



HTML5, CSS3, and JavaScript 6th Edition

Tutorial 8

Enhancing a Website with Multimedia

Objectives

- Understand audio and video formats
- Insert an HTML audio clip
- Support multiple audio formats
- Insert an HTML video clip
- Write a video caption track

Objectives (continued)

- Format video captions
- Create a CSS transition
- Explore transition attributes
- Create a CSS key frame animation
- Apply a CSS animation

Introducing Multimedia on the Web

- HTML is the perfect tool to share text and data
- The next major phase in HTML language was the introduction of multimedia support
- Streaming audio, video, and interactive games, made the web a dominant entertainment platform
- One of the biggest challenges
 - Delivering multimedia content in a form that can be retrieved quickly and easily without loss of quality

Understanding Codecs and Containers

- **Codec:** Computer program that encodes and decodes streams of data
- Codecs compress data to transmit it in fast and efficient manner
- Codecs decompress data when it is to be read or played back

Understanding Codecs and Containers (continued 1)

- The compression method can be either lossy or lossless
- **Lossy compression:** Nonessential data is removed in order to achieve a smaller file size
- Example:
 - An audio file might be compressed by removing sounds that the human ear can barely hear

Understanding Codecs and Containers (continued 2)

- The more the file is compressed, the more the content is lost
- Data removed during compression cannot be recovered
- **Lossless compression:** Data is compressed by removing redundant information
- Example:
 - AAAABBBBBCCCCC requires 15 characters of information, which can be rewritten using 6 characters as 4A5B6C

Understanding Codecs and Containers (continued 3)

- Lossless compression cannot achieve the same level of compression as with lossy compression
- Codecs are placed within a **container** that handles the packaging, transportation and presentation of data
- Container is the file format identified by a file extension

Understanding Codecs and Containers (continued 4)

- The web supports a multitude of container and codec combinations
- Not all containers and codecs are equally supported
- For example, the combination of WebM container and VP8 codec is supported by Google Chrome but not Internet Explorer or any Apple device

Understanding Plug-Ins

- **Media player:** Decodes and plays multimedia content stored within a container file
- **Plug-in:** Software program accessed by a browser to provide a feature or capability not native to the browser
- A plug-in either opens in its own external window or runs within the web page as an **embedded object**

Understanding Plug-Ins (continued 1)

- Problems with the plug-in approach for delivery of multimedia content
 - Plugs-ins require users to install a separate application in addition to their web browsers
 - A common plug-in is not available across all browsers, operating systems, and devices
 - HTML documents that support multiple plug-ins are difficult to create and maintain

Understanding Plug-Ins (continued 2)

- Plug-ins consume valuable system resources, resulting in slow and unreliable performance
- Plug-ins are a security risk with some of the most prominent Internet attacks working their way into browsers via a plug-in

Working with the `audio` Element

- Audio clips are embedded within a web page using `audio` element

```
<audio src="url" attributes />
```

where *url* specifies the source of the audio file and *attributes* define how the audio clip should be handled by the browser

Working with the `audio` Element (continued)

Figure 8-2 Attributes of HTML audio and video elements

Attribute	Description
<code>autoplay</code>	Starts playing the media clip as soon as it is loaded by the browser
<code>controls</code>	Displays the player controls in the web page
<code>loop</code>	Automatically restarts the media clip when it is finished playing
<code>muted</code>	Specifies that the audio output should be muted
<code>preload="auto metadata none"</code>	Specifies whether the media clip should be preloaded by the browser, where type is <code>auto</code> (to load the entire clip), <code>metadata</code> (to preload only descriptive data about the clip), or <code>none</code> (not to preload the media clip)
<code>src="url"</code>	Specifies the source of the media clip, where <code>url</code> is the location and name of the media file

Browsers and Audio Formats

- HTML does not specify any particular audio format

Figure 8-3 Audio formats in HTML

Format	Description	Codec	File Extension(s)	MIME Type
MP3	MPEG-1 Audio Layer 3 or MP3 is one of the most widely used audio types and is the standard format for digital audio players	MP3	.mp3	audio/mpeg
AAC	Advanced Audio Coding or AAC is the encoding standard for all Apple products, as well as YouTube and several gaming systems and mobile devices; AAC was introduced as the successor to MP3 with the goal of achieving better sound quality at similar compression ratios	AAC	.aac .mp4 .m4a	audio/mp4
OGG	A file compression format designed for web audio, Ogg is an open source and royalty-free format; in general, Ogg provides better sound quality than MP3, especially at lower bit rates	Vorbis	.ogg	audio/ogg
WAV	The original audio format for Windows PCs, WAV is commonly used for storing uncompressed audio, making it impractical for all but the shortest audio clips	PCM	.wav	audio/wav

Browsers and Audio Formats (continued 1)

Figure 8-4 Browser support for audio formats

Browser	MP3	AAC	Ogg	WAV
Chrome (desktop)	✓	✓	✓	✓
Chrome (mobile)	✓	✓	✓	✓
Firefox (desktop)	✓	✓	✓	✓
Firefox (mobile)	✓		✓	✓
Microsoft Edge (desktop)	✓	✓		
Internet Explorer (desktop)	✓	✓		
Internet Explorer (mobile)	✓	✓		
Opera (desktop)			✓	✓
Opera (mobile)			✓	✓
Safari (desktop)	✓	✓		✓
Safari (mobile)	✓	✓		✓

Browsers and Audio Formats (continued 2)

- Nest several `source` elements within a single `audio` element to provide several versions of the same media file

```
<audio>
```

```
<source src="url1" type="mime-type" />
```

```
<source src="url2" type="mime-type" />
```

```
... </audio>
```

where, *url1*, *url2*,... are the URLs for each audio file and *mime-type* specifies the audio format associated with each file

Browsers and Audio Formats (continued 3)

Figure 8-5

Inserting an audio clip

```
<p>Lane's greatest musical accomplishment may  
very well be his discovery of an 11-year-old singing sensation named  
Frances Gumm, whom the world now knows better as Judy Garland.  
</p>  
<p>Click the play button below to hear the musical overture  
for Burton Lane's <cite>Royal Wedding</cite>.</p>  
<audio controls>  
  <source src="cp_overture.mp3" type="audio/mp3" />  
  <source src="cp_overture.ogg" type="audio/ogg" />  
</audio>  
</aside>
```

displays the controls
for the audio player

two possible
sources for
the audio file

Applying Styles to the Media Player

- The appearance of a media player is determined by the browser itself
- CSS can be applied to set the width of the media player, add borders and drop shadows, and apply filters and transformations to the player's appearance

Applying Styles to the Media Player (continued)

Figure 8-7 Styles for the native media player

```
/* Audio and Video Player Styles */  
  
audio {  
    box-shadow: rgb(51, 51, 51) 8px 8px 15px;  
    display: block;  
    margin: 10px auto;  
    width: 90%;  
}
```

Providing a Fallback to an Audio Clip

Figure 8-9 Adding fallback text to the audio element

```
<audio controls>
  <source src="cp_overture.mp3" type="audio/mp3" />
  <source src="cp_overture.ogg" type="audio/ogg" />
  <p><em>To play this audio clip, your browser needs to support HTML5.</em></p>
</audio>
</aside>
```

displays this paragraph
when the audio element
is not supported

fallback text displayed for
browsers that don't support
the audio element

Burton Lane's Royal Wedding.

*To play this audio clip, your browser
needs to support HTML5.*

Exploring Embedded Objects

- Plug-ins in older browsers are marked using the `embed` element

```
<embed src="url" type="mime-type"  
      width="value" height="value" />
```

where *url* is the location of the media file, *type* attribute provides the mime-type, and *width* and *height* attributes set the width and height of the media player

Plug-In Attributes

- `src`, `type`, `height`, and `width` attributes are generic attributes applied to `embed` element for any plug-in
- For example, the following `embed` element adds attributes to display the media player controls and prevent the playback from starting automatically:

```
<embed src="cp_overture.mp3"  
width="250" height="50"  
controller="yes" autoplay="no" />
```

Plug-Ins as Fallback Options

- Add `embed` element to the end of the `audio` element as the last option for a browser that does not support HTML5 multimedia elements
- Use of plug-ins has steadily declined since the widespread adoption of HTML5 standard

Video Formats and Codecs

- A video file contains codecs for the following:
 - audio
 - video images

Figure 8-11 Video codecs used on the web

Codec	Description
H.264	Developed by the MPEG group, the H.264 codec is the industry standard for high-definition video streams, movie sharing websites such as YouTube, and video plug-ins
Theora	Theora is a royalty-free codec developed by the Xiph.org Foundation that produces video streams that can be used with almost any container
VP8	VP8 is an open-source royalty-free codec owned by Google for use in Google's WebM video format
VP9	VP9 is Google's successor to the VP8 codec, offering the same video quality as VP8 at half the download size

Video Formats and Codecs (continued 1)

Figure 8-12 Video formats used on the web

Format	Description	Video Codec	File Extension(s)	MIME Type
MPEG-4	MPEG-4 or MP4 is a widely used proprietary format developed by Apple based on the Apple QuickTime movie format	H.264	.mp4 .m4v	video/mp4
Ogg	Ogg is an open source format developed by the Xiph.org Foundation using the Theora codec as an alternative to the MPEG-4 codec	Theora	.ogg	video/ogg
WebM	WebM is an open source format introduced by Google to provide royalty-free video and audio to be used with the HTML5 video element	VP8 VP9	.webm	video/webm

Video Formats and Codecs (continued 2)

Figure 8-13 Browser support for video formats

Browser	MPEG-4	Ogg	WebM
Chrome (desktop)	✓	✓	✓
Chrome (mobile)	✓	✓	✓
Firefox (desktop)	✓	✓	✓
Firefox (mobile)	✓	✓	✓
Microsoft Edge (desktop)	✓		
Internet Explorer (desktop)	✓		
Internet Explorer (mobile)	✓		
Opera (desktop)	✓	✓	✓
Opera (mobile)	✓		✓
Safari (desktop)	✓		
Safari (mobile)	✓		

Using the HTML5 `video` Element

- Videos are embedded into a web page using `video` element

```
<video attributes>
```

```
<source src="url1" type="mime-type" />
```

```
<source src="url2" type="mime-type" />
```

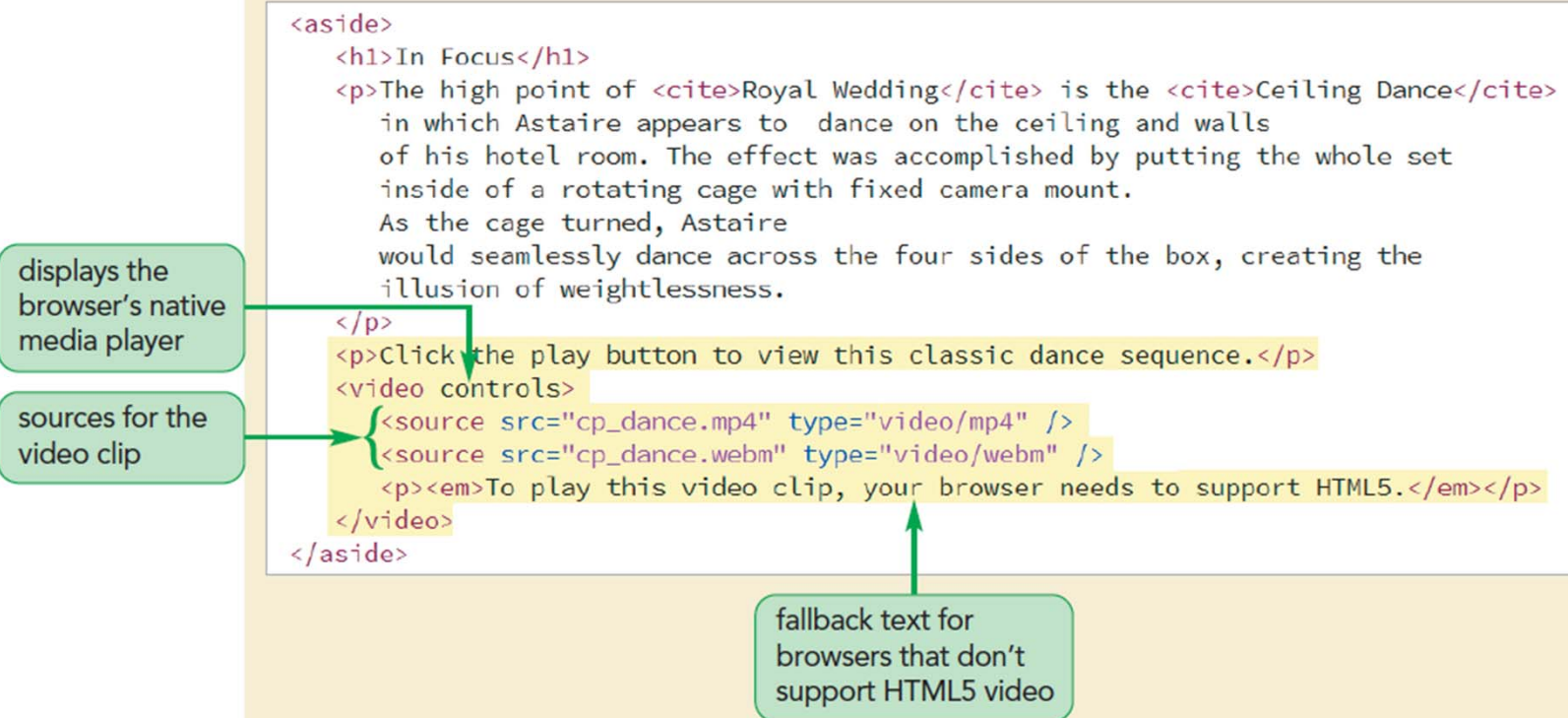
```
... </video>
```

where *attributes* are HTML attributes that control the behaviour and appearance of the video playback, *url1*, *url2*,... are the possible sources of the video

Using the HTML5 `video` Element (continued 1)

mime-type specifies the format associated with each video file

Figure 8-14 Adding a video clip to a web page



Using the HTML5 video Element (continued 2)

Figure 8-15

Defining the video player styles

add video to the
style rule selector

```
audio, video {  
  box-shadow: rgb(51, 51, 51) 8px 8px 15px;  
  display: block;  
  margin: 10px auto;  
  width: 90%;  
}
```

Using the HTML5 `video` Element (continued 3)

- `poster` attribute defines a video's preview image

```
<video poster="url">
```

```
...
```

```
</video>
```

where *url* points to an image file containing the preview image

- `poster` attribute is used as a placeholder image that is displayed when a video is being downloaded

Adding a Text Track to Video

- Text track that needs to be read or recited to visually impaired users can be added to a media clip
- Audio and video content accessible to all users
- Text tracks are added to an audio or video clip using `track` element

Adding a Text Track to Video (continued 1)

```
<track kind="type" src="url"  
label="text" srclang="lang" />
```

where,

- `kind` attribute defines the track type
- `src` attribute references a file containing the track text
- `label` attribute gives the track name
- `srclang` attribute indicates the language of the track

Adding a Text Track to Video (continued 2)

Figure 8-19 Values of the kind attribute

Kind Value	Description
captions	Brief text descriptions synced to specified time points within the media clip; designed for hearing impaired users
chapters	Chapter titles used by the media player to navigate the user to specific time points within the media clip
descriptions	Longer descriptions synced to specified time points within the media clip; designed for visually impaired users
subtitles	(the default) Translation of dialog from the media clip; the language of the subtitle must be specified in the <code>srclang</code> attribute
metadata	Metadata content used by external scripts accessing the media file

Making Tracks with WebVTT

- Tracks are stored as simple text files written in **Web Video Text Tracks** or **WebVTT** language
- Format of a WebVTT file

WEBVTT

cue1

cue2

...

where *cue1*, *cue2*,... are cues matched with specific time intervals within a media clip

Making Tracks with WebVTT (continued 1)

- List of cues is separated by a single blank line after a cue text
- White space is not ignored in WebVTT files
- General form of a cue

label

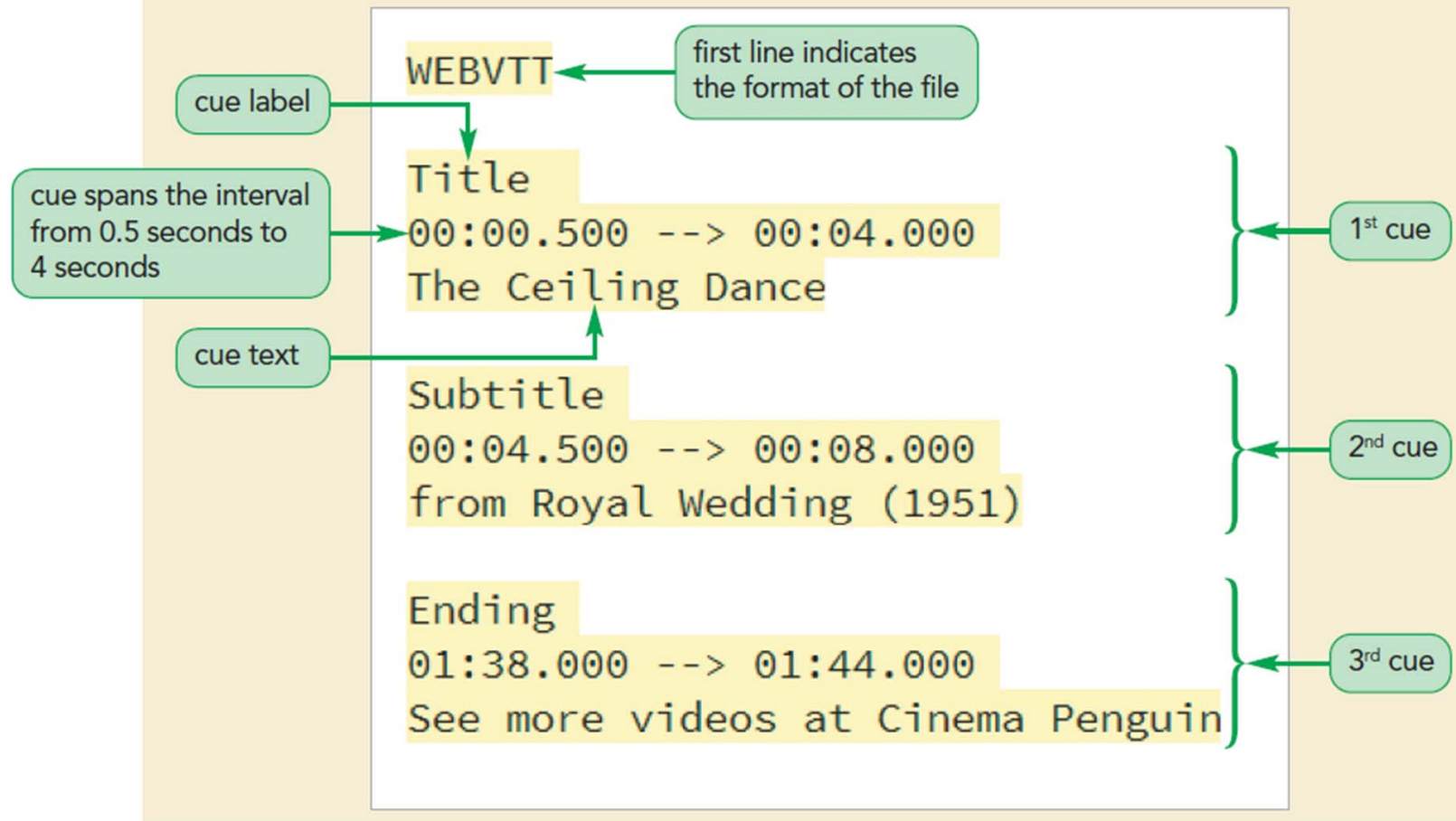
start --> stop

cue text

where *label* is the name assigned to the cue,
start and *stop* define the time interval, and
cue text is the text of the cue

Making Tracks with WebVTT (continued 2)

Figure 8-20 WebVTT file to define track text



Placing the Cue Text

- Size and position of a cue text can be set using cue settings directly after the cue's time interval

setting1:value1 setting2:value2 ...

where *setting1, setting2,...* define the size and position of the cue text and *value1, value2,...* are the setting values

- There is no space between the setting name and value

Placing the Cue Text (continued 1)

Figure 8-23 Cue attributes in WebVTT

Cue Setting	Description
<code>align:value</code>	Sets the horizontal alignment of the text within the cue, where <i>value</i> is <code>start</code> (left-aligned), <code>middle</code> (center-aligned), or <code>end</code> (right-aligned)
<code>line:value</code>	Sets the vertical position of the cue within the video window, where <i>value</i> ranges from 0% (top) to 100% (bottom)
<code>position:value</code>	Sets the horizontal position of the cue within the video window, where <i>value</i> ranges from 0% (left) to 100% (right)
<code>size:value</code>	Sets the width of the cue as a percentage of the width of the video window
<code>vertical:type</code>	Displays the cue text vertically rather than horizontally where <i>type</i> is <code>rl</code> (writing direction is right to left) or <code>lr</code> (writing direction is left to right)

Placing the Cue Text (continued 2)

Figure 8-24

Placing the cue text

places the Title and Subtitle cues near the top of the video window with the text centered

```
Title
00:00:00.500 --> 00:00:04.000 line:5% align:middle
The Ceiling Dance

Subtitle
00:00:04.500 --> 00:00:08.000 line:5% align:middle
from Royal Wedding (1951)

Ending
00:01:38.000 --> 00:01:44.000 line:80% position:95% align:end
See more videos at Cinema Penguin
```

places the Ending cue near the bottom-right corner of the video window with the text right-aligned

Applying Styles to Track Cues

- cue pseudo-element to format the appearance of the cues appearing within a media clip

```
    ::cue {  
      styles  
    }
```
- Styles for the cue pseudo-element are limited to background, color, font, opacity, outline, text-decoration, text-shadow, visibility, and white-space properties

Applying Styles to Track Cues (continued 1)

- Format specific cues or text strings within a cue using the following markup tags:
 - `<i></i>` for italicized text
 - `` for bold-faced text
 - `<u></u>` for underlined text
 - `` to mark spans of text
 - `<ruby></ruby>` to mark ruby text
 - `<rt></rt>` to mark ruby text

Applying Styles to Track Cues (continued 2)

- WebVTT supports tags that are not part of the HTML library
- `<c></c>` tag is used to mark text strings belonging to a particular class

`<c .classname></c>`

- `<v></v>` tag is used for captions that distinguish between one voice and another

`<v name></v>`

Applying Styles to Track Cues (continued 3)

Figure 8-26

Applying a class to cue text

markup tag for
the Main class

WEBVTT

Title

00:00.500 --> 00:04.000 line:5% align:middle

`<c.Main>`The Ceiling Dance`</c>`

Subtitle

00:04.500 --> 00:08.000 line:5% align:middle

from Royal Wedding (1951)

Ending

01:38.000 --> 01:44.000 line:80% position:95% align:end

See more videos at `<i>`Cinema Penguin`</i>`

displays the website
title in italics

Using Third-Party Video Players

- `object` element is used to define browsers with plug-ins

`<object attributes>`

`parameters`

`</object>`

where *attributes* define the object and *parameters* are values passed to the object controlling the object's appearance and actions

Using Third-Party Video Players (continued)

- Parameters of the object are defined using `param` element

```
<param name="name" value="value" />
```

where *name* is the name of the parameter and *value* is the parameter's value

Exploring the Flash Player

- The most-used plug-in for video playback is Adobe Flash player embedded using the following object element:

```
<object data="url"  
type="application/x-shockwave-flash"  
width="value" height="value">  
  <param name="movie" value="url" />  
  parameters  
</object>
```

Exploring the Flash Player (continued)

Figure 8-29 Parameters of the Flash player

Name	Value(s)	Description
<code>bgcolor</code>	<code>color value</code>	Sets the background color of the player
<code>flashvar</code>	<code>text</code>	Contains text values that are passed to the player as variables to control the behavior and content of the movie
<code>id</code>	<code>text</code>	Identifies the movie so that it can be referenced
<code>loop</code>	<code>true</code> <code>false</code>	Plays the movie in a continuous loop
<code>menu</code>	<code>true</code> <code>false</code>	Displays a popup menu when a user right-clicks the player
<code>name</code>	<code>text</code>	Names the movie so that it can be referenced
<code>play</code>	<code>true</code> <code>false</code>	Starts the player when the page loads
<code>quality</code>	<code>low</code> <code>autolow</code> <code>autohigh</code> <code>medium</code> <code>high</code> <code>best</code>	Sets the playback quality of the movie; low values favor playback speed over display quality; high values favor display quality over playback speed
<code>scale</code>	<code>showall</code> <code>noborder</code> <code>exactfit</code>	Defines how the movie clip is scaled within the defined space; a value of <code>showall</code> makes the entire clip visible in the specified area without distortion; a value of <code>noborder</code> scales the movie to fill the specified area without distortion but possibly with some cropping; a value of <code>exactfit</code> makes the entire movie visible in the specified area without trying to preserve the original aspect ratio
<code>wmode</code>	<code>window</code> <code>opaque</code> <code>transparent</code>	Sets the appearance of the player against the page background; a value of <code>window</code> causes the movie to play within its own window; a value of <code>opaque</code> hides everything behind the player; a value of <code>transparent</code> allows the page background to show through transparent colors in the player

Embedding Videos from YouTube

- YouTube videos are easy to embed in a web page using YouTube's HTML5 video player
- Click the Share button below the YouTube video player to share it
- YouTube provides options to post a hypertext link to the video to a multitude of social media sites or to share the link via e-mail

Embedding Videos from YouTube

(continued 1)

- To embed a video within a website, click Embed, which brings up a preview of the embedded player and the HTML code that needs to be added to the web page
- The general code for the embedded player is

```
<iframe width="value" height="value"  
src="url"  
frameborder="0" allowfullscreen>  
</iframe>
```

where, the *url* provides the link to the

Embedding Videos from YouTube

(continued 2)

YouTube video

- `width` and `height` attributes define the dimensions of the player embedded on a web page
- `frameborder` attribute sets the width of the border around the player in pixels
- `allowfullscreen` attribute allows the user to play the video in full screen mode

Embedding Videos from YouTube

(continued 3)

- **iframe element:** Used to mark **inline frames**
- **Inline frames:** Windows embedded within a web page that display the content of another page or Internet resource

HTML5 Video Players

- HTML5 video player works within a browser with CSS and JavaScript files
- It presents a customizable player that can be adapted to the needs of business or organization
- For example, YouTube player that provides both the player and a hosting service for the video content

HTML5 Video Players (continued)

- HTML5 includes the following video players:
 - **JWPlayer** (*www.jwplayer.com*)
 - **Video.js** (*www.videojs.com*)
 - **MediaElement.js** (*mediaelementjs.com*)
 - **Projekktor** (*www.projekktor.com*)
 - **Flowplayer/Flash player** (*flowplayer.org*)

Introducing Transitions

- **Transition:** Change in an object's style from the initial state to the ending state, usually in response to an event initiated by the user or the browser
- It slows down the change from one color to another and provides intermediate styles

Introducing Transitions (continued)

- To create transition, employ the following transition style:

`transition: property duration;`

where,

- *property* is a property of the object that changes between the initial and end states
- *duration* is the transition time in seconds or milliseconds

Setting the Transition Timing

- Varying speed of transition is defined using `transition: property duration timing-function;`

where *timing-function* is one of the following keywords:

- `ease`: (the default) Transition occurs more rapidly at the beginning and slows down near the end

Setting the Transition Timing

(continued 1)

- `ease-in`: Transition starts slowly and maintains a constant rate until the finish
- `ease-out`: Transition starts at a constant rate and then slows down toward the finish
- `ease-in-out`: Transition starts slowly, reaches a constant rate, and then slows down toward the finish
- `linear`: Transition is applied at a constant rate throughout the duration

Setting the Transition Timing (continued 2)

- Timing function can be visualized as a graph
- It shows the progress of transition vs. duration
- The graphical representation of the timing function is the basis of another measure of transition timing using

`cubic-bezier(n, n, n, n)`

where *n* parameter values define the shape of the timing curve

Delaying a Transition

- Transition does not need to start immediately after the event that triggers it
- Start of the transition can be delayed by adding *delay* value to the following:

```
transition: property duration timing-  
function delay;
```

where *delay* is measured in seconds or milliseconds

Creating a Hover Transition

Figure 8-39

Style rules for the initial state and end state

initial state displays the
hypertext links in white
with a small text shadow

end state displays the
hypertext links in light
orange with a larger font
and a larger text shadow

```
/* Transition Styles */  
  
nav#topLinks a {  
    color: rgb(255, 255, 255);  
    font-size: 1em;  
    letter-spacing: 0em;  
    text-shadow: rgba(0, 0, 0, 1) 1px -1px 1px;  
}  
  
nav#topLinks a:hover {  
    color: rgb(255, 183, 25);  
    font-size: 3em;  
    letter-spacing: 0.1em;  
    text-shadow: rgba(0, 0, 0, 0.5) 15px -3px 8px;  
}
```

- Transition property can be added to slow down the transition from initial to end state

Creating a Hover Transition (continued)

- Limitations of transition
 - It can only be run when a CSS property is being changed, such as during the hover event
 - It is run once and cannot be looped for repetition
 - Initial and end states of the transition can be defined but not the styles of intermediate states
- Animation is created to overcome the limitations

Animating Objects with CSS

- **Key frame:** Sequence of changing images to create illusive movement for animation
- CSS replaces the concept of key frame images with key frame styles that are applied in rapid succession to a page object

- @keyframes Rule

```
@keyframes name {  
  keyframe1 {styles;}  
  keyframe2 {styles;}  
  ...}
```

Animating Objects with CSS (continued)

- *name* provides the name or title of the animated sequence
- *keyframe1* , *keyframe2* , ... defines the progress of individual key frames that are expressed as percentages or with keywords `from` and `to`
- *styles* are styles applied within each key frame

Applying an Animation

- Key frames animation is applied to an object using `animation-name` and `animation-duration` properties

`animation-name: keyframes;`

`animation-duration: times;`

where *keyframes* is a comma-separated list of animations applied to the object using the names from the *@keyframes* rule and *times* are the lengths of each animation expressed in seconds or milliseconds

Applying an Animation (continued)

Figure 8-46 Animation Properties

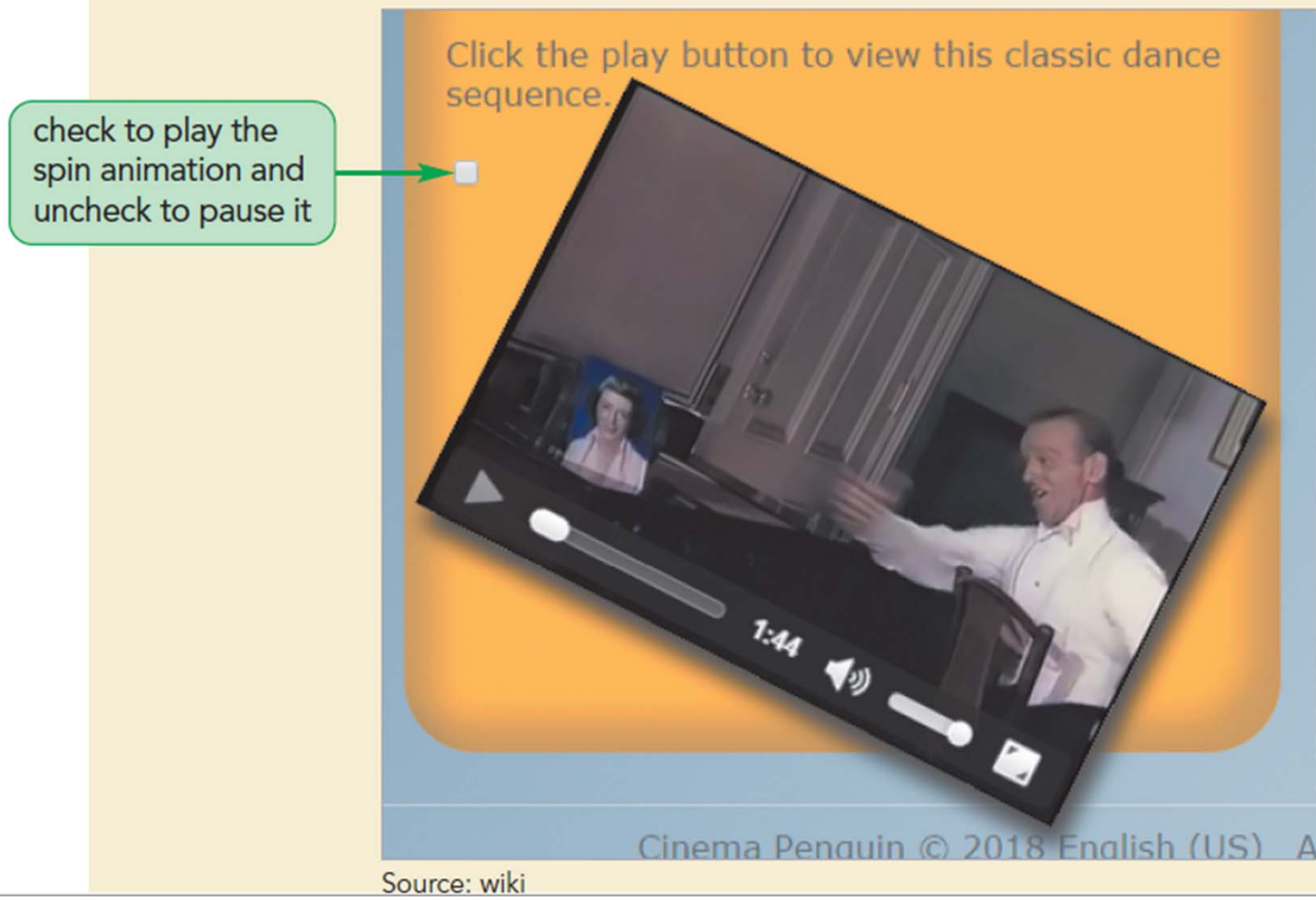
Property	Description
<code>animation-name = <i>keyframes</i></code>	Assigns the <i>keyframes</i> animation to the object
<code>animation-duration = <i>time</i></code>	Sets the length of the animation in seconds or milliseconds (default = 0s)
<code>animation-timing-function = ease ease-in ease-out ease-in-out linear cubic-bezier(<i>n,n,n,n</i>)</code>	Defines the default timing between key frames in the animation (default = ease)
<code>animation-delay = <i>time</i></code>	Sets the delay time in seconds and milliseconds before animation is started (default = 0s)
<code>animation-iteration-count = <i>value</i> infinite</code>	Specifies the number of times the animation is played, where <i>value</i> is an integer and <i>infinite</i> repeats the animation without stopping (default = 1)
<code>animation-direction = normal reverse alternate alternate-reverse</code>	Defines the direction of the animation, where <i>normal</i> plays the animation as defined in the <i>@keyframes</i> rule, <i>reverse</i> reverses the order, <i>alternate</i> plays the animation in the normal direction followed by the reverse direction (for multiple iterations), and <i>alternate-reverse</i> plays the animation in reverse direction followed by normal direction (default=normal)
<code>animation-fill-mode = none backwards forwards both</code>	Defines what styles from the animation are applied to the object outside the time it is running, where <i>none</i> does not apply any styles, <i>backwards</i> applies the styles from the first key frame, <i>forwards</i> applies the styles from the last key frame, and <i>both</i> applies styles in both directions (default=none)
<code>animation-play-state = running paused</code>	Defines whether the animation is running or paused (default = running)

Controlling an Animation



- Animation can have two states of operation — play or pause
- Check box can be used to control animation
- Selecting the check box will play the animation
- Unselecting the check box will pause the animation

Controlling an Animation (continued 1)

Figure 8-51 Using a check box to control the animation playback



Controlling an Animation (continued 2)

- Check box can be replaced with more attractive icons
- Display symbol  to run the animation
- Display symbol  to pause the animation
- The two symbols have the Unicode values \21bb and \270b, respectively

Controlling an Animation (continued 3)

Figure 8-52 Displaying playback icons

hides the rotateVideo
check box

inserts the ⏮ symbol
when the rotateVideo
check box is not
checked

inserts the 🖱 symbol
when the rotateVideo
check box is checked

```
/* Animation Icon Styles */
```

```
{input#rotateVideo {  
  display: none;  
}}
```

```
{input#rotateVideo:not(:checked)+label::after {  
  content: "\21bb";  
}}
```

```
{input#rotateVideo:checked+label::after {  
  content: "\270b";  
}}
```