Multimedia and Embedding

Images and the <img> element

Let's start with the humble `<img>` element, used to embed a simple image in a webpage. In order to put a simple image on a webpage, we use the `<img>` element. This is an empty element (meaning that it has no text content or closing tag) that requires a minimum of one attribute to be useful — `src`. The `src` attribute contains a path pointing to the image you want to embed in the page, which can be a relative or absolute URL, in the same way as `href` attribute values in `<a>` elements.

So for example, if your image is called dinosaur.jpg, and it sits in the same directory as your HTML page, you could embed the image like so:

```
<img src="dinosaur.jpg">
```

If the image was in an "images" subdirectory, which was inside the same directory as the HTML page, then you'd embed it like this:

```
<img src="images/dinosaur.jpg">
```

Note: Search engines also read image filenames and count them towards SEO. Therefore, you should give your image a descriptive filename; dinosaur.jpg is better than img835.png.

You could also embed the image using its absolute URL, for example:

```
<img src="https://www.example.com/images/dinosaur.jpg">
```

But this is pointless, as it just makes the browser do more work, looking up the IP address from the DNS server all over again, etc. You'll almost always keep the images for your website on the same server as your HTML.

Note: Elements like `<img>` and `<video>` are sometimes referred to as **replaced elements**. This is because the element's content and size are defined by an external resource (like an image or video file), not by the contents of the element itself.

Alternative text

The next attribute we'll look at is `alt`. Its value is supposed to be a textual description of the image, for use in situations where the image cannot be seen/displayed or takes a long time to render because of a slow internet connection. For example,

```
<img src="images/dinosaur.jpg"alt="The head and torso of a dinosaur skeleton;
          it has a large head with long sharp teeth">
```

The easiest way to test your alt text is to purposely misspell your filename.

The `alt` can come in handy for a number of reasons:

- The user is visually impaired, and is using a screen reader to read the web out to them. In fact, having alt text available to describe images is useful to most users.

- As described above, the spelling of the file or path name might be wrong.
- The browser doesn't support the image type. Some people still use text-only browsers, such as Lynx, which displays the alt text of images.
- You may want to provide text for search engines to utilize; for example, search engines can match alt text with search queries.
- Users have turned off images to reduce data transfer volume and distractions. This is especially common on mobile phones, and in countries where bandwidth is limited or expensive.

Width and height

You can use the `width` and `height` attributes to specify the width and height of your image.

```
<img src="images/dinosaur.jpg"
     alt="The head and torso of a dinosaur skeleton;
          it has a large head with long sharp teeth"
      width="400"
      height="341">
```

This doesn't result in much difference to the display, under normal circumstances. But if the user has just navigated to the page, and the image hasn't yet loaded, you'll notice the browser is leaving a space for the image to appear in.

However, you shouldn't alter the size of your images using HTML attributes. If you set the image size too big, you'll end up with images that look grainy, fuzzy, or too small, and wasting bandwidth downloading an image that is not fitting the user's needs. The image may also end up looking distorted, if you don't maintain the correct aspect ratio. You should use an image editor to put your image at the correct size before putting it on your webpage. Or else you should use CSS to alter an image's size.

Image titles

As with links, you can also add title attributes to images, to provide further supporting information if needed.

```
<img src="images/dinosaur.jpg"
     alt="The head and torso of a dinosaur skeleton;
          it has a large head with long sharp teeth"
     width="400"
     height="341"
     title="A T-Rex on display in the Manchester University Museum">
```

This gives us a tooltip on mouse hover, just like link titles:

Annotating images with figures and figure captions

The HTML5 `<figure>` and `<figcaption>` elements can be used to semantically link the image to its caption, which will not cause problems to screen readers.

For example, when you have 50 images and captions, the question of which caption goes with which image can be addressed by using `<figure>` and `<figcaption>`. The `<figcaption>` element tells browsers, and assistive technology that the caption describes the other content of the `<figure>` element.

```
<figure>
  <img src="images/dinosaur.jpg"
       alt="The head and torso of a dinosaur skeleton;
```

```
            it has a large head with long sharp teeth"
       width="400"
       height="341">

  <figcaption>A T-Rex on display in the Manchester University Museum.</figcaption>
</figure>
```

Note: From an accessibility viewpoint, captions and alt text have distinct roles. Captions benefit even people who can see the image, whereas alt text provides the same functionality as an absent image. Therefore, captions and alt text shouldn't just say the same thing, because they both appear when the image is gone. Try turning images off in your browser and see how it looks.

A figure doesn't have to be an image. It is an independent unit of content that:

- Expresses your meaning in a compact, easy-to-grasp way.
- Could go in several places in the page's linear flow.
- Provides essential information supporting the main text.

A figure could be several images, a code snippet, audio, video, equations, a table, or something else.

CSS background images

You can also use CSS to embed images into webpages (and JavaScript, but that's another story entirely). The CSS `background-image` property, and the other `background-*` properties, are used to control background image placement. For example, to place a background image on every paragraph on a page, you could do this:

```
p {
  background-image: url("images/dinosaur.jpg");
}
```

The resulting embedded image is arguably easier to position and control than HTML images. So why bother with HTML images? As hinted to above, CSS background images are for decoration only. If you just want to add something pretty to your page to enhance the visuals, this is fine. Though, such images have no semantic meaning at all. They can't have any text equivalents, are invisible to screen readers, and so on. This is where HTML images shine!

If an image has meaning, in terms of your content, you should use an HTML image. If an image is purely decoration, you should use CSS background images.

Video and Audio on the web

In the early days, native web technologies such as HTML didn't have the ability to embed video and audio on the Web, so proprietary (or plugin-based) technologies like Flash — and later, Silverlight (both of which are now obsolete) — became popular for handling such content. This kind of technology worked ok, but it had a number of problems, including not working well with HTML/CSS features, security issues, and accessibility issues.

The HTML5 specification provided native solution and added the `<video>` and `<audio>` elements would solve much of above problems if implemented correctly.

Note: There are quite a few OVPs (online video providers) like YouTube, Dailymotion, and Vimeo, and online audio providers like Soundcloud. Such companies offer a convenient, easy way to host and consume videos, so you don't have to worry about the enormous bandwidth consumption. OVPs even usually offer ready-made code for embedding video/audio in your webpages; if you use that route, you can avoid some of the difficulties we discuss further.

The <video> element

The `<video>` element allows you to embed a video very easily. A really simple example looks like this:

```
<video src="rabbit320.webm" controls>
  <p>Your browser doesn't support HTML5 video. Here is a <a href="rabbit320.webm">link  to the video</a> instea
d.</p>
</video>
```

The features of note are:

- `src`

  In the same way as for the `<img>` element, the `src` (source) attribute contains a path to the video you want to embed. It works in exactly the same way.

- `controls`

  For the users to be able to control video and audio playback, you must either use the controls attribute to include the browser's own control interface, or build your interface using the appropriate JavaScript API. At a minimum, the interface must include a way to start and stop the media, and to adjust the volume.

- **The paragraph inside the <video> tags**

  This is called **fallback content** — this will be displayed if the browser accessing the page doesn't support the `<video>` element, allowing us to provide a fallback for older browsers. This can be anything you like; in this case, we've provided a direct link to the video file, so the user can at least access it some way regardless of what browser they are using.

Using multiple source formats to improve compatibility

Different browsers support different video (and audio) formats. Things become slightly more complicated because not only does each browser support a different set of container file formats, they also each support a different selection of codecs. Fortunately, there are things you can do to help prevent this from being a problem. In order to maximize the likelihood that your web site or app will work on a user's browser, you may need to provide each media file you use in multiple formats. If your site and the user's browser don't share a media format in common, your media won't play.

Also, mobile browsers may support additional formats not supported by their desktop equivalents, just like they may not support all the same formats the desktop version does. On top of that, both desktop and mobile browsers may be designed to offload handling of media playback (either for all media or only for specific types it can't handle internally). This means media support is partly dependent on what software the user has installed.

So how do we do this?

```
<video controls>
  <source src="rabbit320.mp4" type="video/mp4">
  <source src="rabbit320.webm" type="video/webm">
```

```
  <p>Your browser doesn't support HTML5 video. Here is a <a href="rabbit320.mp4">link to the video</a> instead.</p>
</video>
```

Here we've taken the `src` attribute out of the actual `<video>` tag, and instead included separate `<source>` elements that point to their own sources. In this case the browser will go through the `<source>` elements and play the first one that it has the codec to support. Including WebM and MP4 sources should be enough to play your video on most platforms and browsers these days.

Each `<source>` element also has a `type` attribute. This is optional, but it is advised that you include it. The `type` attribute contains the media type of the file specified by the `<source>`, and browsers can use the type to immediately skip videos they don't understand. If type isn't included, browsers will load and try to play each file until they find one that works, which obviously takes time and is an unnecessary use of resources.

Other <video> features

There are a number of other features you can include when displaying an HTML video as shown:

```
<video controls width="400" height="400"
       autoplay loop muted preload="auto"
       poster="poster.png">
  <source src="rabbit320.mp4" type="video/mp4">
  <source src="rabbit320.webm" type="video/webm">
  <p>Your browser doesn't support HTML video. Here is a <a href="rabbit320.mp4">link to the video</a> instead.</p>
</video>
```

`width` and `height`

You can control the video size either with these attributes or with CSS. In both cases, videos maintain their native width-height ratio — known as the **aspect ratio**. If the aspect ratio is not maintained by the sizes you set, the video will grow to fill the space horizontally, and the unfilled space will just be given a solid background color by default.

`autoplay`

Makes the audio or video start playing right away, while the rest of the page is loading. You are advised not to use autoplaying video (or audio) on your sites, because users can find it really annoying.

`loop`

Makes the video (or audio) start playing again whenever it finishes. This can also be annoying, so only use if really necessary.

`muted`

Causes the media to play with the sound turned off by default.

`poster`

The URL of an image which will be displayed before the video is played. It is intended to be used for a splash screen or advertising screen.

`preload`

Used for buffering large files; it can take one of three values:

- `"none"` does not buffer the file
- `"auto"` buffers the media file
- `"metadata"` buffers only the metadata for the file

The <audio> element

he `<audio>` element works just like the `<video>` element, with a few small differences as outlined below. A typical example might look like so:

```
<audio controls>
  <source src="viper.mp3" type="audio/mp3">
  <source src="viper.ogg" type="audio/ogg">
  <p>Your browser doesn't support HTML5 audio. Here is a <a href="viper.mp3">link to the audio</a> instead.</p>
</audio>
```

Other differences from HTML video are as follows:

- The `<audio>` element doesn't support the `width`/`height` attributes — again, there is no visual component, so there is nothing to assign a width or height to.
- It also doesn't support the `poster` attribute — again, no visual component.

Other than this, <audio> supports all the same features as <video> — review the above sections for more information about them.

Displaying video text tracks

HTML can provide a transcript of the words being spoken in the audio/video.

To do so we use the WebVTT file format and the `<track>` element.

WebVTT is a format for writing text files containing multiple strings of text along with metadata such as the time in the video at which each text string should be displayed, and even limited styling/positioning information. These text strings are called **cues**, and there are several kinds of cues which are used for different purposes. The most common cues are:

- **subtitles:** Translations of foreign material, for people who don't understand the words spoken in the audio.
- **captions:** Synchronized transcriptions of dialog or descriptions of significant sounds, to let people who can't hear the audio understand what is going on.
- **timed descriptions:** Text which should be spoken by the media player in order to describe important visuals to blind or otherwise visually impaired users.

A typical WebVTT file will look something like this:

```
WEBVTT

1
00:00:22.230 --> 00:00:24.606
This is the first subtitle.

2
00:00:30.739 --> 00:00:34.074
This is the second.

  ...
```

To get this displayed along with the HTML media playback, you need to:

1. Save it as a `.vtt` file in a sensible place.

2. Link to the `.vtt` file with the `<track>` element. `<track>` should be placed within `<audio>` or `<video>`, but after all `<source>` elements. Use the `kind` attribute to specify whether the cues are `subtitles`, `captions`, or `descriptions`. Further, use `srclang` to tell the browser what language you have written the subtitles in. Finally, add `label` to help readers identify the language they are searching for.

Here's an example:

```
<video controls>
    <source src="example.mp4" type="video/mp4">
    <source src="example.webm" type="video/webm">
    <track kind="subtitles" src="subtitles_es.vtt" srclang="es" label="Spanish">
</video>
```