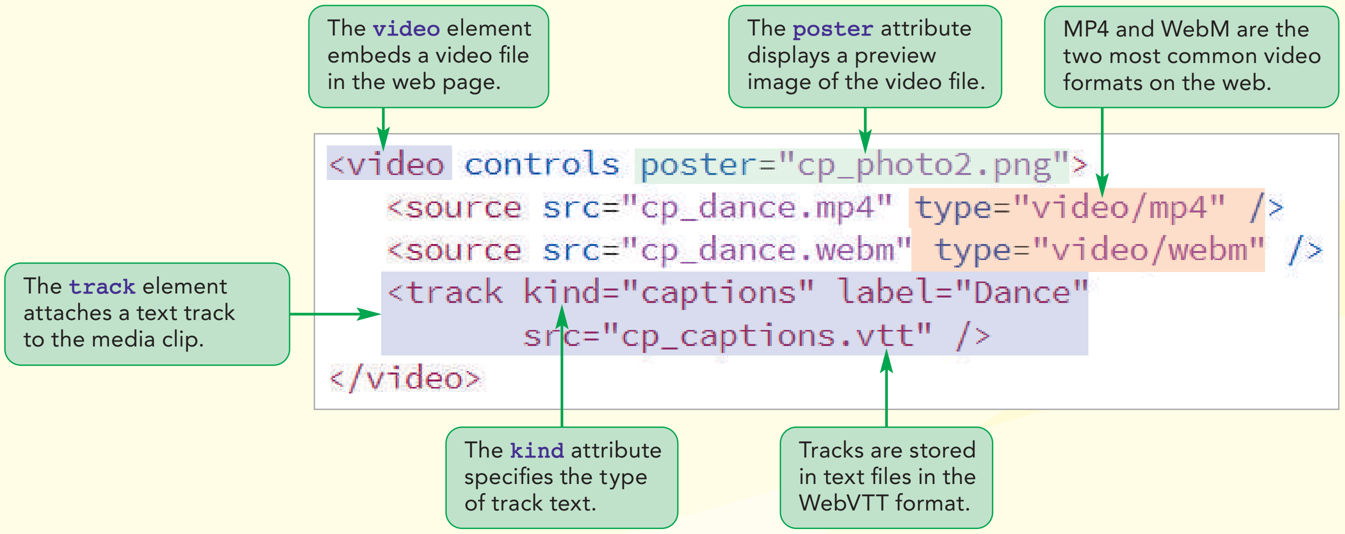
Session 8.2 Visual Overview:



```
<video controls poster="cp_photo2.png">
   <source src="cp_dance.mp4" type="video/mp4" />
   <source src="cp_dance.webm" type="video/webm" />
   <track kind="captions" label="Dance"
          src="cp_captions.vtt" />
</video>
```

The video element embeds a video file in the web page.

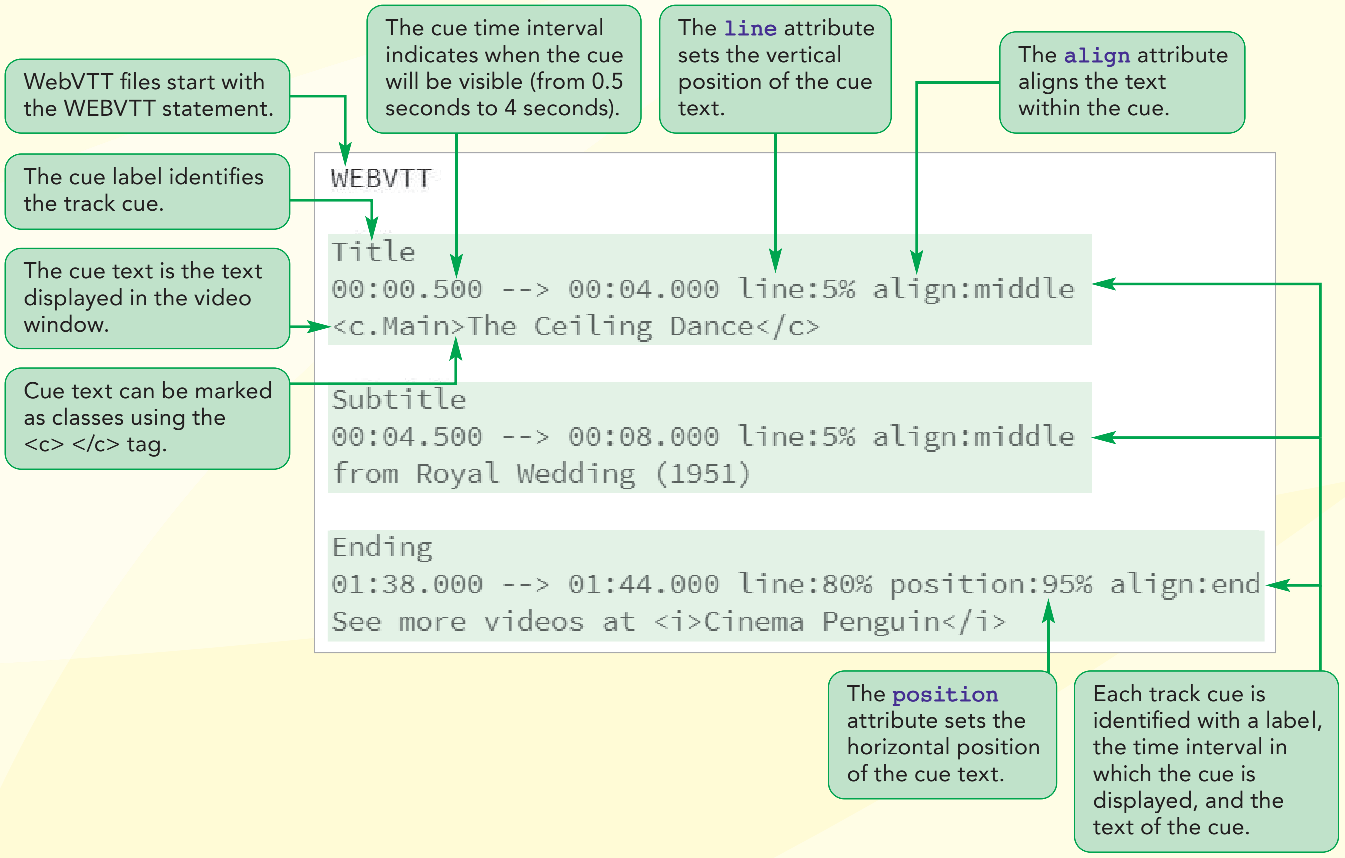
The poster attribute displays a preview image of the video file.

MP4 and WebM are the two most common video formats on the web.

The track element attaches a text track to the media clip.

The kind attribute specifies the type of track text.

Tracks are stored in text files in the WebVTT format.



```
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>
```

WebVTT files start with the WEBVTT statement.

The cue label identifies the track cue.

The cue text is the text displayed in the video window.

Cue text can be marked as classes using the <c> </c> tag.

The cue time interval indicates when the cue will be visible (from 0.5 seconds to 4 seconds).

The line attribute sets the vertical position of the cue text.

The align attribute aligns the text within the cue.

The position attribute sets the horizontal position of the cue text.

Each track cue is identified with a label, the time interval in which the cue is displayed, and the text of the cue.

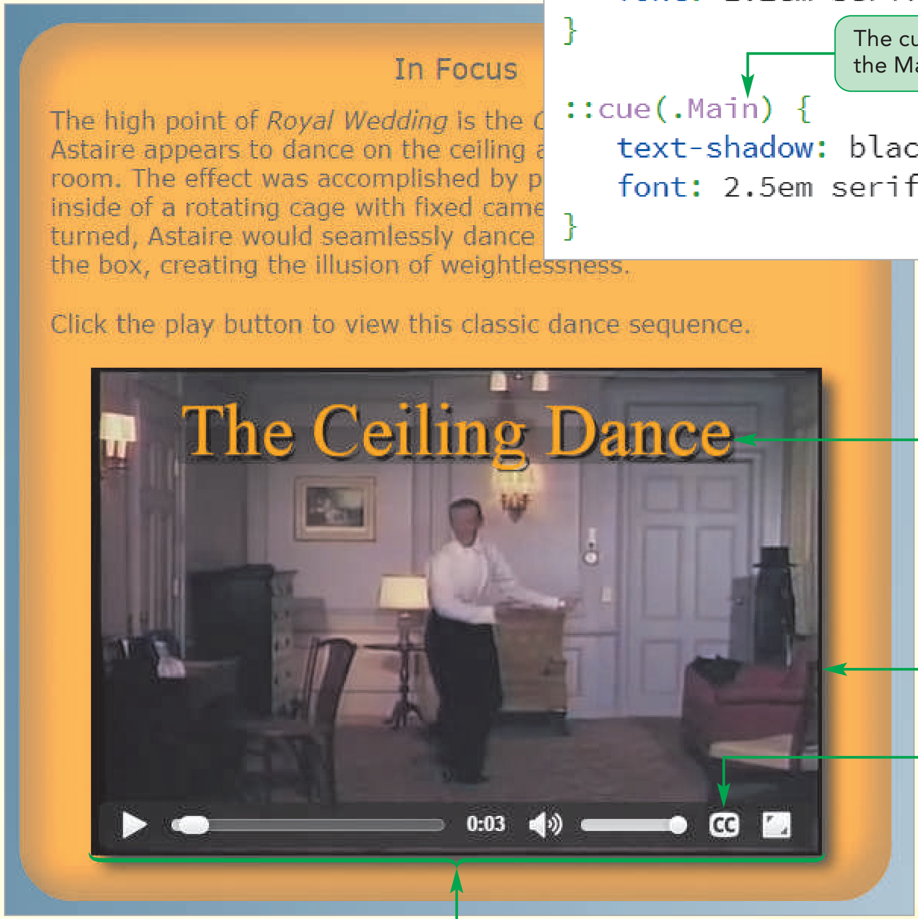
Playing Web Video

The cue pseudo-element selects the cues from the media track.

```
::cue {
   background: rgba(0, 0, 0, 0.3);
   color: orange;
   font: 1.2em serif;
}

::cue(.Main) {
   text-shadow: black 3px 3px 0px;
   font: 2.5em serif;
}
```

The cue text is from the Main class.



In Focus

The high point of *Royal Wedding* is the C Astaire appears to dance on the ceiling a room. The effect was accomplished by p inside of a rotating cage with fixed came turned, Astaire would seamlessly dance the box, creating the illusion of weightlessness.

Click the play button to view this classic dance sequence.

Source: wiki

The Title cue text as rendered in the video window.

Native media player provided by the browser to play video files.

CC (closed captioning) button is used to display captions within the video player.

The media player displays controls for video playback.

Exploring Digital Video

In this session, you explore how to embed video within your web pages. Before exploring the HTML `video` elements, you examine some of the issues involved with producing video files suitable for the web.

Video Formats and Codecs

A video file typically contains two codecs: one codec for audio and another for the video images. The audio codecs are the same ones you examined in the last session. Figure 8-11 describes the most commonly used video codecs on the web.

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

The most popular video codec is H.264 used by YouTube and most commercial vendors; however, because H.264 is a commercial product, it is not royalty free. This is not an issue if you are creating a video that is not actually being sold. If you are creating a commercial video that uses the H.264 codec, you might have to pay licensing fees depending on the number of subscribers to your video content. The Theora, VP8, and VP9 codecs are royalty free, but they are not as widely supported at the time of this writing.

Browser support for video containers is focused on three formats: MP4, Ogg, and WebM, with multiple combinations of video and audio codecs available within each container. Figure 8-12 summarizes these formats and their use on the web.

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 content suffers the same limitation as audio content in that no single format has universal support among all browsers and devices. Thus, as with audio content, you may have to supply multiple versions of the same video if you want the widest cross-browser support. See Figure 8-13.

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)	✓		

The level of support for video formats is constantly changing as are the video formats themselves. As always, the best way to determine whether a browser supports a particular video format is to test the video file on that browser.

Using the HTML5 `video` Element

Videos are embedded into a web page using the following `video` element

```
<video attributes>
   <source src="url1" type="mime-type" />
   <source src="url2" type="mime-type" />
 …
</video>
```

where *`attributes`* are the HTML attributes that control the behavior and appearance of the video playback, *`url1`*, *`url2`*, and so on are the possible sources of the video, and *`mime-type`* specifies the format associated with each video file. As with sources for the `audio` element, a browser uses the first source it finds in a format it supports. Fallback content can also be added after the list of video sources for browsers that don't support HTML5 video. The `video` element supports many of the same attributes used within the `audio` element shown earlier in Figure 8-2.

The following code embeds two possible video files on the web page with a fallback message for browsers that don't support the `video` element:

```
<video controls>
   <source src="cp_dance5.mp4" type="video/mp4" />
   <source src="cp_dance5.webm" type="video/webm" />
   <p><em>To play this video clip, your browser needs
   to support HTML5.</em></p>
</video>
```

Maxine has a video clip of a classic dance sequence from *Royal Wedding* in which Fred Astaire appears to dance on the walls and ceiling of his hotel room. She has two versions of the clip: one in MP4 format and the other in the WebM format. Use the `video` element now to embed these videos on her web page.

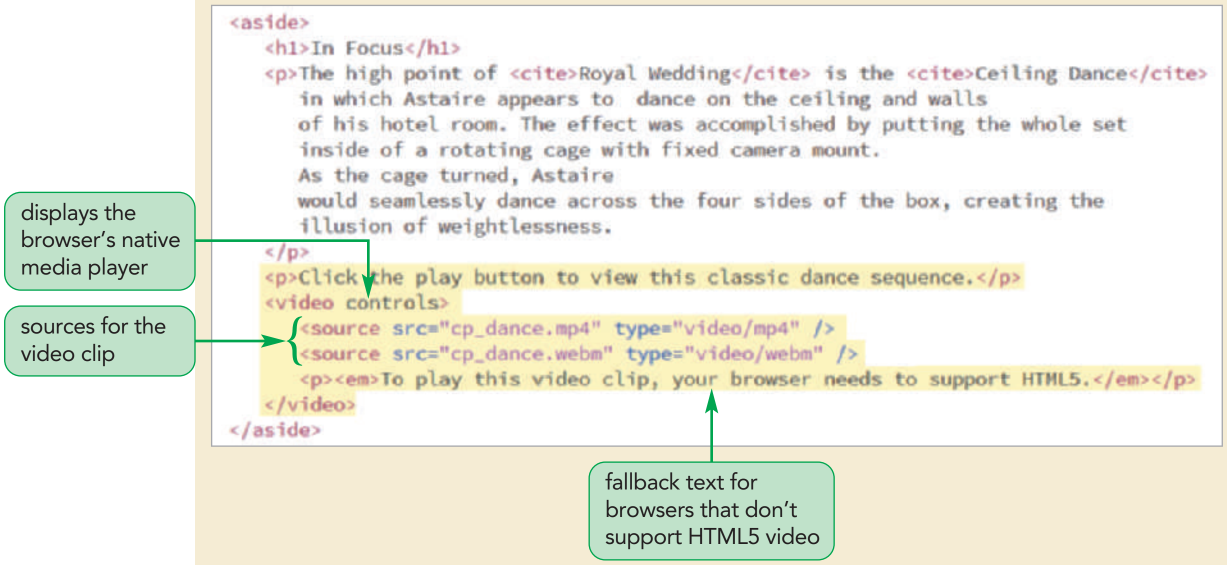
To embed a video file into the web page:

1. If you took a break after the previous session, make sure the **cp_royal.html** file is open in your editor.
2. Scroll down to the `aside` element titled In Focus and add the following code directly before the closing `</aside>` tag:

```
<p>Click the play button to view this classic dance
sequence.</p>
<video controls>
   <source src="cp_dance.mp4" type="video/mp4" />
   <source src="cp_dance.webm" type="video/webm" />
   <p><em>To play this video clip, your browser needs to
      support HTML5.</em></p>
</video>
```

Figure 8-14 highlights the code for the embedded video.

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



```
<aside>
   <h1>In Focus</h1>
   <p>The high point of <cite>Royal Wedding</cite> is the <cite>Ceiling Dance</cite>
      in which Astaire appears to  dance on the ceiling and walls
      of his hotel room. The effect was accomplished by putting the whole set
      inside of a rotating cage with fixed camera mount.
      As the cage turned, Astaire
      would seamlessly dance across the four sides of the box, creating the
      illusion of weightlessness.
   </p>
   <p>Click the play button to view this classic dance sequence.</p>
   <video controls>
      <source src="cp_dance.mp4" type="video/mp4" />
      <source src="cp_dance.webm" type="video/webm" />
      <p><em>To play this video clip, your browser needs to support HTML5.</em></p>
   </video>
</aside>
```

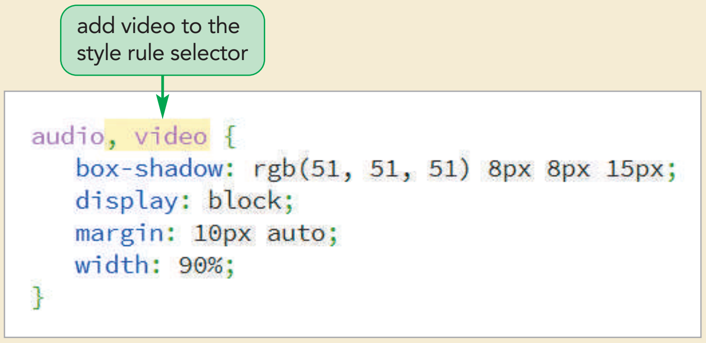
displays the browser's native media player
sources for the video clip
fallback text for browsers that don't support HTML5 video

3. Save your changes to the file.

   Next, you modify the cp_media.css style sheet file to format the appearance of the video media player for your browser.
4. Go to the **mp_media.css** file in your editor.
5. Modify the style rule for the `audio` element by adding the **video** selector.

   Figure 8-15 highlights the modified style rule.

Figure 8-15 Defining the video player styles

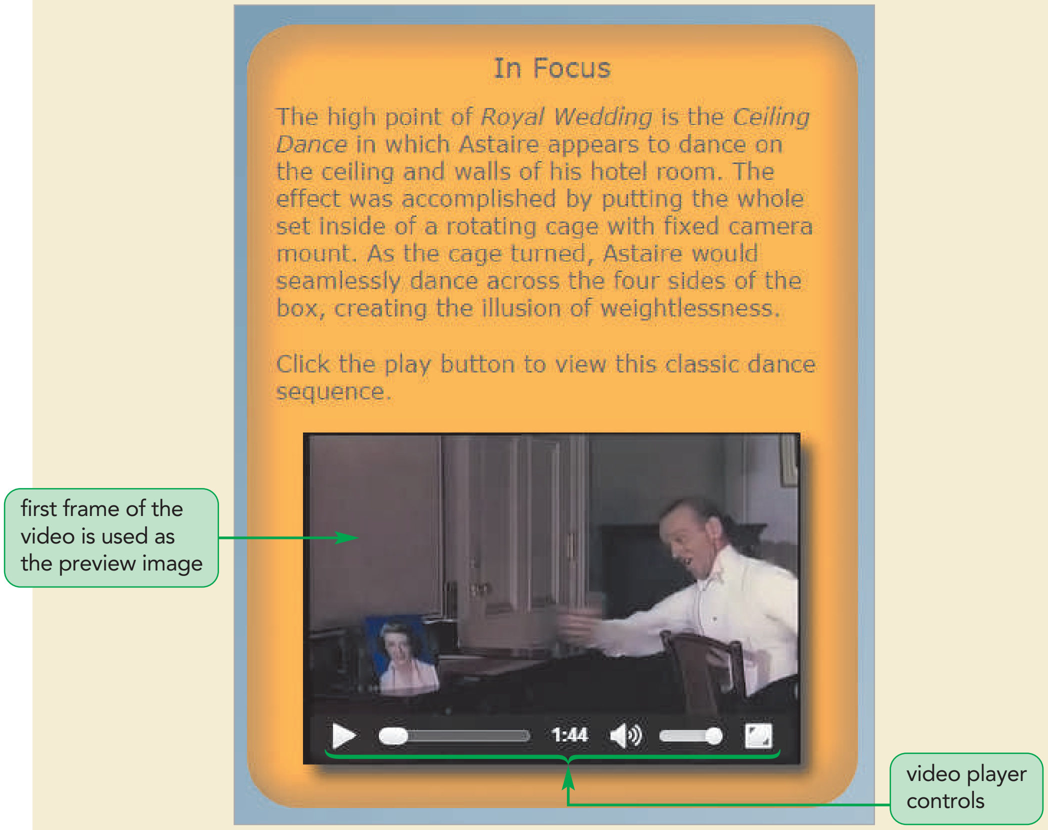


```
audio, video {
   box-shadow: rgb(51, 51, 51) 8px 8px 15px;
   display: block;
   margin: 10px auto;
   width: 90%;
}
```

6. Save your changes to the style sheet and then reopen the **cp_royal.html** file in your browser.
7. Click the play button on your browser's media player to play the video clip.

   Figure 8-16 shows the video clip in action as it replays the ceiling dance sequence from *Royal Wedding*.

Figure 8-16 Video clip embedded in the web page



In Focus

The high point of *Royal Wedding* is the *Ceiling Dance* in which Astaire appears to dance on the ceiling and walls of his hotel room. The effect was accomplished by putting the whole set inside of a rotating cage with fixed camera mount. As the cage turned, Astaire would seamlessly dance across the four sides of the box, creating the illusion of weightlessness.

Click the play button to view this classic dance sequence.

When the media player initially loads a video file, the player shows the first video frame as a preview of the video's content. Maxine would like to replace that preview image with an image of Astaire dancing on his apartment room's wall. To define the video's preview image, you apply the following `poster` attribute to the `video` element

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

where *`url`* points to an image file containing the preview image. The `poster` attribute is also used as a placeholder image that is displayed when the video is still being downloaded or used in place of the video if the browser fails to download the video file at all.

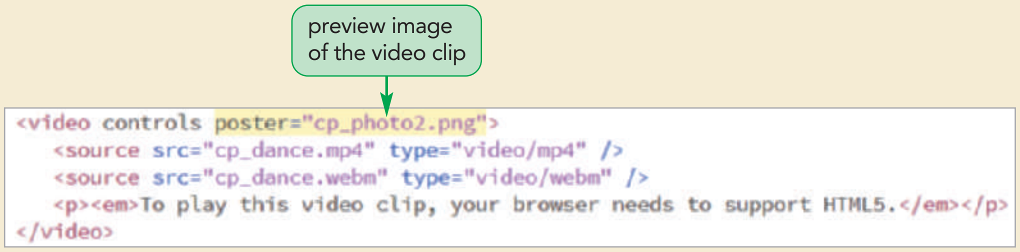
Maxine suggests you use the cp_photo2.png file as the poster image for the video clip.

To set the video's poster image:

1. Return to the **cp_royal.html** file in your editor.
2. Add the following attribute to the `video` element: **`poster="cp_photo2.png"`**.

   Figure 8-17 highlights the poster attribute.

Figure 8-17 **Defining a poster image for the video**

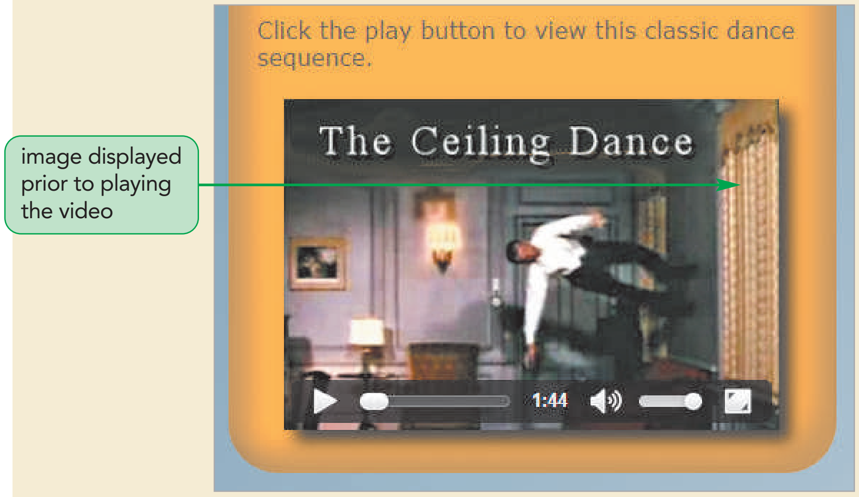


```
<video controls poster="cp_photo2.png">
   <source src="cp_dance.mp4" type="video/mp4" />
   <source src="cp_dance.webm" type="video/webm" />
   <p><em>To play this video clip, your browser needs to support HTML5.</em></p>
</video>
```

3. Save your changes to the file and then reload cp_royal.html in your browser.

   Figure 8-18 shows the poster image applied to the video of the ceiling dance sequence.

Figure 8-18 **Video clip poster image**


Click the play button to view this classic dance sequence.
The Ceiling Dance
1:44
image displayed prior to playing the video

Source: wiki

Maxine suggests that the video clip would benefit from some descriptive captions. Rather than modifying the video clip itself, you can add those captions using media tracks.

Adding a Text Track to Video

With the increased reliance on multimedia on the web comes the responsibility of making audio and video content accessible to all users. This can be done by adding a text track to the media clip that can be read or recited to visually impaired users. Text tracks are added to an audio or video clip using the following `track` element

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

where the `kind` attribute defines the `track` type, the `src` attribute references a file containing the track text, the `label` attribute gives the track name, and the `srclang` attribute indicates the language of the track. A single media clip might be associated with multiple track types as indicated by the value of the `kind` attribute. Figure 8-19 lists the different kinds of tracks that can be associated with an audio or video file.

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 `srclang` attribute
`metadata`	Metadata content used by external scripts accessing the media file

For example, the following code attaches six tracks to the story.mp4 video file:

```
<video controls>
   <source src="story.mp4" type="video/mp4" />
   <track kind="captions" src="captions.vtt" label=" Captions" />
   <track kind="chapters" src="chapters.vtt" label="Chapters" />
   <track kind="subtitles" src="english.vtt" srclang="en"
default />
   <track kind="subtitles" src="french.vtt" srclang="fr" />
   <track kind="subtitles" src="spanish.vtt" srclang="es" />
   <track kind="descriptions" src="summary.vtt" label="Summary" />
</video>
```

The tracks contain captions, chapter titles, subtitles in English, French, and Spanish, and finally a track that summarizes the contents of the video. The media player can only show one of these tracks at a time. The active track by is marked with the `default` attribute, which enables that track and disables all of the other tracks. In this case, the track English subtitles is enabled while all of the other tracks are disabled. The user can choose a different track from the media player, usually by selecting the track from one of the media player controls. Note that the `default` attribute is required, even if the track list contains only one track.

Making Tracks with WebVTT

TIP

A WebVTT file has the file extension .vtt.

Tracks are stored as simple text files written in the **Web Video Text Tracks** or **WebVTT** language. The format of a WebVTT file follows the structure

```
WEBVTT

cue1

cue2
…
```

where *cue1*, *cue2*, and so on are cues matched with specific time intervals within the media clip. Note that the list of cues is separated by a single blank line after the cue text. Unlike HTML, white space is not ignored in WebVTT files.

Each cue has the general form

```
label
start --> stop
cue text
```

where *label* is the name assigned to the cue, *start* and *stop* define the time interval associated with the cue, and *cue text* is the text of the cue. Times are entered in the format *mm*:*ss*.*ms* for the minutes, seconds, and milliseconds values. For longer clips, you can include an hours value, writing the time in the form *hh*:*mm*:*ss*.*ms*.

The following code adds two cues to the WebVTT file:

```
WEBVTT

Intro
00:00.500 --> 00:09.000
Once upon a time

Conclusion
14:20.000 --> 14:30.000
And they all lived happily ever after
```

TIP

Cue text entered on multiple lines in the WebVTT file will also be displayed on multiple lines when played back.

The Intro cue, “Once upon a time”, starts at the half-second mark and runs through the 9-second mark; the Conclusion cue, “And they all lived happily ever after”, covers the time interval from 14 minutes 20 seconds through 14 minutes 30 seconds of the clip.

Create a track file now for the ceiling dance video displaying a title and subtitle at the start of the video and an advertisement for Cinema Penguin near the end.

To create a track file:

1. Use a text editor to open the **cp_captions_txt.vtt** file from the html08 ▸ tutorial folder. Save this blank text file as **cp_captions.vtt**.
2. In the first line of the file, enter **WEBVTT**.
3. Insert a blank line followed by the text for the Title cue spanning the interval from the first half-second up to the fourth second:

```
Title
00:00.500 --> 00:04.000
The Ceiling Dance
```

4. Insert a blank line followed by the Subtitle cue in the interval from 4.5 seconds through 8 seconds:

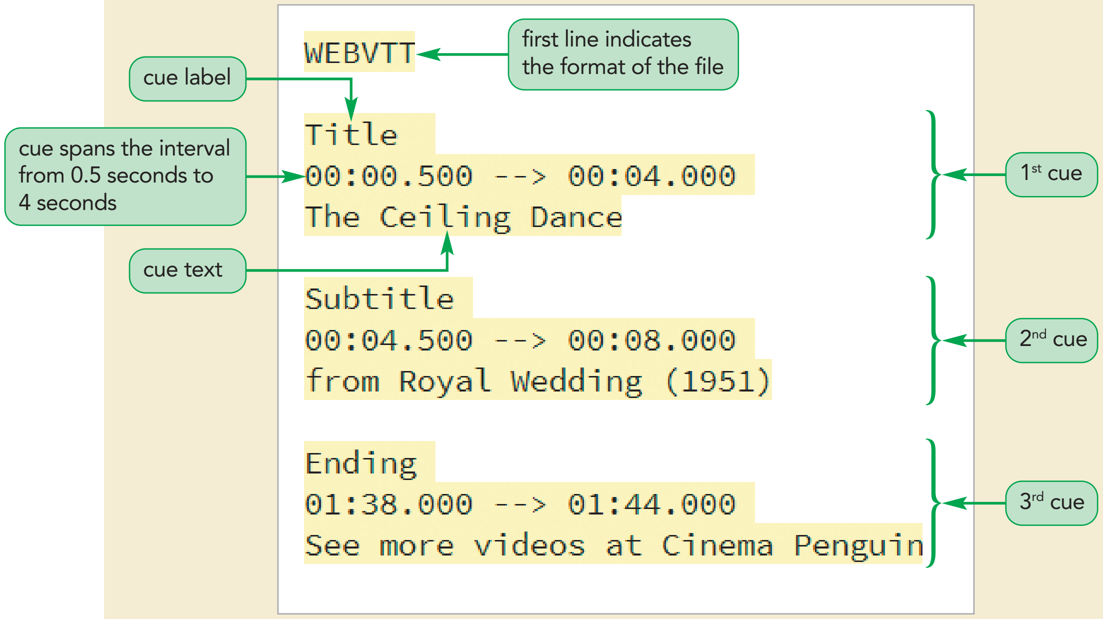
```
Subtitle
00:04.500 --> 00:08.000
from Royal Wedding (1951)
```

5. Insert a blank line followed by the Ending cue in the interval from 1 minute 38 seconds through 1 minute 44 seconds:

```
Ending
01:38.000 --> 01:44.000
See more videos at Cinema Penguin
```

Figure 8-20 describes the contents of the cp_captions.vtt file.

Figure 8-20 WebVTT file to define track text



```
WEBVTT

Title
00:00.500 --> 00:04.000
The Ceiling Dance

Subtitle
00:04.500 --> 00:08.000
from Royal Wedding (1951)

Ending
01:38.000 --> 01:44.000
See more videos at Cinema Penguin
```

first line indicates the format of the file
cue label
cue spans the interval from 0.5 seconds to 4 seconds
cue text
1st cue
2nd cue
3rd cue

6. Save your changes to the file.

Next, you apply the cp_captions.vtt file to the dance sequence video. If you are testing your page on Google Chrome or Opera, be aware that you will have to upload your files to a web server. Neither of these browsers will open track files stored locally.

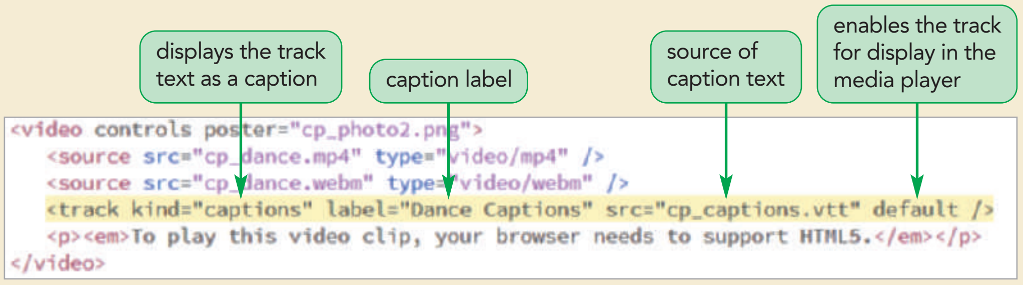
To add captions to a video clip:

1. Return to the **cp_royal.html** file in your editor and scroll down to the `video` element.
2. Directly after the second `source` element, insert the following track:

```
<track kind="captions" label="Dance Captions"
 src="cp_captions.vtt" default />
```

Figure 8-21 highlights the HTML code for the track.

Figure 8-21 **Applying a track to a video clip**

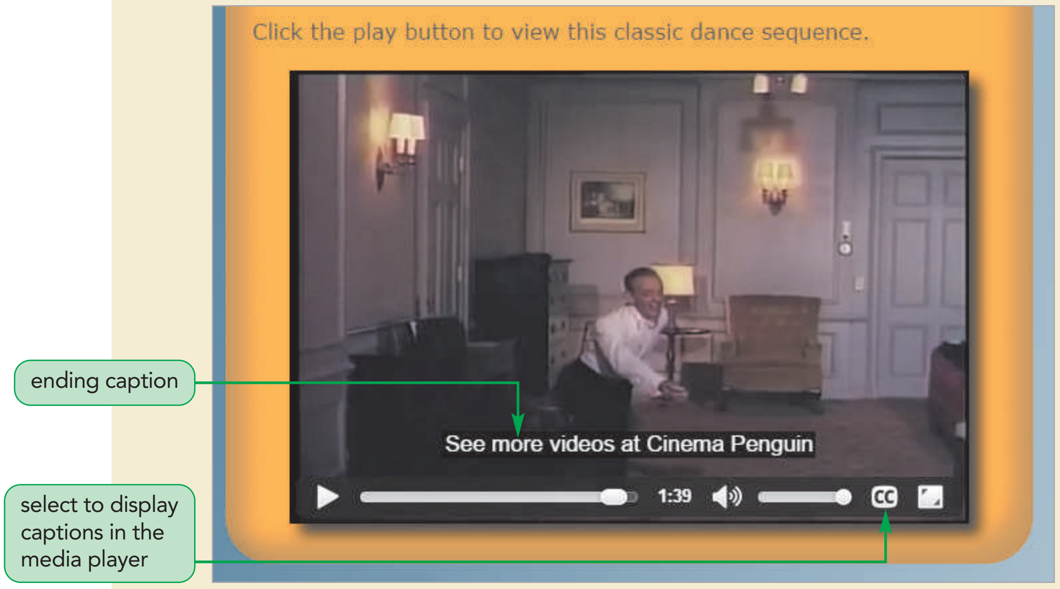

displays the track text as a caption
caption label
source of caption text
enables the track for display in the media player

```
<video controls poster="cp_photo2.png">
   <source src="cp_dance.mp4" type="video/mp4" />
   <source src="cp_dance.webm" type="video/webm" />
   <track kind="captions" label="Dance Captions" src="cp_captions.vtt" default />
   <p><em>To play this video clip, your browser needs to support HTML5.</em></p>
</video>
```

3. Save your changes to the file and then reload cp_royal.html in your browser. If you are using Google Chrome or Opera, upload your files to a web server for testing.
4. Click the play button on your browser's media player and, if necessary, click the Closed Captioning button to display the video captions. Verify that captions appear at the half-second, 4.5 second, and 1 minute 38 second marks in the video.

   Figure 8-22 shows the final caption from the video clip.

Figure 8-22 **Caption in the ceiling dance movie**


Click the play button to view this classic dance sequence.
See more videos at Cinema Penguin
1:39
ending caption
select to display captions in the media player

Trouble? If you don't see any captions, check the code in your WebVTT file against the code shown in Figure 8-20. Make sure your text matches that figure because the WebVTT syntax must be strictly followed.

By default, the caption cues are centered at the bottom of the video window. To change the position of the cues, you can edit the settings in the WebVTT file, as discussed next.

Placing the Cue Text

The size and position of the cue text can be set by adding the following cue settings directly after the cue's time interval

```
setting1:value1 setting2:value2 …
```

where *`setting1`*, *`setting2`*, and so on define the size and position of the cue text and *`value1`*, *`value2`*, and so on are the setting values. Note that there is no space between the setting name and value. Figure 8-23 describes the different settings supported in the WebVTT language.

Figure 8-23 **Cue attributes in WebVTT**

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

TIP

To center the cue in the video window, set the line and position values to 50% and the align value to middle.

By applying the `line` and `align` settings, the Intro cue in the following track is placed at the top of the video window and centered horizontally. By applying the line, position, and align settings, the Conclusion cue is placed at the bottom-left corner of the video window with the cue text left-aligned. Note that the default placement of the cue is the bottom center of the video window.

```
Intro
00:00.500 --> 00:09.000 line:0% align:middle
Once upon a time

Conclusion
14:20.000 --> 14:30.000 line:100% position:0% align:start
And they all lived happily ever after
```

Maxine suggests that you center the Title and Subtitle cues for the ceiling dance video near the top of the video window and place the Ending cue near the bottom-right corner with the text right-aligned.

To position the track cues:

1. Return to the **cp_captions.vtt** file in your editor.
2. After the time intervals for the Title and Subtitle cues, insert the attributes:

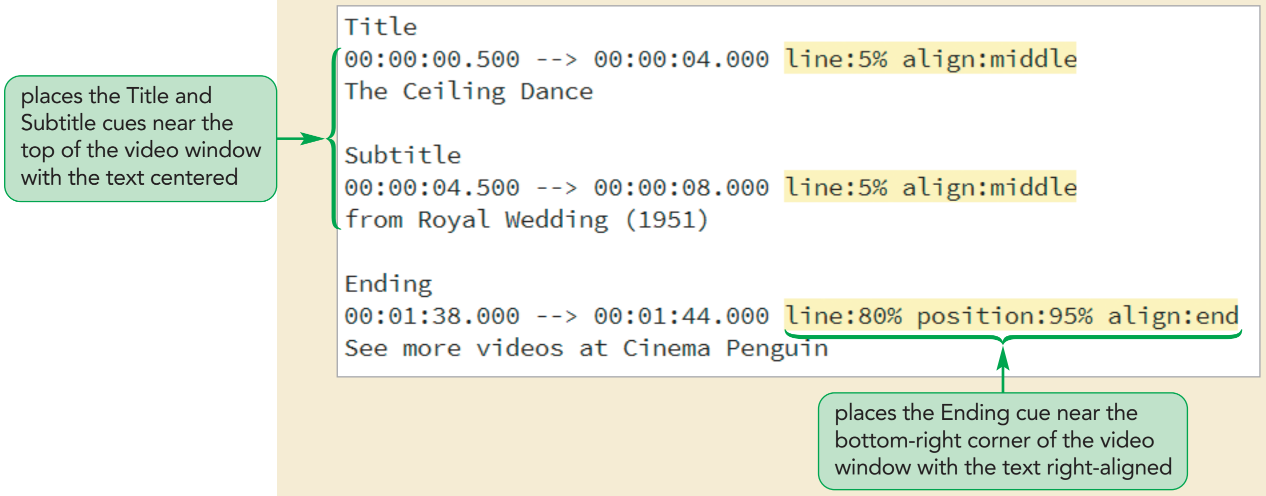
   ```
   line:5% align:middle
   ```

3. After the time interval for the Ending cue, insert the attributes:

   ```
   line:80% position:95% align:end
   ```

   Figure 8-24 highlights the attributes for each cue within the track.

Figure 8-24 **Placing the cue text**



```
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 Title and Subtitle cues near the top of the video window with the text centered
places the Ending cue near the bottom-right corner of the video window with the text right-aligned

4. Save your changes to the file and then reload the cp_royal.html file in your browser. If you are using Google Chrome or Opera, upload your files to a web server for testing.

5. Play the video clip and verify that the Title and Subtitle cues appear centered near the top of the video window and the Ending cue appears near the bottom-right corner of the window.

Trouble? If the cues have not changed position, check your code against the code in Figure 8-24. Make sure you have not entered a space between the attribute name and the attribute value or placed the values on a new line. At the time of this writing, Microsoft Edge and Internet Explorer do not support positioning the track cues.

The WebVTT settings do not include styles to format the appearance of the cue text, but you can create such styles using CSS. Note that browser support for such styles is mixed; therefore you should not make these styles crucial to users' interpretation of the video cues text.

Applying Styles to Track Cues

CSS supports the following `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 the `background`, `color`, `font`, `opacity`, `outline`, `text-decoration`, `text-shadow`, `visibility`, and `white-space` properties. For example, the following style rule displays all cues in a 2em yellow serif font on a red background:

```
::cue {
   background: red;
   color: yellow;
   font: 2em serif;
}
```

Use the `cue` pseudo-element to change the format of the captions in the ceiling dance video to an orange color on a semi-transparent black background. Set the size and type of the caption text to a 1.2em serif font.

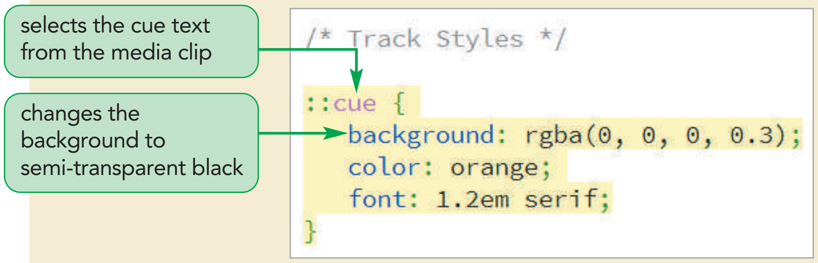
To format the cue text:

1. Return to the **cp_media.css** file in your editor and scroll down to the Track Styles section.
2. Add the following style rule:

```
::cue {
   background: rgba(0, 0, 0, 0.3);
   color: orange;
   font: 1.2em serif;
}
```

   Figure 8-25 highlights the style rule for cue text.

Figure 8-25 Applying styles to cue text



```
/* Track Styles */

::cue {
   background: rgba(0, 0, 0, 0.3);
   color: orange;
   font: 1.2em serif;
}
```

selects the cue text from the media clip
changes the background to semi-transparent black

3. Save your changes to the style sheet and then reload the cp_royal.html file in your browser. If you are using Google Chrome or Opera, upload your files to a web server for testing.
4. Play the video clip and verify that the captions appear in a serif orange font on a semi-transparent black background.

   Trouble? At the time of this writing, only Google Chrome and Opera support styles that modify the color and typeface of the cue text. Safari supports styles to change the font size. Internet Explorer and Firefox do not support any cue styles.

The `cue` pseudo-element selects all of the cue text in the media clip. To format specific cues or text strings within a cue you identify sections of the cue text using the following markup tags:

- `<i></i>` for italicized text
- `<b></b>` for bold-faced text
- `<u></u>` for underlined text
- `<span></span>` to mark spans of text
- `<ruby></ruby>` to mark ruby text
- `<rt></rt>` to mark ruby text

TIP

Ruby text refers to annotative characters placed above or to the right of other characters and is often used with Chinese or Japanese symbols.

For example, the following code italicizes the name of the website using the `<i></i>` tag

```
Ending
01:38.000 --> 01:44.000
See more videos at <i>Cinema Penguin</i>
```

WebVTT also supports tags that are not part of the HTML library, such as the following `<c></c>` tag used to mark text strings belonging to a particular class

```
<c.classname></c>
```

where ***classname*** defines the class.

And for captions that distinguish between one voice and another, WebVTT supports the following `<v></v>` tag

```
<v name></v>
```

where ***name*** is the name of the voice associated with the caption. The following track shows how to apply the `<c></c>` and `<v></v>` tags to mark scenes and the voices of Bernardo and Francisco from the opening of *Hamlet*.

```
Cue1
00:00.000 --> 00:05.000
<c.scene>Elsinore. A platform before the castle</c>

Cue2
00:05.500 --> 00:12.000
<v Bernardo>Who's there?</v>
<v Francisco>Nay, answer me: stand, and unfold yourself.</v>

Cue3
00:12.500 --> 00:18.000
<v Bernardo>Long live the king!</v>
<v Francisco>Bernardo?</v>
```

For cues based on their class name, add the class name to the `cue` pseudo-element as follows

```
::cue(.classname) {
   style rules
}
```

where ***classname*** is the class marked within in the `<c></c>` tag.

To format voice text, use the style rule

```
::cue(v[voice=name]) {
   style rules
}
```

where ***name*** is the name assigned in the `<v></v>` tag.

Thus, the following style rules display the scene and voices from the opening scene in *Hamlet* using red for the scene direction text, blue for Bernardo's voice, and green for Francisco's voice:

```
::cue(.scene) {color: red;}
::cue(v[voice=Bernardo]) {color: blue;}
::cue(v[voice=Francisco]) {color: green;}
```

Maxine suggests that the opening Title caption be displayed in a larger font with a drop shadow and that the website in the ending cue be italicized. Mark the caption text using the `<c></c>` tag belonging to the Main class and italicize the name of the website. Then, add the appropriate style rule to your style sheet.

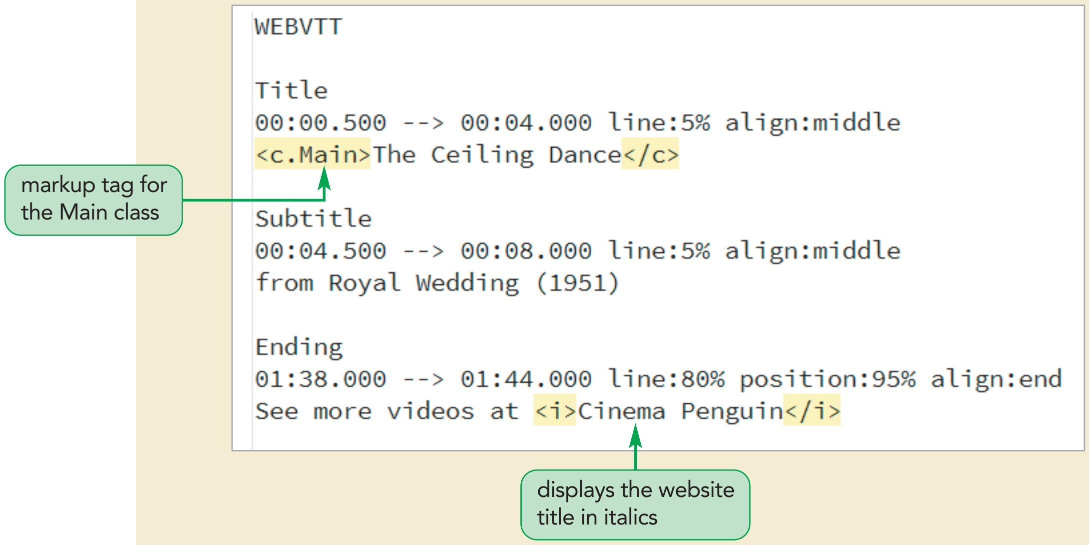
To apply styles to cue text:

1. Return to the **cp_captions.vtt** file in your editor.
2. Enclose the cue text for the Title cue within **<c>** tags with the class name **Main**.

3. Go to the Ending cue and enclose *Cinema Penguin* within opening and closing <i> tags.

   Figure 8-26 highlights the revised code in the file.

Figure 8-26 **Applying a class to cue text**



```
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>
```

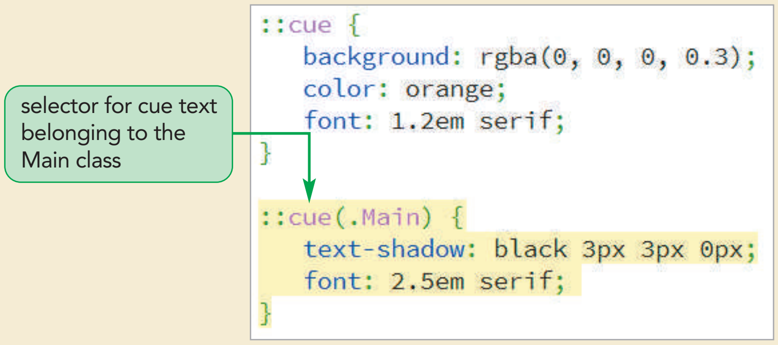
markup tag for the Main class
displays the website title in italics

4. Save your changes to the file and then return to the **cp_media.css** in your editor.
5. Scroll to the bottom of the file and add the following style rule:

```
::cue(.Main) {
   text-shadow: black 3px 3px 0px;
   font: 2.5em serif;
}
```

   Figure 8-27 highlights the style rule for cues belonging to the Main class.

Figure 8-27 **Styling cues based on class name**

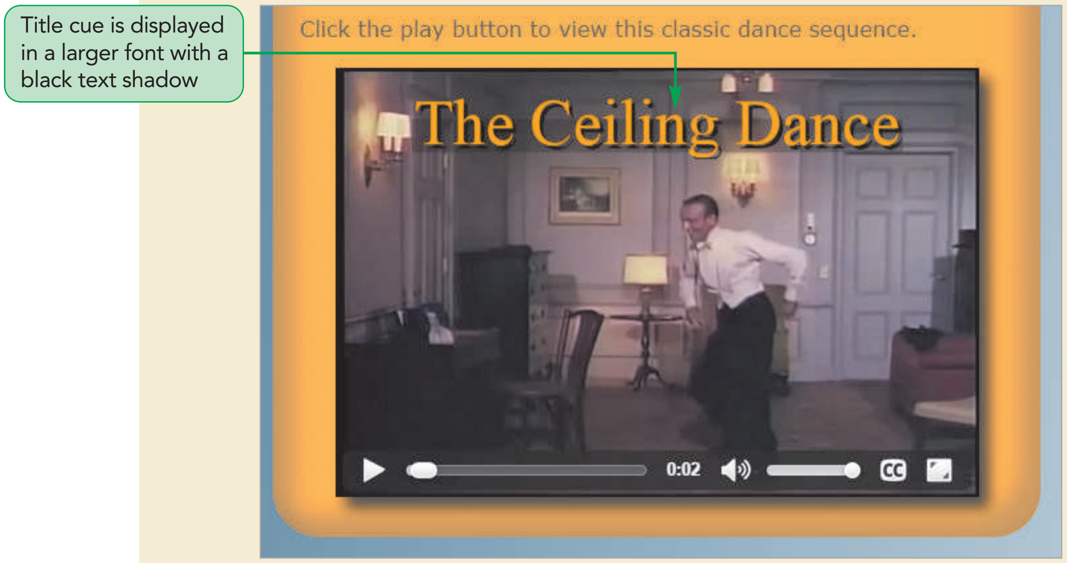


```
::cue {
   background: rgba(0, 0, 0, 0.3);
   color: orange;
   font: 1.2em serif;
}

::cue(.Main) {
   text-shadow: black 3px 3px 0px;
   font: 2.5em serif;
}
```

6. Save your changes to the file and then reload the cp_royal.html file in your browser. If you are using Google Chrome or Opera, upload your files to a web server for testing.
7. Play the ceiling dance video and note that the first caption is displayed in a larger font with a black drop shadow. See Figure 8-28.

Figure 8-28 **Formatted text from the Title cue**


Title cue is displayed in a larger font with a black text shadow
Click the play button to view this classic dance sequence.
The Ceiling Dance
0:02

Trouble? At the time of this writing, only Google Chrome and Opera support all of the text styles used to format captions. Safari will apply the larger font size. Internet Explorer and Firefox do not support styling caption text.

Maxine likes your work in creating and formatting the ceiling dance video clip. However, she would like to review other options for embedding multimedia content on her website.

Using Third-Party Video Players

Prior to the widespread adoption of HTML5 for embedded video, browsers used plug-ins using the following `object` element

```
<object attributes>
   parameters
</object>
```

where *`attributes`* define the object and *`parameters`* are values passed to the object controlling the object's appearance and actions. The `object` element, which replaced the `embed` element introduced in the last session, could be used for any type of content—from sound and video clips to graphic images, PDF files, and even the content of other web pages.

The parameters of the object are defined using the following `param` element

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

where the *`name`* is the name of the parameter and the *`value`* is the parameter's value. There is no standard list of parameter names and values because they are based on the plug-in used to display the object. For example, the following `object` element could be used in place of the `embed` element you studied in the first session to insert the cp_overture.mp3 audio clip:

```
<object data="overture.mp3" type="audio/mp3"
        height="20" width="250">
   <param name="src" value="cp_overture.mp3" />
</object>
```

The presumption is that content of the audio/mp3 mime-type would be associated with a plug-in that the browser would run to play the audio clip using the `src` and `value` attributes of the `param` element. Thus, the page's author would have to know how to work with the individual plug-ins and provide workarounds for the different plug-ins the user may have installed.

Exploring the Flash Player

Perhaps the most-used plug-in for video playback was the Adobe Flash player, which was 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>
```

where *`url`* is the location and filename of the file containing the Flash video, and *`parameters`* are the other `param` elements that manage the appearance and actions of the player. Notice that you must always include at least the movie parameter to identify the video file to be played in the Flash player. Figure 8-29 lists some of the other parameters recognized by the Adobe Flash Player.

Figure 8-29 Parameters of the Flash player

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

The Flash player can act as a fallback for older browsers that don't support HTML5 by nesting the `object` element within the `audio` or `video` element. The following code demonstrates how the Flash video from the cp_dance.swf file can be set as a fallback video for browsers that can't play either the cp_dance.mp4 or cp_dance.webm files using a native media player:

```
<video controls>
   <source src="cp_dance.mp4" type="video/mp4" />
   <source src="cp_dance.webm" type="video/webm" />
   <object data="cp_dance.swf"
           type="application/x-shockwave-flash"
           width="320" height="240">
      <param name="movie" value="cp_dance.swf" />
      <param name="quality" value="high" />
      <p>You must have the Flash Player to play
         the video clip</p>
   </object>
</video>
```

TIP

To hide the Flash player, set the width and height values to 0.

Note that within the `object` element is a final fallback message for users with browsers that cannot play the video clip in any of the formats.

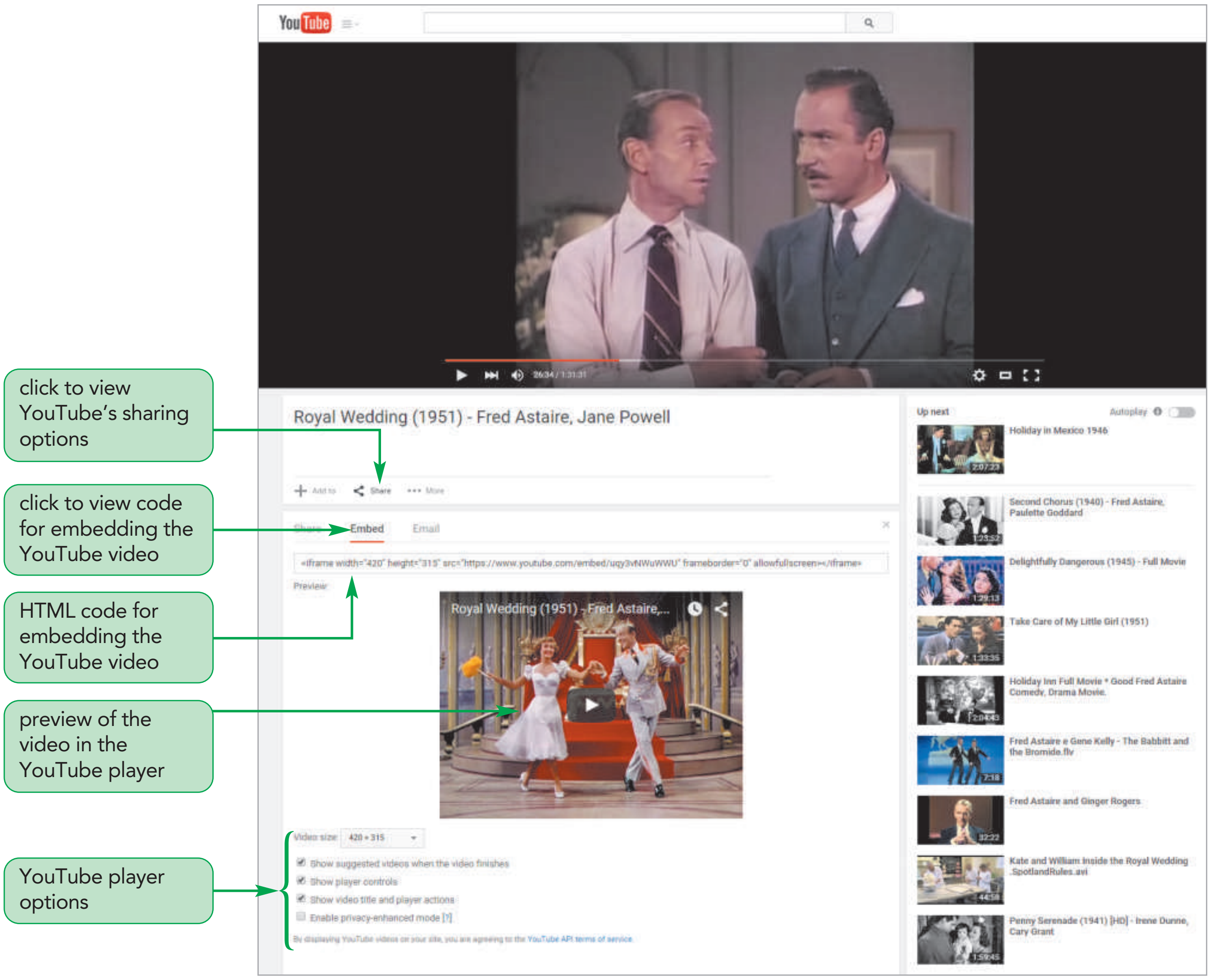
Maxine decides against providing a Flash version of the ceiling dance movie because she feels that the browser support for HTMl5 video is sufficient for users of her website.

Embedding Videos from YouTube

The biggest supplier of online videos is YouTube with over 1 billion users and 4 billion views per day. YouTube videos are easy to embed in your web page using YouTube's HTML5 video player, which supports a wide variety of video formats and codecs including H.264, WebM VP8, and WebM VP9.

To share a YouTube video, bring up the video on the YouTube site and click the Share button below the video player. 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. To embed the video within your website, click Embed, which brings up a preview of the embedded player and the HTML code you need to add to your web page. Figure 8-30 shows the embed options on the YouTube website for the full movie version of *Royal Wedding*.

Figure 8-30 **Sharing a YouTube video**


click to view YouTube's sharing options
click to view code for embedding the YouTube video
HTML code for embedding the YouTube video
preview of the video in the YouTube player
YouTube player options
Royal Wedding (1951) - Fred Astaire, Jane Powell
Add to
Share
More
Share
Embed
Email
Preview
Royal Wedding (1951) - Fred Astaire,...
Up next
Autoplay

```
<iframe width="420" height="315" src="https://www.youtube.com/embed/uqy3vNWuWWU" frameborder="0" allowfullscreen></iframe>
```

Video size: 420 × 315

- [✓] Show suggested videos when the video finishes
- [✓] Show player controls
- [✓] Show video title and player actions
- [ ] Enable privacy-enhanced mode [?]

By displaying YouTube videos on your site, you are agreeing to the YouTube API terms of service.

Source: YouTube/Viewster

The general code for the embedded player is

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

where *url* provides the link to the YouTube video, while the `width` and `height` attributes define the dimensions of the player embedded on your web page. The `frameborder` attribute sets the width of the border around the player in pixels. The `allowfullscreen` attribute allows the user to play the video in fullscreen mode. Finally, the **`iframe` element** itself is used to mark **inline frames**, which are windows embedded within the web page that display the content of another page or Internet resource. Please note that any videos submitted to YouTube are still subject to copyright restrictions and might be removed if those restrictions are violated.

HTML5 Video Players

Rather than using your browser's native video player, you can use one of the many commercial and free HTML5 video players on the market. Unlike plug-ins, which are applications distinct from your browser, an HTML5 video player works within your browser with CSS and JavaScript files to present a customizable player that can be adapted to the needs of your business or organization. The YouTube player is one example of an HTML5 player in which YouTube provides both the player and a hosting service for the video content. Some of the other popular HTML5 video players include the following:

- **JWPlayer** (*www.jwplayer.com*) A popular commercial player that supports both HTML5 and Flash video. A free non-commercial version is also available.
- **Video.js** (*www.videojs.com*) A free player that works with the popular WordPress HTML framework. Video.js is extremely customizable and provides support for captions and subtitles.
- **MediaElement.js** (*mediaelementjs.com*) An HTML5 audio and video player with support for Flash and Microsoft Silverlight.
- **Projekktor** (*www.projekktor.com*) A free and open source video (and audio) player. Projekktor also includes support for embedded playlists.
- **Flowplayer** (*flowplayer.org*) Originally marketed as a Flash player, Flowplayer is a commercially licensed audio and video player, payable as a one-time fee for perpetual use.

Any of these video players and many others can cover your basic needs. As your company or organization grows, you may find that you will need the professional service and quality of a licensed-based player with either a one-time or annual subscription.

PROSKILLS

Problem Solving: Tips for Effective Web Video

Web video is one of the most important mediums for conveying information and advertising products and services. With inexpensive hardware and sophisticated video editing software, almost anyone can be a movie producer with free, instant distribution available through the web. However, this also means you have a lot of competition, making it hard to get noticed; it is essential that your videos be polished and professional. Here are some things to keep in mind when creating a video for the web:

- *Keep it short.* Studies have shown that web users have an attention span of about 4 seconds. If they don't receive valuable information within that time, they will go to a different site. This means your video must get to the point quickly and keep users' attention. Also recognize that most users will not watch your entire video, so make your key points early.
- *Keep the image simple.* Your video will probably be rendered in a tiny frame, so make your content easier to view by shooting close-ups. Avoid wide-angle shots that will make your subject even smaller to the user's eye. Avoid complex backgrounds and distracting color schemes.
- *Keep the human element.* Eye-tracking studies have shown that people naturally gravitate to human faces for information and emotional content. Use tight shots in which the narrator speaks directly into the camera.
- *Use effective lighting.* Light should be projected onto the subject. Avoid relying solely on overhead lights, which can create distracting facial shadows. Video compression can result in loss of detail; thus, make sure you use bright lighting on key areas to highlight and focus users' attention on the important images in your video.
- *Follow the rule of thirds.* Avoid static layouts by imagining the frame divided into a 3 × 3 grid. Balance items of interest along the lines of the intersection in the grid rather than centered within the frame. If interviewing a subject, leave ample headroom at the top of the frame.
- *Avoid pans and zooms.* Excessive panning and zooming can make your video appear choppy and distorted, and unnecessary movement slows down the video stream.

Finally, consider investing in professional video services that can storyboard an idea for you and that have the experience and expertise to create a finished product that will capture and keep the attention of users and customers.