**http://www.whatwg.org/specs/web-apps/current-work/multipage/the-video-element.html#video**

### 4.8.6 The `video` element

**Categories**

> Flow content.
> Phrasing content.
> Embedded content.
> If the element has a `controls` attribute: Interactive content.
> Palpable content.

**Contexts in which this element can be used:**

> Where embedded content is expected.

**Content model:**

> If the element has a `src` attribute: zero or more `track` elements, then transparent, but with no media element descendants.
> If the element does not have a `src` attribute: zero or more `source` elements, then zero or more `track` elements, then transparent, but with no media element descendants.

**Content attributes:**

> Global attributes
> `src`
> `crossorigin`
> `poster`
> `preload`
> `autoplay`
> `mediagroup`
> `loop`
> `muted`
> `controls`
> `width`
> `height`

**DOM interface:**

```
interface HTMLVideoElement : HTMLMediaElement {
           attribute unsigned long width;
           attribute unsigned long height;
  readonly attribute unsigned long videoWidth;
  readonly attribute unsigned long videoHeight;
           attribute DOMString poster;
};
```

A `video` element is used for playing videos or movies, and audio files with captions.

Content may be provided inside the `video` element. User agents should not show this content to the user; it is intended for older Web browsers which do not support `video`, so that legacy video plugins can be tried, or to show text to the users of these older browsers informing them of how to access the video contents.

***In particular, this content is not intended to address accessibility concerns. To make video content accessible to the blind, deaf, and those with other physical or cognitive disabilities, a variety of features are available. Captions can be provided, either embedded in the video stream or as external files using the `track` element. Sign-language tracks can be provided, again either embedded in the video stream or by synchronizing multiple `video` elements using the `mediagroup` attribute or a `MediaController` object. Audio descriptions can be provided, either as a separate track embedded in the video stream, or a separate audio track in an `audio` element slaved to the same controller as the `video` element(s), or in text form using a WebVTT file referenced using the `track` element and synthesized into speech by the user agent. WebVTT can also be used to provide chapter titles. For users who would rather not use a media element at all, transcripts or other textual alternatives can be provided by simply linking to them in the prose near the `video` element. [WEBVTT]***

The `video` element is a media element whose media data is ostensibly video data, possibly with associated audio data.

The `src`, `preload`, `autoplay`, `mediagroup`, `loop`, `muted`, and `controls` attributes are the attributes common to all media elements.

The **`poster`** attribute gives the address of an image file that the user agent can show while no video data is available. The attribute, if present, must contain a valid non-empty URL potentially surrounded by spaces.

If the specified resource is to be used, then, when the element is created or when the `poster` attribute is set, changed, or removed, the user agent must run the following steps to determine the element's **poster frame**:

1. If there is an existing instance of this algorithm running for this `video` element, abort that instance of this algorithm without changing the poster frame.

2. If the `poster` attribute's value is the empty string or if the attribute is absent, then there is no poster frame; abort these steps.

3. Resolve the `poster` attribute's value relative to the element. If this fails, then there is no poster frame; abort these steps.

4. Fetch the resulting absolute URL, from the element's `Document`'s origin. This must delay the load event of the element's document.

5. If an image is thus obtained, the poster frame is that image. Otherwise, there is no poster frame.

***The image given by the `poster` attribute, the poster frame, is intended to be a representative frame of the video (typically one of the first non-blank frames) that gives the user an idea of what the video is like.***

---

When no video data is available (the element's `readyState` attribute is either `HAVE_NOTHING`, or `HAVE_METADATA` but no video data has yet been obtained at all, or the element's `readyState` attribute is any subsequent value but the media resource does not have a video channel), the `video` element represents the poster frame.

When a `video` element is paused and the current playback position is the first frame of video, the element represents the poster frame, unless a frame of video has already been shown, in which case the element represents the frame of video corresponding to the current playback position.

When a `video` element is paused at any other position, and the media resource has a video channel, the element represents the frame of video corresponding to the current playback position, or, if that is not yet available (e.g. because the video is seeking or buffering), the last frame of the video to have been rendered.

When a `video` element whose media resource has a video channel is potentially playing, it represents the frame of video at the continuously increasing "current" position. When the current playback position changes such that the last frame rendered is no longer the frame corresponding to the current playback position in the video, the new frame must be rendered. Similarly, any audio associated with the media resource must, if played, be played synchronized with the current playback position, at the element's effective media volume.

When a `video` element whose media resource has a video channel is neither potentially playing nor paused (e.g. when seeking or stalled), the element represents the last frame of the video to have been rendered.

***Which frame in a video stream corresponds to a particular playback position is defined by the video stream's format.***

The `video` element also represents any text track cues whose text track cue active flag is set and whose text track is in the showing or showing by default modes.

In addition to the above, the user agent may provide messages to the user (such as "buffering", "no video loaded", "error", or more detailed information) by overlaying text or icons on the video or other areas of the element's playback area, or in another appropriate manner.

User agents that cannot render the video may instead make the element represent a link to an external video playback utility or to the video data itself.

When a `video` element's media resource has a video channel, the element provides a paint source whose width is the media resource's intrinsic width, whose height is the media resource's intrinsic height, and whose appearance is the frame of video corresponding to the current playback position, if that is available, or else (e.g. when the video is seeking or buffering) its previous appearance, if any, or else (e.g. because the video is still loading the first frame) blackness.

---

***video* . `videoWidth`**

***video* . `videoHeight`**

These attributes return the intrinsic dimensions of the video, or zero if the dimensions are not known.

The **intrinsic width** and **intrinsic height** of the media resource are the dimensions of the resource in CSS pixels after taking into account the resource's dimensions, aspect ratio, clean aperture, resolution, and so forth, as defined for the format used by the resource. If an anamorphic format does not define how to apply the aspect ratio to the video data's dimensions to obtain the "correct" dimensions, then the user agent must apply the ratio by increasing one dimension and leaving the other unchanged.

The **`videoWidth`** IDL attribute must return the intrinsic width of the video in CSS pixels. The **`videoHeight`** IDL attribute must return the intrinsic height of the video in CSS pixels. If the element's `readyState` attribute is `HAVE_NOTHING`, then the attributes must return 0.

The `video` element supports dimension attributes.

In the absence of style rules to the contrary, video content should be rendered inside the element's playback area such that the video content is shown centered in the playback area at the largest possible size that fits completely within it, with the video content's aspect ratio being preserved. Thus, if the aspect ratio of the playback area does not match the aspect ratio of the video, the video will be shown letterboxed or pillarboxed. Areas of the element's playback area that do not contain the video represent nothing.

***In user agents that implement CSS, the above requirement can be implemented by using the style rule suggested in the rendering section.***

The intrinsic width of a `video` element's playback area is the intrinsic width of the video resource, if that is available; otherwise it is the intrinsic width of the poster frame, if that is available; otherwise it is 300 CSS pixels.

The intrinsic height of a `video` element's playback area is the intrinsic height of the video resource, if that is available; otherwise it is the intrinsic height of the poster frame, if that is available; otherwise it is 150 CSS pixels.

---

User agents should provide controls to enable or disable the display of closed captions, audio description tracks, and other additional data associated with the video stream, though such features should, again, not interfere with the page's normal rendering.

User agents may allow users to view the video content in manners more suitable to the user (e.g. full-screen or in an independent resizable window). As for the other user interface features, controls to enable this should not interfere with the page's

normal rendering unless the user agent is exposing a user interface. In such an independent context, however, user agents may make full user interfaces visible, with, e.g., play, pause, seeking, and volume controls, even if the `controls` attribute is absent.

User agents may allow video playback to affect system features that could interfere with the user's experience; for example, user agents could disable screensavers while video playback is in progress.

---

The **`poster`** IDL attribute must reflect the `poster` content attribute.

This example shows how to detect when a video has failed to play correctly:

```
<script>
 function failed(e) {
   // video playback failed - show a message saying why
   switch (e.target.error.code) {
     case e.target.error.MEDIA_ERR_ABORTED:
       alert('You aborted the video playback.');
       break;
     case e.target.error.MEDIA_ERR_NETWORK:
       alert('A network error caused the video download to fail
part-way.');
       break;
     case e.target.error.MEDIA_ERR_DECODE:
       alert('The video playback was aborted due to a corruption
problem or because the video used features your browser did not
support.');
       break;
     case e.target.error.MEDIA_ERR_SRC_NOT_SUPPORTED:
       alert('The video could not be loaded, either because the
server or network failed or because the format is not supported.');
       break;
     default:
       alert('An unknown error occurred.');
       break;
   }
 }
</script>
<p><video src="tgif.vid" autoplay controls
onerror="failed(event)"></video></p>
<p><a href="tgif.vid">Download the video file</a>.</p>
```

### 4.8.7 The `audio` element

**Categories**

Flow content.
Phrasing content.
Embedded content.
If the element has a `controls` attribute: Interactive content.
If the element has a `controls` attribute: Palpable content.

**Contexts in which this element can be used:**

> Where embedded content is expected.

**Content model:**

> If the element has a `src` attribute: zero or more `track` elements, then transparent, but with no media element descendants.
> If the element does not have a `src` attribute: zero or more `source` elements, then zero or more `track` elements, then transparent, but with no media element descendants.

**Content attributes:**

> Global attributes
> `src`
> `crossorigin`
> `preload`
> `autoplay`
> `mediagroup`
> `loop`
> `muted`
> `controls`

**DOM interface:**

```
[NamedConstructor=Audio(),
 NamedConstructor=Audio(DOMString src)]
interface HTMLAudioElement : HTMLMediaElement {};
```

An `audio` element represents a sound or audio stream.

Content may be provided inside the `audio` element. User agents should not show this content to the user; it is intended for older Web browsers which do not support `audio`, so that legacy audio plugins can be tried, or to show text to the users of these older browsers informing them of how to access the audio contents.

***In particular, this content is not intended to address accessibility concerns. To make audio content accessible to the deaf or to those with other physical or cognitive disabilities, a variety of features are available. If captions or a sign language video are available, the `video` element can be used instead of the `audio` element to play the audio, allowing users to enable the visual alternatives. Chapter titles can be provided to aid navigation, using the `track` element and a WebVTT file. And, naturally, transcripts or other textual alternatives can be provided by simply linking to them in the prose near the `audio` element. [WEBVTT]***

The `audio` element is a media element whose media data is ostensibly audio data.

The `src`, `preload`, `autoplay`, `mediagroup`, `loop`, `muted`, and `controls` attributes are the attributes common to all media elements.

When an `audio` element is potentially playing, it must have its audio data played synchronized with the current playback position, at the element's effective media volume.

When an audio element is not potentially playing, audio must not play for the element.

***audio* = new Audio( [ *url* ] )**

Returns a new audio element, with the src attribute set to the value passed in the argument, if applicable.

Two constructors are provided for creating HTMLAudioElement objects (in addition to the factory methods from DOM Core such as createElement()): **Audio()** and **Audio(*src*)**. When invoked as constructors, these must return a new HTMLAudioElement object (a new audio element). The element must have its preload attribute set to the literal value "auto". If the *src* argument is present, the object created must have its src content attribute set to the provided value, and the user agent must invoke the object's resource selection algorithm before returning. The element's document must be the active document of the browsing context of the Window object on which the interface object of the invoked constructor is found.

### 4.8.8 The source element

**Categories**

None.

**Contexts in which this element can be used:**

As a child of a media element, before any flow content or track elements.

**Content model:**

Empty.

**Content attributes:**

Global attributes
src
type
media

**DOM interface:**

```
interface HTMLSourceElement : HTMLElement {
           attribute DOMString src;
           attribute DOMString type;
           attribute DOMString media;
};
```

The source element allows authors to specify multiple alternative media resources for media elements. It does not represent anything on its own.

The **src** attribute gives the address of the media resource. The value must be a valid non-empty URL potentially surrounded by spaces. This attribute must be present.

***Dynamically modifying a source element and its attribute when the element is already inserted in a video or audio element will have no effect. To change what is playing, just use the src attribute on the media element directly, possibly making use of the canPlayType() method to pick from amongst available***

***resources. Generally, manipulating `source` elements manually after the document has been parsed is an unncessarily complicated approach.***

The **`type`** attribute gives the type of the media resource, to help the user agent determine if it can play this media resource before fetching it. If specified, its value must be a valid MIME type. The `codecs` parameter, which certain MIME types define, might be necessary to specify exactly how the resource is encoded. [RFC4281]

The following list shows some examples of how to use the `codecs=` MIME parameter in the `type` attribute.

**H.264 Constrained baseline profile video (main and extended video compatible) level 3 and Low-Complexity AAC audio in MP4 container**

```
<source src='video.mp4' type='video/mp4; codecs="avc1.42E01E,
mp4a.40.2"'>
```

**H.264 Extended profile video (baseline-compatible) level 3 and Low-Complexity AAC audio in MP4 container**

```
<source src='video.mp4' type='video/mp4; codecs="avc1.58A01E,
mp4a.40.2"'>
```

**H.264 Main profile video level 3 and Low-Complexity AAC audio in MP4 container**

```
<source src='video.mp4' type='video/mp4; codecs="avc1.4D401E,
mp4a.40.2"'>
```

**H.264 'High' profile video (incompatible with main, baseline, or extended profiles) level 3 and Low-Complexity AAC audio in MP4 container**

```
<source src='video.mp4' type='video/mp4; codecs="avc1.64001E,
mp4a.40.2"'>
```

**MPEG-4 Visual Simple Profile Level 0 video and Low-Complexity AAC audio in MP4 container**

```
<source src='video.mp4' type='video/mp4; codecs="mp4v.20.8,
mp4a.40.2"'>
```

**MPEG-4 Advanced Simple Profile Level 0 video and Low-Complexity AAC audio in MP4 container**

```
<source src='video.mp4' type='video/mp4; codecs="mp4v.20.240,
mp4a.40.2"'>
```

**MPEG-4 Visual Simple Profile Level 0 video and AMR audio in 3GPP container**

```
<source src='video.3gp' type='video/3gpp; codecs="mp4v.20.8,
samr"'>
```

**Theora video and Vorbis audio in Ogg container**

```
<source src='video.ogv' type='video/ogg; codecs="theora,
vorbis"'>
```

**Theora video and Speex audio in Ogg container**

```
<source src='video.ogv' type='video/ogg; codecs="theora,
speex"'>
```

**Vorbis audio alone in Ogg container**

```
<source src='audio.ogg' type='audio/ogg; codecs=vorbis'>
```

**Speex audio alone in Ogg container**

```
<source src='audio.spx' type='audio/ogg; codecs=speex'>
```

**FLAC audio alone in Ogg container**

```
<source src='audio.oga' type='audio/ogg; codecs=flac'>
```

**Dirac video and Vorbis audio in Ogg container**

```
<source src='video.ogv' type='video/ogg; codecs="dirac,
vorbis"'>
```

The **`media`** attribute gives the intended media type of the media resource, to help the user agent determine if this media resource is useful to the user before fetching it. Its value must be a valid media query.

The default, if the `media` attribute is omitted, is "`all`", meaning that by default the media resource is suitable for all media.

If a `source` element is inserted as a child of a media element that has no `src` attribute and whose `networkState` has the value `NETWORK_EMPTY`, the user agent must invoke the media element's resource selection algorithm.

The IDL attributes **`src`**, **`type`**, and **`media`** must reflect the respective content attributes of the same name.

If the author isn't sure if the user agents will all be able to render the media resources provided, the author can listen to the `error` event on the last `source` element and trigger fallback behavior:

```
<script>
 function fallback(video) {
   // replace <video> with its contents
   while (video.hasChildNodes()) {
     if (video.firstChild instanceof HTMLSourceElement)
       video.removeChild(video.firstChild);
     else
       video.parentNode.insertBefore(video.firstChild, video);
   }
   video.parentNode.removeChild(video);
 }
</script>
<video controls autoplay>
 <source src='video.mp4' type='video/mp4; codecs="avc1.42E01E,
mp4a.40.2"'>
 <source src='video.ogv' type='video/ogg; codecs="theora, vorbis"'
         onerror="fallback(parentNode)">
 ...
</video>
```