Stills/Photos
Videos/clips
Quotes
Quiz
Unsung heroes...

EVERY YEAR
WHEN I WATCH THE OSCARS
I'M ANNOYED...
that films like King Kong, Moulin Rouge and Munich get the statue whilst the real cinematic heroes lose out. Not very Hollywood is it?
We're here to put things right.
these should have won »

Talk about one step forward and two steps back! The reason is we haven't been and amended the CSS to match the new elements. Let's do that now before we proceed. We need to amend all references to `#header` to be simply `header`, all references to `#navigation` to be `nav`, and all references to `#footer` to be `footer`. For example, the first CSS rule relating to the header will change from:

```
#header {
  background-position: 0 top;
  background-repeat: repeat-x;
  background-image: url(../img/buntingSlice3Invert.png);
  margin-right: 1.0416667%; /* 10 ÷ 960 */
  margin-left: 1.0416667%; /* 10 ÷ 960 */
  width: 97.9166667%; /* 940 ÷ 960 */
}
```

To become:

```
header {
  background-position: 0 top;
  background-repeat: repeat-x;
  background-image: url(../img/buntingSlice3Invert.png);
  margin-right: 1.0416667%; /* 10 ÷ 960 */
  margin-left: 1.0416667%; /* 10 ÷ 960 */
  width: 97.9166667%; /* 940 ÷ 960 */
}
```

This was particularly easy for the header, navigation, and footer as the IDs were the same as the element we were changing them for – we merely omitted the initial '#'. The sidebar is a little different: we need to change references from `#sidebar` to `aside` instead. However, performing a "find and replace" in the code editor of your choice will help here. To clarify, rules like the following:

```
#sidebar {
}
```

Will become:

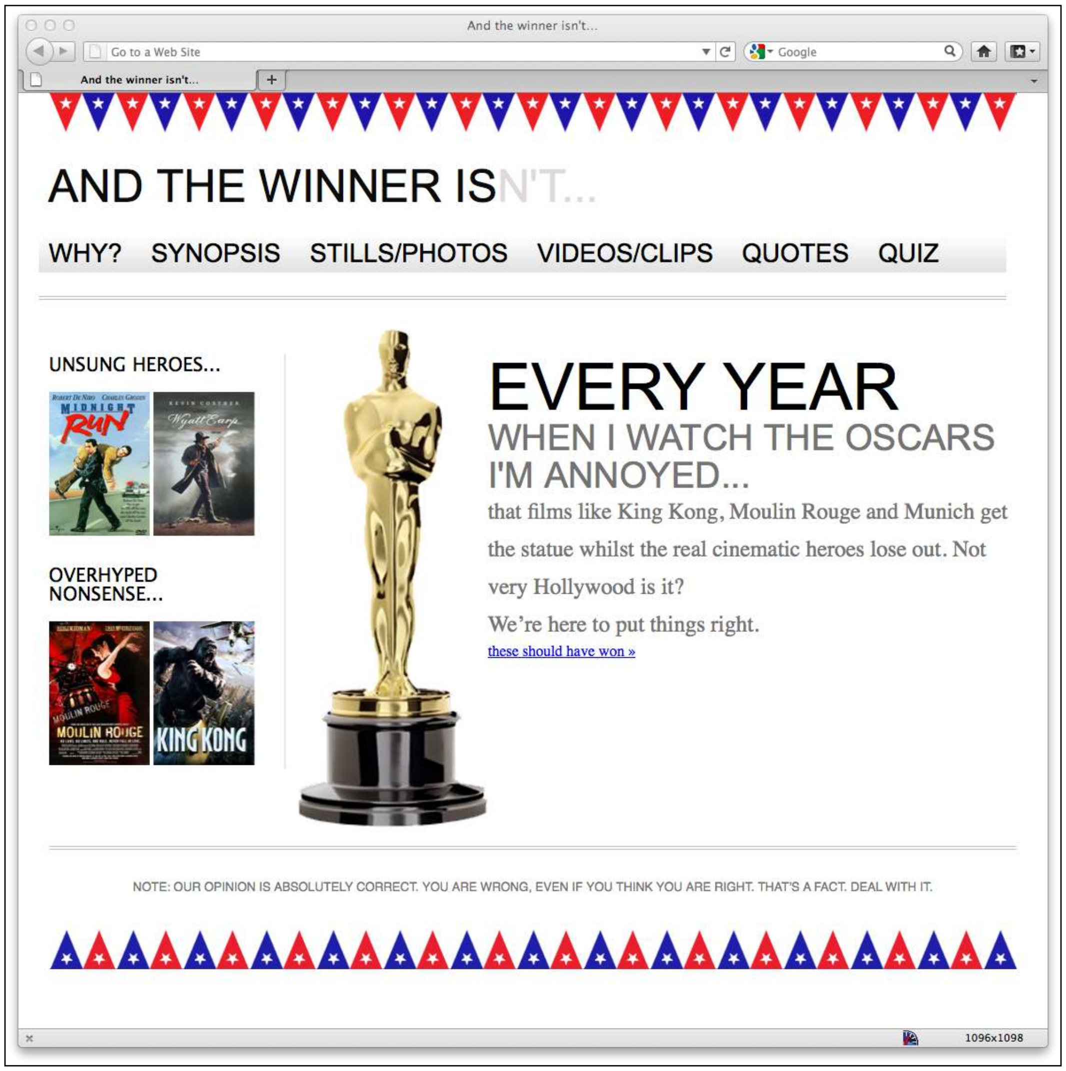
```
aside {
}
```

Even if you've written a huge CSS stylesheet, swapping the references from HTML 4.01 IDs to HTML5 elements is a fairly painless task.

Beware multiple elements in HTML5

Be aware that with HTML5 there may be multiple `<header>`, `<footer>`, and `<aside>` elements within a page so you may need to write more specific styles for individual instances.

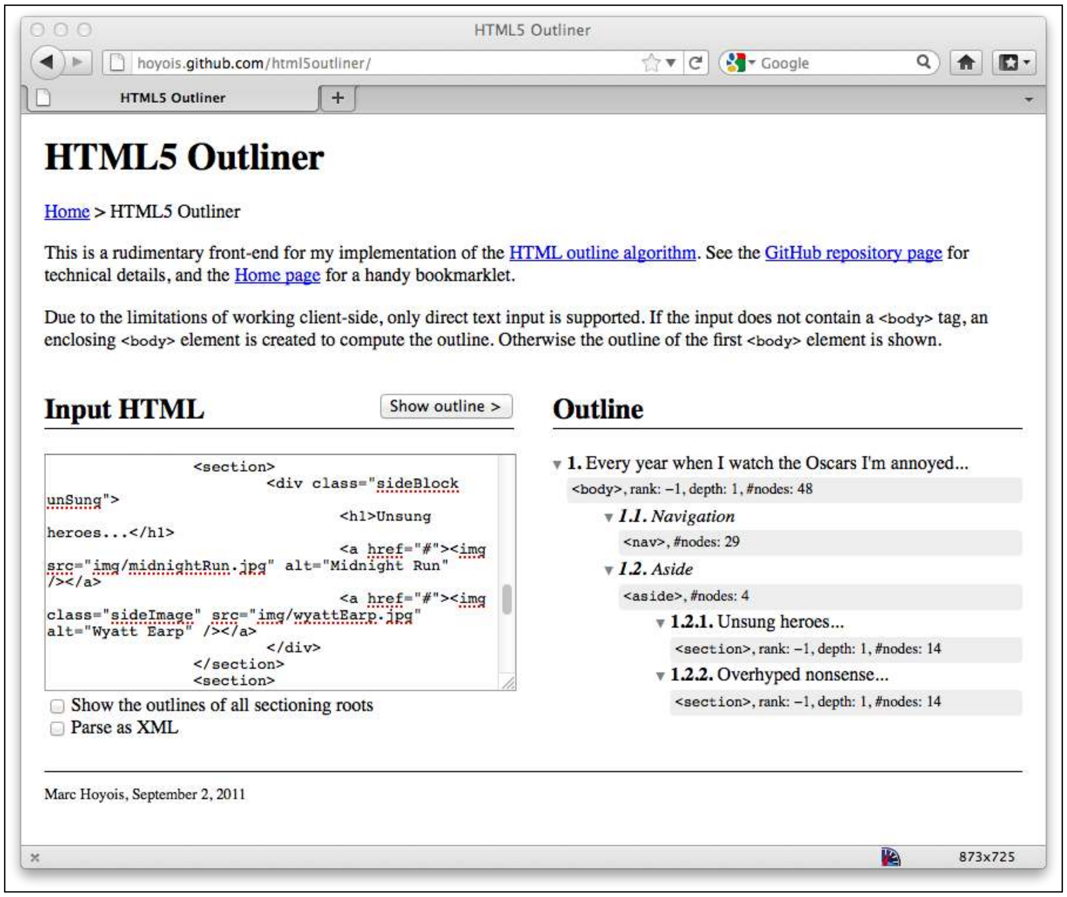
Once the styles for the *And the winner isn't...* have been amended accordingly we're back in business:


And the winner isn't...
Go to a Web Site
Google
AND THE WINNER ISN'T...
WHY? SYNOPSIS STILLS/PHOTOS VIDEOS/CLIPS QUOTES QUIZ
UNSUNG HEROES...
OVERHYPED NONSENSE...
EVERY YEAR
WHEN I WATCH THE OSCARS I'M ANNOYED...
that films like King Kong, Moulin Rouge and Munich get the statue whilst the real cinematic heroes lose out. Not very Hollywood is it?
We're here to put things right.
these should have won »
NOTE: OUR OPINION IS ABSOLUTELY CORRECT. YOU ARE WRONG, EVEN IF YOU THINK YOU ARE RIGHT. THAT'S A FACT. DEAL WITH IT.
1096x1098

Now, although we're telling user agents which section of the page is the aside, within that we have two distinct sections, **UNSUNG HEROES** and **OVERHYPED NONSENSE**. Therefore, in the interest of semantically defining those areas let's amend our code further:

```
<!-- the sidebar -->
  <aside>
    <section>
      <div class="sideBlock unSung">
        <h4>Unsung heroes...</h4>
        <a href="#"><img src="img/midnightRun.jpg" alt="Midnight Run"
/></a>
        <a href="#"><img class="sideImage" src="img/wyattEarp.jpg"
alt="Wyatt Earp" /></a>
      </div>
    </section>
    <section>
      <div class="sideBlock overHyped">
        <h4>Overhyped nonsense...</h4>
        <a href="#"><img src="img/moulinRouge.jpg" alt="Moulin Rouge"
/></a>
        <a href="#"><img src="img/kingKong.jpg" alt="King Kong" /></a>
      </div>
    </section>
  </aside>
```

The important thing to remember is that `<section>` isn't intended for styling purposes, rather to identify a distinct, separate piece of content. Sections should normally have natural headings too, which suits our cause perfectly. Because of the HTML5 outline algorithm, we can also amend our `<h4>` tags to `<h1>` tags and it will still produce an accurate outline of our document:



HTML5 Outliner

hoyois.github.com/html5outliner/

HTML5 Outliner

Home > HTML5 Outliner

This is a rudimentary front-end for my implementation of the HTML outline algorithm. See the GitHub repository page for technical details, and the Home page for a handy bookmarklet.

Due to the limitations of working client-side, only direct text input is supported. If the input does not contain a `<body>` tag, an enclosing `<body>` element is created to compute the outline. Otherwise the outline of the first `<body>` element is shown.

Input HTML

```
                <section>
                        <div class="sideBlock
unSung">
                                <h1>Unsung
heroes...</h1>
                                <a href="#"><img
src="img/midnightRun.jpg" alt="Midnight Run"
/></a>
                                <a href="#"><img
class="sideImage" src="img/wyattEarp.jpg"
alt="Wyatt Earp" /></a>
                        </div>
                </section>
                <section>
```

☐ Show the outlines of all sectioning roots
☐ Parse as XML

Outline

- **1.** Every year when I watch the Oscars I'm annoyed...
  `<body>`, rank: −1, depth: 1, #nodes: 48
  - ***1.1.*** *Navigation*
    `<nav>`, #nodes: 29
  - ***1.2.*** *Aside*
    `<aside>`, #nodes: 4
    - **1.2.1.** Unsung heroes...
      `<section>`, rank: −1, depth: 1, #nodes: 14
    - **1.2.2.** Overhyped nonsense...
      `<section>`, rank: −1, depth: 1, #nodes: 14

Marc Hoyois, September 2, 2011

What about the main content of the site?

It may surprise you that there isn't a distinct element to markup the main content of a page. However, the logic follows that as it's possible to demarcate everything else, what remains should be the main content of the page.

HTML5 text-level semantics

Besides the structural elements we've looked at, HTML5 also revises a few tags that used to be referred to as **inline** elements. The HTML5 specification now refers to these tags as text-level semantics (`http://dev.w3.org/html5/spec/Overview.html#text-level-semantics`). Let's take a look at a few common examples.

The <b> element

Although we may have often used the `<b>` element merely as a styling hook, it actually meant "make this bold". However, you can now officially use it merely as a styling hook in CSS as the HTML5 specification now declares that `<b>` is:

> *…a span of text to which attention is being drawn for utilitarian purposes without conveying any extra importance and with no implication of an alternate voice or mood, such as key words in a document abstract, product names in a review, actionable words in interactive text-driven software, or an article lede.*

The <em> element

OK, hands up, I've often used `<em>` merely as a styling hook, too. I need to mend my ways as in HTML5 it's meant to be used to:

> *…stress emphasis of its contents.*

Therefore, unless you actually want the enclosed contents to be emphasized, consider using a `<b>` tag or, where relevant, an `<i>` tag instead.

The <i> element

The HTML5 specification describes the `<i>` as:

> *…a span of text in an alternate voice or mood, or otherwise offset from the normal prose in a manner indicating a different quality of text.*

Suffice it to say, it's not to be used to merely italicize something.

Applying text-level semantics to our markup

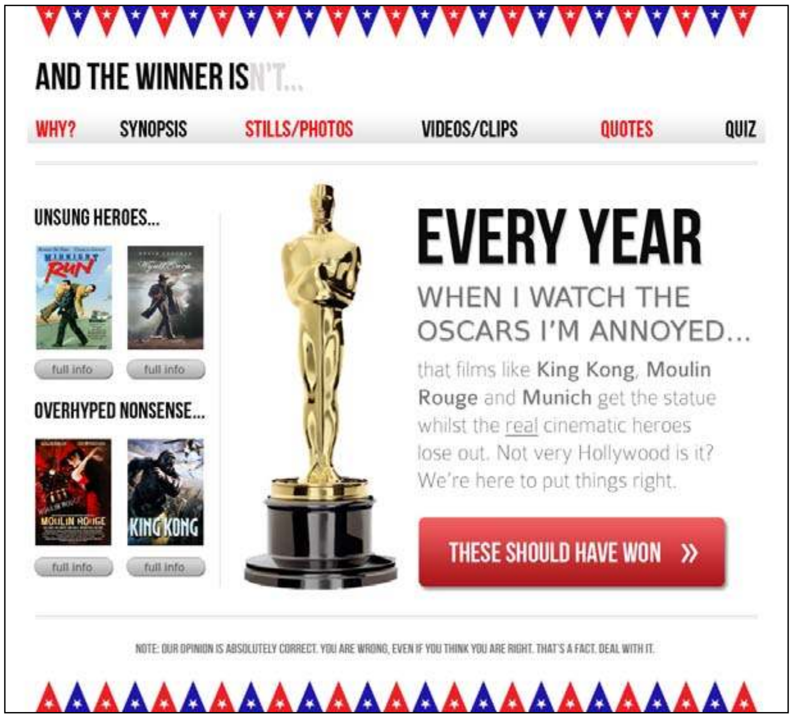
Let's take a look at our current markup for the main content area of our home page and see if we can enhance the meaning to user agents. This is what we have currently:

```
<!-- the content -->
  <div id="content">
    <img class="oscarMain" src="img/oscar.png" alt="atwi_oscar" />
    <h1>Every year <span>when I watch the Oscars I'm annoyed...</
span></h1>
    <p>that films like King Kong, Moulin Rouge and Munich get the
statue whilst the real cinematic heroes lose out. Not very Hollywood
is it?</p>
<p>We're here to put things right. </p>
  <a href="#">these should have won &raquo;</a>
  </div>
```

We can definitely improve things here. To begin with, the `<span>` tag within our headline `<h1>` tag is semantically meaningless in that context so as we're attempting to add emphasis with our style, let's also do it with our code:

```
<h1>Every year <em>when I watch the Oscars I'm annoyed…</em></h1>
```

Let's look at our initial composite again:


AND THE WINNER ISN'T...
WHY?
SYNOPSIS
STILLS/PHOTOS
VIDEOS/CLIPS
QUOTES
QUIZ
UNSUNG HEROES...
full info
full info
OVERHYPED NONSENSE...
full info
full info
EVERY YEAR
WHEN I WATCH THE OSCARS I'M ANNOYED...
that films like King Kong, Moulin Rouge and Munich get the statue whilst the real cinematic heroes lose out. Not very Hollywood is it? We're here to put things right.
THESE SHOULD HAVE WON »
NOTE: OUR OPINION IS ABSOLUTELY CORRECT. YOU ARE WRONG, EVEN IF YOU THINK YOU ARE RIGHT. THAT'S A FACT. DEAL WITH IT.

We also need to style the film names differently, but they don't need to suggest a different mood or voice. Seems like the `<b>` tag is the perfect candidate here:

```
<p>that films like <b>King Kong</b>, <b>Moulin Rouge</b> and
<b>Munich</b> get the statue whilst the real cinematic heroes lose
out. Not very Hollywood is it?</p>
```

Default styling of text-level semantic elements

Because of the historical use of `<b>`, most browsers will still render that as bold so depending upon your situation you may need to restyle the default style in the associated CSS.

Finally, I mean it when I say 'we're here to put things right' – I'm not messing around and I want user agents to know it! So, finally, let's wrap that in a `<i>` tag. You could argue that I should also use the `<em>` tag here instead. That would also be fine in this case but I'm going with `<i>`. So there! This would look like the following:

```
<p><i>We're here to put things right.</i></p>
```

Like `<b>`, browsers will default to italicize the `<i>` tag so where needed, restyle as necessary.

So, we've now added some text-level semantics to our content to give greater meaning to our markup. There are plenty of other text-level semantic tags in HTML5; for the full run down, take a look at the relevant section of the specification at the following URL:

`http://dev.w3.org/html5/spec/Overview.html#text-level-semantics`

However, with a little extra effort we can take things one step further still by providing additional meaning for users of assistive technology.

Adding accessibility to your site with WAI-ARIA

The aim of WAI-ARIA is principally to solve the problem of making dynamic content on a page accessible. It provides a means of describing roles, states, and properties for custom widgets (dynamic sections in web applications) so that they are recognizable and usable by assistive technology users.

For example, if an onscreen widget displays a constantly updating stock price, how would a blind user accessing the page know that? WAI-ARIA attempts to solve this problem. Fully implementing ARIA is outside the scope of this book (for full information, head over to `http://www.w3.org/WAI/intro/aria`). However, there are some very easy to implement parts of ARIA that we can adopt to enhance any site written in HTML5 for users of assistive technologies.

If you're tasked with building a website for a client, there often isn't any time/money set aside for adding accessibility support beyond the basics (sadly, it's often given no thought at all). However, we can use ARIA's **landmark roles** to fix some of the glaring shortfalls in HTML's semantics and allow screen readers that support WAI-ARIA to switch between different screen regions easily.

ARIA's landmark roles

Implementing ARIA's landmark roles isn't specific to a responsive web design. However, as it's relatively simple to add partial support (that also validates as HTML5 with no further effort), there seems little point in leaving it out of any web page you write in HTML5 from this day onwards. Enough talk! Now let's see how it works.

Consider our new HTML5 navigation area:

```
<nav>
  <ul>
    <li><a href="#">Why?</a></li>
    <li><a href="#">Synopsis</a></li>
    <li><a href="#">Stills/Photos</a></li>
    <li><a href="#">Videos/clips</a></li>
    <li><a href="#">Quotes</a></li>
    <li><a href="#">Quiz</a></li>
  </ul>
</nav>
```

We can make this area easy for a WAI-ARIA capable screen reader to jump to by adding a landmark role attribute to it, as shown in the following code snippet:

```
<nav role="navigation">
  <ul>
    <li><a href="#">Why?</a></li>
    <li><a href="#">Synopsis</a></li>
    <li><a href="#">Stills/Photos</a></li>
    <li><a href="#">Videos/clips</a></li>
    <li><a href="#">Quotes</a></li>
    <li><a href="#">Quiz</a></li>
  </ul>
</nav>
```

How easy is that? There are landmark roles for the following parts of a document's structure:

- `application`: This role is used to specify a region used for a web application.
- `banner`: This role is used to specify a sitewide (rather than document specific) area. The header and logo of a site, for example.
- `complementary`: This role is used to specify an area complementary to the main section of a page. In our *And the winner isn't...* site, the **UNSUNG HEROES** and **OVERHYPED NONSENSE** areas would be considered complementary.
- `contentinfo`: This role should be used for information about the main content. For example, to display copyright information at the footer of a page.
- `form`: You guessed it, a form! However, note that if the form in question is a search form, use the `search` role, instead.
- `main`: This role is used to specify the main content of the page.
- `navigation`: This role is used to specify navigation links for the current document or related documents.
- `search`: This role is used to define an area that performs a search.

Taking ARIA further

ARIA isn't limited to landmark roles only. To take things further, a full list of the roles and a succinct description of their usage suitability is available at `http://www.w3.org/TR/wai-aria/roles#role_definitions`

Let's skip ahead and extend our current HTML5 version of the *And the winner isn't...* markup with the relevant ARIA landmark roles:

```
<body>
<div id="wrapper">
  <!-- the header and navigation -->
  <header role="banner">
    <div id="logo">And the winner is<span>n't...</span></div>
    <nav role="navigation">
      <ul>
        <li><a href="#">Why?</a></li>
        <li><a href="#">Synopsis</a></li>
        <li><a href="#">Stills/Photos</a></li>
        <li><a href="#">Videos/clips</a></li>
        <li><a href="#">Quotes</a></li>
```

```
          <li><a href="#">Quiz</a></li>
        </ul>
      </nav>
    </header>
    <!-- the content -->
    <div id="content" role="main">
      <img class="oscarMain" src="img/oscar.png" alt="atwi_oscar" />
      <h1>Every year <em>when I watch the Oscars I'm annoyed…</em></h1>
      <p>that films like <b>King Kong</b>, <b>Moulin Rouge</b> and 
  <b>Munich</b> get the statue whilst the real cinematic heroes lose 
  out. Not very Hollywood is it?</p>
  <p><i>We're here to put things right.</i></p>
    <a href="#">these should have won &raquo;</a>
    </div>
    <!-- the sidebar -->
    <aside>
      <section role="complementary">
        <div class="sideBlock unSung">
          <h1>Unsung heroes...</h1>
          <a href="#"><img src="img/midnightRun.jpg" alt="Midnight Run" 
  /></a>
          <a href="#"><img class="sideImage" src="img/wyattEarp.jpg" 
  alt="Wyatt Earp" /></a>
        </div>
      </section>
      <section role="complementary">
        <div class="sideBlock overHyped">
          <h1>Overhyped nonsense...</h1>
          <a href="#"><img src="img/moulinRouge.jpg" alt="Moulin Rouge" 
  /></a>
          <a href="#"><img src="img/kingKong.jpg" alt="King Kong" /></a>
        </div>
      </section>
    </aside>
    <!-- the footer -->
    <footer role="contentinfo">
      <p>Note: our opinion is absolutely correct. You are wrong, even if 
  you think you are right. That's a fact. Deal with it.</p>
    </footer>

  </div>
  </body>
```

Test your designs for free with NonVisual Desktop Access (NVDA)

If you develop on the Windows platform and you'd like to test your ARIA enhanced designs on a screen reader, you can do so for free with NVDA. You can get it at the following URL:

`http://www.nvda-project.org/`

Hopefully, this brief introduction to WAI-ARIA has demonstrated how easy it is to add partial support for those using assistive technology and you'll consider enhancing your next HTML5 project with it.

Styling ARIA roles

Like any attributes, it's possible to style them directly using the attribute selector. For example, you can add a CSS rule to the `navigation` role using `nav[role="navigation"] {}`.

Embedding media in HTML5

For many, HTML5 first entered their vocabulary when Apple refused to add support for Flash in their iOS devices. Flash had gained market dominance (some would argue market stranglehold) as the plugin of choice to serve up video through a web browser. However, rather than using Adobe's proprietary technology, Apple decided to rely on HTML5 instead to handle rich media rendering. Whilst HTML5 was making good headway in this area anyway, Apple's public support of HTML5 gave it a major leg up and helped its media tools gain greater traction in the wider community.

As you might imagine, Internet Explorer 8 and lower versions don't support HTML5 video and audio. However, there are easy to implement fallback workarounds for Microsoft's ailing browsers, which we'll discuss shortly. Most other modern browsers (Firefox 3.5+, Chrome 4+, Safari 4, Opera 10.5+, Internet Explorer 9+, iOS 3.2+, Opera Mobile 11+, Android 2.3+) handle it just fine.

Adding video and audio the HTML5 way

I'll be honest. I've always found adding media such as video and audio into a web page is an utter pain in HTML 4.01. It's not difficult, just messy. HTML5 makes things far easier. The syntax is much like adding an image:

```
<video src="myVideo.ogg"></video>
```

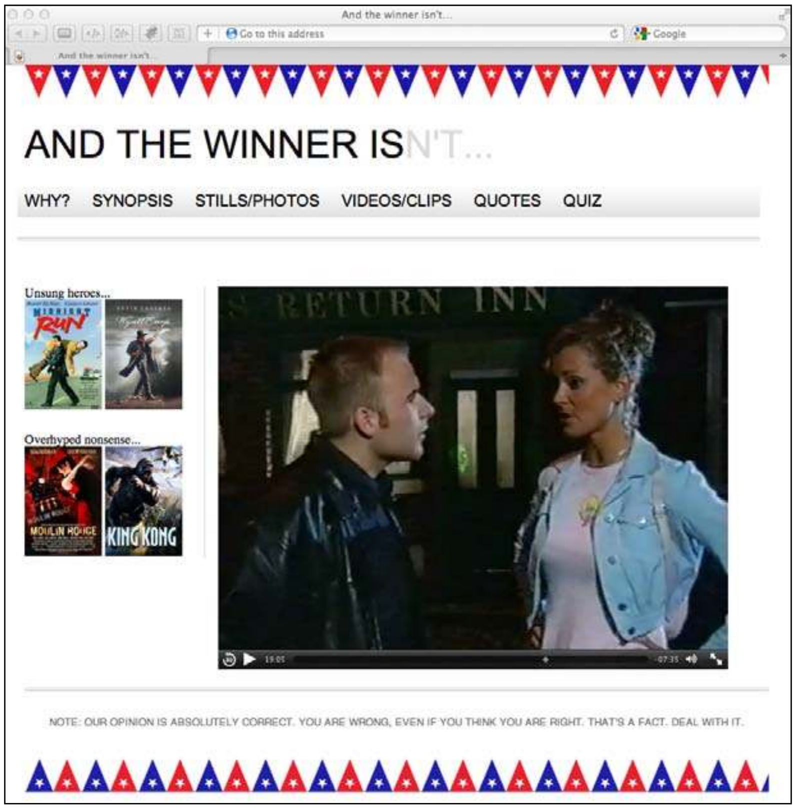
A breath of fresh air for most web designers! Rather than the abundance of code currently needed to include video in a page, HTML5 allows a single `<video></video>`tag (or `<audio></audio>` for audio) to do all the heavy lifting. It's also possible to insert text between the opening and closing tag to inform users when they aren't using an HTML5 compatible browser and there are additional attributes you'd ordinarily want to add, such as the `height` and `width`. Let's add these in:

```
<video src="video/myVideo.mp4" width="640" height="480">What, do you
mean you don't understand HTML5?</video>
```

Now, if we add the preceding code snippet into our page and look at it in Safari, it will appear but there will be no controls for playback. To get the default playback controls we need to add the `controls` attribute. We could also add the `autoplay` attribute (not recommended—it's common knowledge that everyone hates videos that auto-play). This is demonstrated in the following code snippet:

```
<video src="video/myVideo.mp4" width="640" height="480" controls
autoplay>What, do you mean you don't understand HTML5?</video>
```

The result of the preceding code snippet is shown in the following screenshot:



Further attributes include `preload` to control pre-loading of media (early HTML5 adopters should note that `preload` replaces `autobuffer`), `loop` to repeat the video, and `poster` to define a poster frame of video. This is useful if there's likely to be a delay in the video playing. To use an attribute, simply add it to the tag. Here's an example including all these attributes:

```
<video src="video/myVideo.mp4" width="640" height="480" controls
autoplay preload="auto" loop poster="myVideoPoster.jpg">What, do you
mean you don't understand HTML5?</video>
```

Providing alternate source files

The original specification for HTML5 called for all browsers to support the direct playback (without plugins) of video and audio inside Ogg containers. However, due to disputes within the HTML5 working group, the insistence on support for Ogg (including Theora video and Vorbis audio), as a baseline standard, was dropped by more recent iterations of the HTML5 specification. Therefore at present, some browsers support playback of one set of video and audio files whilst others support the other set. For example, Safari only allows MP4/H.264/AAC media to be used with the `<video>` and `<audio>` elements whilst Firefox and Opera only support Ogg and WebM.

Why can't we all just get along! (Mars Attacks)

Thankfully, there is a way to support multiple formats within one tag. It doesn't however preclude us from needing to create multiple versions of our media. Whilst we all keep our fingers crossed this situation resolves itself in due course, in the meantime, armed with multiple versions of our file, we can markup the video as follows:

```
<video width="640" height="480" controls autoplay preload="auto" loop
poster="myVideoPoster.jpg">
    <source src="video/myVideo.ogv" type="video/ogg">
    <source src="video/myVideo.mp4" type="video/mp4">
    What, do you mean you don't understand HTML5?
</video>
```

If the browser supports playback of Ogg, it will use that file; if not, it will continue down to the next `<source>` tag.

Fallback for older browsers

Using the `<source>` tag in this manner, enables us to provide a number of fallbacks, if needed. For example, alongside providing both MP4 and Ogg versions, if we wanted to ensure a suitable fallback for Internet Explorer 8 and lower versions, we could add a Flash fallback. Further still, if the user didn't have any suitable playback technology, we could provide download links to the files themselves:

```
<video width="640" height="480" controls autoplay preload="auto" loop
poster="myVideoPoster.jpg">
    <source src="video/myVideo.mp4" type="video/mp4">
    <source src="video/myVideo.ogv" type="video/ogg">
    <object width="640" height="480" type="application/x-shockwave-
flash" data="myFlashVideo.SWF">
    <param name="movie" value="myFlashVideo.swf" />
    <param name="flashvars" value="controlbar=over&amp;image=myVideoPo
ster.jpg&amp;file=video/myVideo.mp4" />
    <img src="myVideoPoster.jpg" width="640" height="480" alt="__
TITLE__"
         title="No video playback capabilities, please download the
video below" />
    </object>
    <p>  <b>Download Video:</b>
  MP4 Format:  <a href="myVideo.mp4">"MP4"</a>
  Ogg Format:  <a href="myVideo.ogv">"Ogg"</a>
    </p>
</video>
```

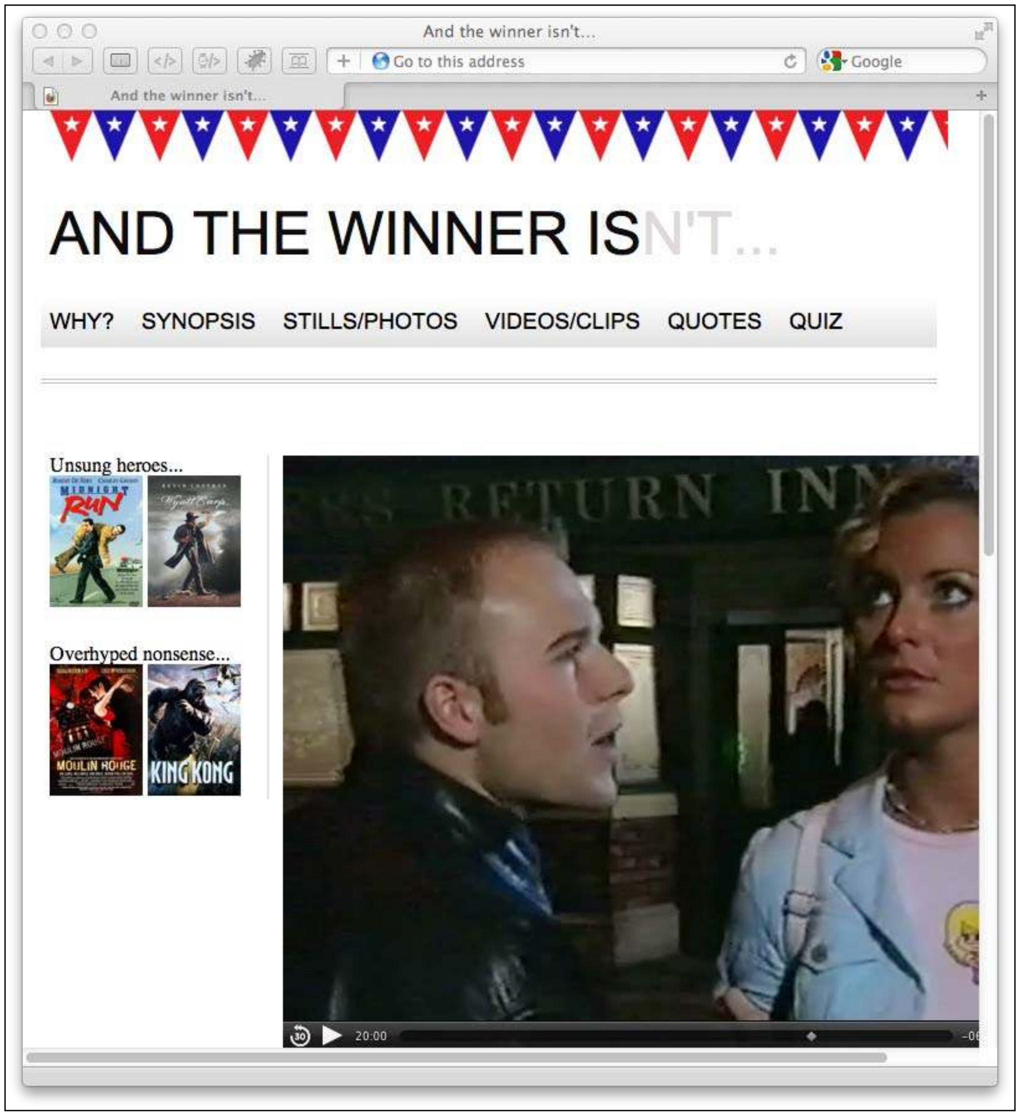
Audio and video tags work almost identically

The `<audio>` tag works on the same principles with the same attributes excluding `width`, `height`, and `poster`. Indeed, you can also use `<video>` and `<audio>` tags almost interchangeably. The main difference between the two being the fact that `<audio>` has no playback area for visible content.

Responsive video

We have seen that, as ever, supporting older browsers leads to code bloat. What began with the `<video>` tag being one or two lines ended up being 10 or more lines (and an extra Flash file) just to make older versions of Internet Explorer happy! For my own part, I'm usually happy to forego the Flash fallback in pursuit of a smaller code footprint but each usage case differs.

Now, the only problem with our lovely HTML5 video implementation is it's not responsive. That's right. All that hard work and our responsive web design doesn't err… respond. Take a look at the following screenshot and do your best to fight back the tears:



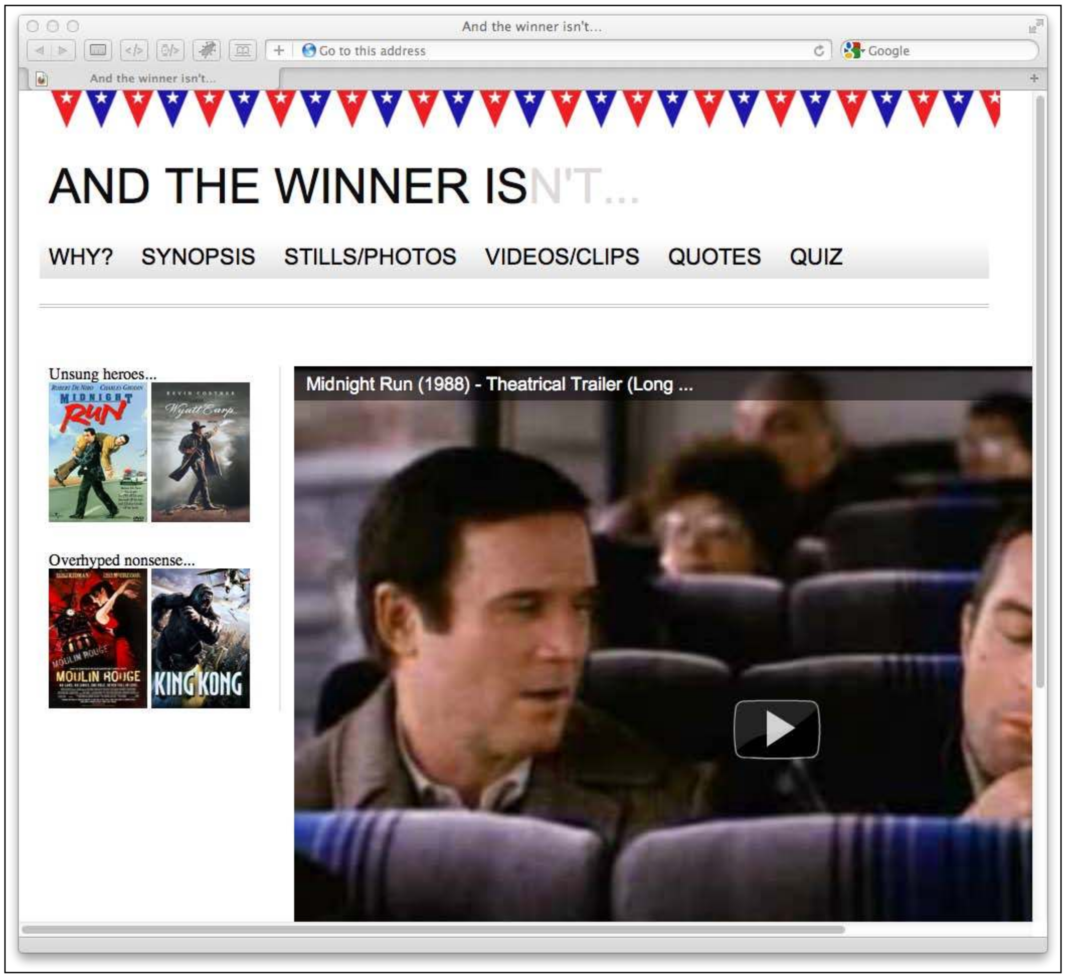
Thankfully, for HTML5 embedded video, the fix is easy. Simply remove any `height` and `width` attributes in the markup (for example, remove `width="640" height="480"`) and add the following in the CSS:

```
video { max-width: 100%; height: auto; }
```

However, whilst that works fine for files that we might be hosting locally, it doesn't solve the problem of videos embedded within an iFrame (take a bow YouTube, Vimeo, et al). The following code adds a film trailer for Midnight Run from YouTube:

```
<iframe width="960" height="720" src="http://www.youtube.com/embed/
B1_N28DA3gY" frameborder="0" allowfullscreen></iframe>
```

Despite my earlier CSS rule, here's what happens:



I'm sure DeNiro wouldn't be too happy about this! There are a number of ways of solving the issue, but by far the easiest I have come across is a small jQuery plugin called **FitVids**. Let's see how easy it is to use the plugin by adding it to the *And the winner isn't...* site.

First of all, we'll need the jQuery JavaScript library. Load this into your `<head>` element. Here, I'm using the version from Google's **Content Delivery Network** (**CDN**).

```
<script src="https://ajax.googleapis.com/ajax/libs/jquery/1.6.4/
jquery.min.js"></script>
```

Download the FitVids plugin from `http://fitvidsjs.com/` (more information on the plugin is at `http://daverupert.com/2011/09/responsive-video-embeds-with-fitvids/`).

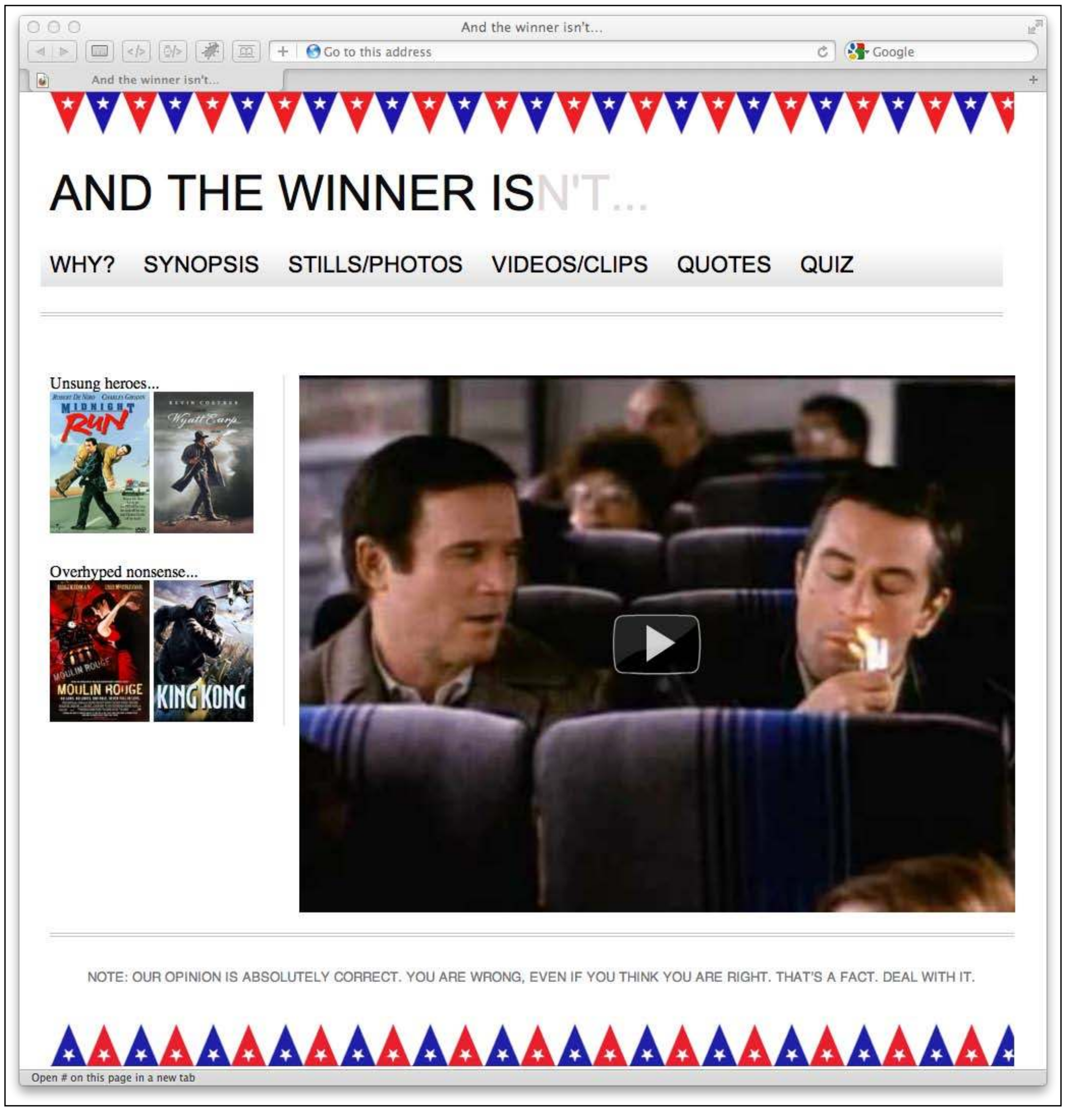
Now, save the FitVids JavaScript file into a suitable folder (I've imaginatively called mine "js") and then link to the FitVids JavaScript in the `<head>` element:

```
<script src="js/fitvids.js"></script>
```

Finally, we just need to use jQuery to target the particular element containing our YouTube video. Here, I've added my Midnight Run YouTube video within the `#content` div:

```
<script>
  $(document).ready(function(){
    // Target your .container, .wrapper, .post, etc.
    $("#content").fitVids();
  });
</script>
```

That's all there is to it. Thanks to the FitVid jQuery plugin, I now have a fully responsive YouTube video. (Note: kids, don't pay any attention to Mr. DeNiro; smoking is bad!)



NOTE: OUR OPINION IS ABSOLUTELY CORRECT. YOU ARE WRONG, EVEN IF YOU THINK YOU ARE RIGHT. THAT'S A FACT. DEAL WITH IT.

Phew, all fixed. That should keep me on Bobby's Christmas card list!

Offline Web applications

Although there are plenty of exciting features within HTML5 that don't explicitly help our responsive quest (the Geolocation API, for example), Offline Web applications potentially could. As we're aware of the growing number of mobile users likely to be accessing our sites, how about we provide a means for them to view our content without even being connected to the Internet? The HTML5 Offline Web applications feature provides this possibility.

Such functionality is of most obvious use to web applications (funnily enough; wonder how they thought up the title). Imagine an online note-taking web application. A user may be halfway through completing a note when their cell phone connection drops. With HTML5 Offline Web applications, they would be able to continue writing the note whilst offline and the data could be sent once a connection is later available.

What's great about the HTML5 Offline Web applications tools is that they are too easy to set up and use. Here, we are going to use them in a basic way—to create an offline version of our site. That means that if users want to look at our site while they don't have a network connection, they can.

Offline Web applications in a nut shell

Offline Web applications work by each page that needs to be used offline, pointing to a text file known as a `.manifest` file. This file lists all the resources (HTML, images, JavaScript, and so on) that are needed by the page should it be offline. An Offline Web application enabled browser (Firefox 3+, Chrome 4+, Safari 4+, Opera 10.6+, iOS 3.2+, Opera Mobile 11+, Android 2.1+, Internet Explorer 10+) reads the `.manifest` file, downloads the resources listed, and caches them locally should the connection be dropped. Simple, eh? Let's do this…

Making web pages work offline

In the opening HTML tag, we point to a `.manifest` file:

```
<html lang="en" manifest="/offline.manifest">
```

You can call this file anything you want but it is recommended that the file extension used is `.manifest`.

[You must add the `manifest="/offline.manifest"` attribute to the HTML tag of every page you want to be available offline.]

If your web server runs on Apache, you'll probably need to amend the `.htaccess` file with the following line:

```
AddType text/cache-manifest .manifest
```

This will allow the file to have the correct MIME type, which is `text/cache-manifest`.

While we're in the `.htaccess` file, also add the following:

```
<Files offline.manifest>
  ExpiresActive On
  ExpiresDefault "access"
</Files>
```

Adding the preceding lines of code, stops the browser from caching the cache. Yes, you read that right. As the `offline.manifest` file is a static file, by default the browser will cache the `offline.manifest` file. So, this tells the server to tell the browser not to!

Now we need to write the `offline.manifest` file. This will instruct the browser about which files to make available offline. Here's the content of the `offline.manifest` file for the *And the winner isn't...* site:

```
CACHE MANIFEST
#v1

CACHE:
basic_page_layout_ch4.html
css/main.css
img/atwiNavBg.png
img/kingHong.jpg
img/midnightRun.jpg
img/moulinRouge.jpg
img/oscar.png
img/wyattEarp.jpg
img/buntingSlice3Invert.png
img/buntingSlice3.png

NETWORK:
*

FALLBACK:
/ /offline.html
```

Understanding the manifest file

The manifest file must begin with `CACHE MANIFEST`. The next line is merely a comment, stating the version number of the manifest file. More on that shortly.

The `CACHE:` section lists the files that we need for offline use. These should be relative to the `offline.manifest` file, so paths may need to be changed depending upon the resources that need caching. It's also possible to use absolute URLs if needed.

The `NETWORK:` section lists any resources that should not be cached. Think of it as an "online whitelist". Whatever is listed here will always by-pass the cache if a network connection is available. If you want to make your site content available where a network is available (rather than only looking in the offline cache), the `*` character allows it. It's known as the **online whitelist wildcard flag**.

The `FALLBACK:` section uses the `/` character to define a URL pattern. It basically asks "is this page in the cache?" If it finds the page there, great, it displays it. If not, it shows the user the file specified—`offline.html`.

Automatic loading of pages to the offline manifest

Depending on the circumstances, there's an even easier way of setting an `offline.manifest` file up. Any page that points to an offline manifest file (remember that we do this by adding `manifest="/offline.manifest"` in our opening `<html>` tag) gets automatically added to the cache when a user visits it. This technique will add every page on your site that a user visits to their cache so they can view it again offline. Here's what the manifest should look like:

```
CACHE MANIFEST
# Cache Manifest v1
FALLBACK:
/ /offline.html
NETWORK:
*
```

One point of note when opting for this technique is that just the HTML of the page that is visited will be downloaded and cached. Not the images/JavaScript and other resources it may contain and link to. If these are essential, specify them in a `CACHE:` section as already described earlier in the *Understanding the manifest file* section.

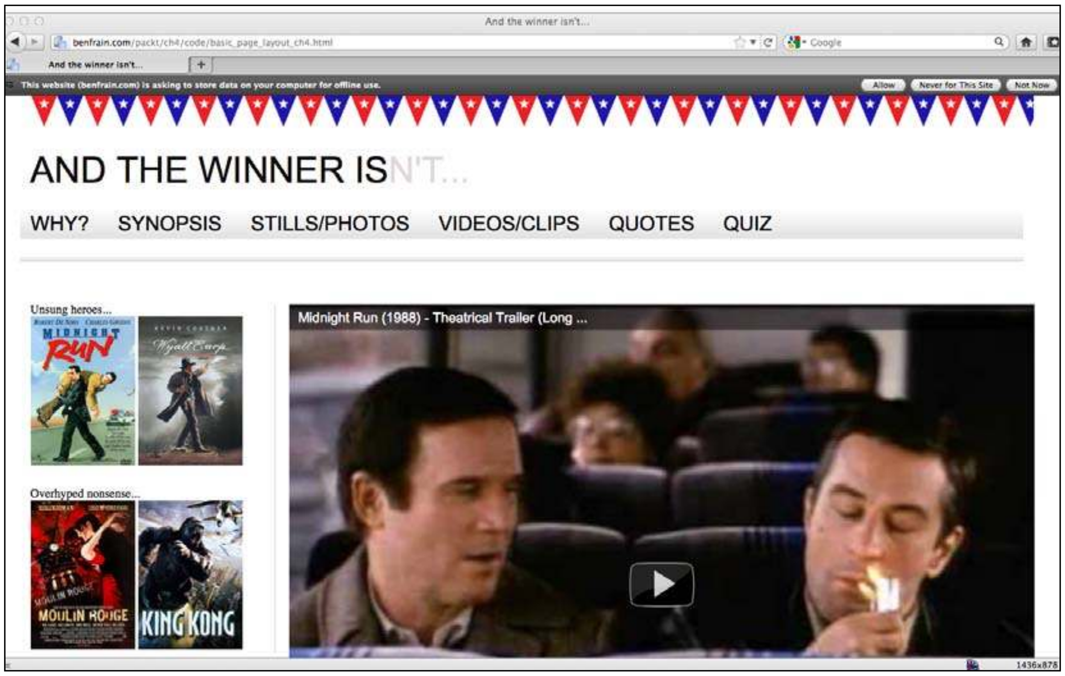
About that version comment

When you make changes to your site or any of its resources, you must change the `offline.manifest` file somehow and re-upload it. This will enable the server to provide the new file to the browser, which will then get the new versions of the files and kick off the offline process again. I follow Nick Pilgrim's example (from the excellent *Dive into HTML5*) and add a comment to the top of the `offline.manifest` file that I increment with each change:

```
# Cache Manifest v1
```

Viewing the site offline

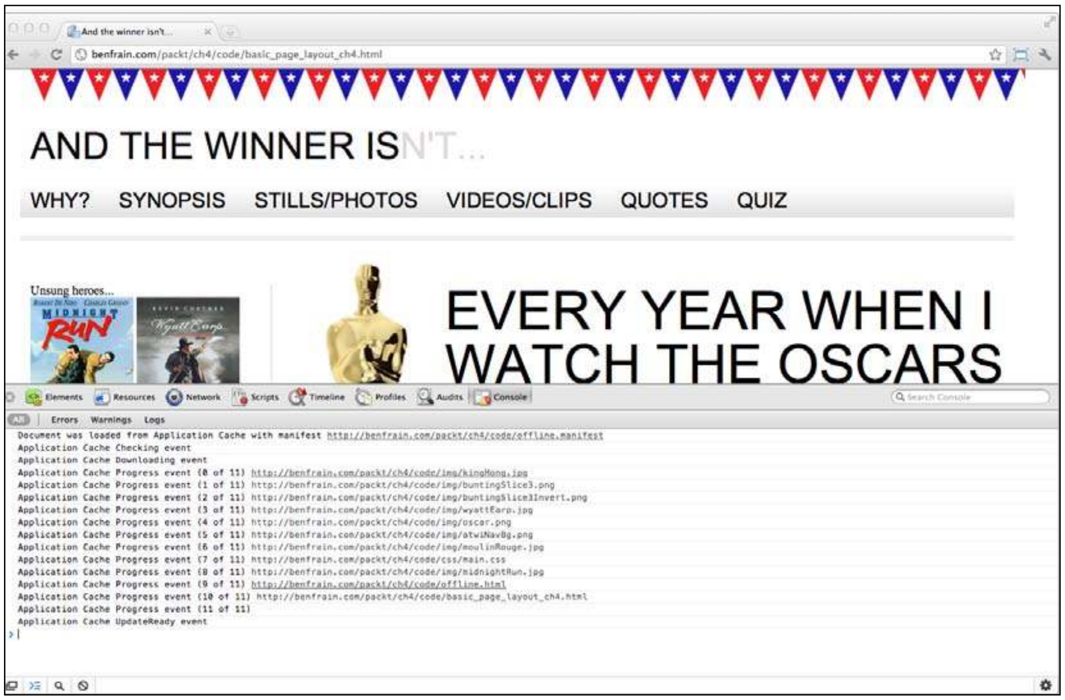
Now, it's time to test our handiwork. Visit the page in an Offline Web application capable browser. Some browsers will warn about offline mode (Firefox for example—note the top bar) whilst Chrome makes no mention of it:


And the winner isn't...
benfrain.com/packt/ch4/code/basic_page_layout_ch4.html
This website (benfrain.com) is asking to store data on your computer for offline use.
Allow
Never for This Site
Not Now
AND THE WINNER ISN'T...
WHY?
SYNOPSIS
STILLS/PHOTOS
VIDEOS/CLIPS
QUOTES
QUIZ
Unsung heroes...
Overhyped nonsense...
Midnight Run (1988) - Theatrical Trailer (Long ...

Now, pull the plug (or you know, switch off WiFi—that just didn't sound as dramatic as "pull the plug") and refresh the browser. Hopefully, the page will refresh as if connected – only it isn't.

Troubleshooting Offline Web applications

When I have problems getting sites to work correctly in Offline mode I tend to use Chrome to troubleshoot. The built-in Developer tools have a handy Console section (access it by clicking the spanner logo to the right of the address bar and then go to **Tools** | **Developer tools** and click the **Console** tab) that flags up success or failure of the offline cache and often points out what you're doing wrong. In my experience, it's usually path issues; for example, not pointing my pages to the correct location of the manifest file.



```
Document was loaded from Application Cache with manifest http://benfrain.com/packt/ch4/code/offline.manifest
Application Cache Checking event
Application Cache Downloading event
Application Cache Progress event (0 of 11) http://benfrain.com/packt/ch4/code/img/kingHong.jpg
Application Cache Progress event (1 of 11) http://benfrain.com/packt/ch4/code/img/buntingSlice3.png
Application Cache Progress event (2 of 11) http://benfrain.com/packt/ch4/code/img/buntingSlice3Invert.png
Application Cache Progress event (3 of 11) http://benfrain.com/packt/ch4/code/img/wyattEarp.jpg
Application Cache Progress event (4 of 11) http://benfrain.com/packt/ch4/code/img/oscar.png
Application Cache Progress event (5 of 11) http://benfrain.com/packt/ch4/code/img/atwiNavBg.png
Application Cache Progress event (6 of 11) http://benfrain.com/packt/ch4/code/img/moulinRouge.jpg
Application Cache Progress event (7 of 11) http://benfrain.com/packt/ch4/code/css/main.css
Application Cache Progress event (8 of 11) http://benfrain.com/packt/ch4/code/img/midnightRun.jpg
Application Cache Progress event (9 of 11) http://benfrain.com/packt/ch4/code/offline.html
Application Cache Progress event (10 of 11) http://benfrain.com/packt/ch4/code/basic_page_layout_ch4.html
Application Cache Progress event (11 of 11)
Application Cache UpdateReady event
> |
```

For the full specification of the Offline Web applications, head over to the following URL:

`http://dev.w3.org/html5/spec/Overview.html#offline`

Summary

We've covered a lot in this chapter. Everything from the basics of creating a page that validates as HTML5, to enabling our pages to work offline when users are lacking an Internet connection. We've also tackled embedding rich media (video) into our markup, and ensured it behaves responsively for differing viewports. Although not specific to responsive designs, we've also covered how we can write semantically rich and meaningful code and also provide help to users that rely on assistive technologies. However, our site is still facing some major shortfalls. Without putting too fine a point on it—it looks pretty shabby. Our text is un-styled and we're completely lacking details such as the buttons visible in the original composite. We've avoided loading the markup with images to solve these issues thus far with good reason. We don't need them! Instead, in the next few chapters we're going to embrace the power and flexibility of CSS3 to create a faster and more maintainable responsive design.